Transducing Markov Sequences

Extended Abstract

Benny Kimelfeld
IBM Research–Almaden
San Jose, CA 95120, USA
kimelfeld@us.ibm.com

Christopher Ré
University of Wisconsin-Madison
Madison, WI 53706, USA
chrisre@cs.wisc.edu

ABSTRACT

A Markov sequence is a basic statistical model representing uncertain sequential data, and it is used within a plethora of applications, including speech recognition, image processing, computational biology, radio-frequency identification (RFID), and information extraction. The problem of querying a Markov sequence is studied under the conventional semantics of querying a probabilistic database, where queries are formulated as finite-state transducers. Specifically, the complexity of two main problems is analyzed. The first problem is that of computing the confidence (probability) of an answer. The second is the enumeration of the answers in the order of decreasing confidence (with the generation of the top-k answers as a special case), or in an approximate order thereof. In particular, it is shown that enumeration in any sub-exponential-approximate order is generally intractable (even for some fixed transducers), and a matching upper bound is obtained through a proposed heuristic. Due to this hardness, a special consideration is given to restricted (yet common) classes of transducers that extract matches of a regular expression (subject to prefix and suffix constraints), and it is shown that these classes are, indeed, significantly more tractable.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Systems—*Query processing*

General Terms

Theory, Algorithms

Keywords

Markov sequences, hidden Markov models, transducers, probabilistic databases, ranked query evaluation, enumeration

PODS'10, June 6–11, 2010, Indianapolis, Indiana, USA.

1. INTRODUCTION

A large fraction of the world's raw data is sequential and low-level [18], such as hand-written forms, audio feeds, and video feeds. And while this low-level data is rich in information, the data is difficult for many applications to use directly, since applications often need higher-level information, such as the ascii text corresponding to the low-level image of hand writing in a form. To make higher-level information available to applications, a popular approach is to use a statistical model, which infers the higher-level information from the lower-level, raw data. One popular statistical model to extract sequential data from raw, unstructured data is the *hidden Markov model* (HMM) [46]. Abstractly, an HMM takes as input a sequence of *observations* and then produces as output a probability distribution over a sequence of *hidden states*. HMMs can be used to extract structure that is useful in a diverse set of applications: In RFID applications [39, 47], the observations are the low-level antenna sightings, and the hidden states are sequences of higher-level locations, such as rooms or hallways. In speech applications [21, 40, 46, 52], the observations are acoustic signals, and the hidden states are sequences of words or phonemes. There are other data-rich applications that use HMMs as well: sequence matching in biological data [13, 20], optical character recognition [4], and image classification [17].

In this paper, we consider *Markov sequences*. A Markov sequence is a chain of random variables that hold the *Markov property*—a random variable is independent of its history given its predecessor. Markov sequences represent the output of statistical models such as HMMs; in particular, the distribution encoded by an HMM and a sequence of observations can be efficiently translated into a Markov sequence.[1] In our setting, the proper translation into a Markov sequence has already taken place. In particular, we do not process the observations of an HMM (e.g., signal readings) directly, but rather the implied Markov sequence. There are other statistical models, notably Chain CRFS [37], that produce output that can be modeled by Markov sequences (in spite of requiring different translations). This opens up even more applications for Markov sequences (e.g., more sophisticated information extraction [37, 51]).

LAHAR [39,40,47] is a *Markov-sequence database* that supports query processing over a collection of Markov sequences. In this paper, we study the complexity of querying a single Markov sequence with the goal of introducing strong querying capabilities into LAHAR. As in many probabilistic

[1]See the extended version [31] for a discussion on the details of this translation.

databases [8, 26, 30, 48, 49], queries in LAHAR are formulated as if the data were precise (i.e., deterministic); in contrast to standard deterministic databases, each answer to a query is assigned a *confidence value*, which is the probability of obtaining that answer when querying a random possible world. The query language we study is based on *finite-state transducers* (or just *transducers* for short), which have been used for querying both strings [1,2] and trees [41,42] (e.g., XML). A transducer is essentially an automaton that emits output strings throughout its run on the input string (in our case, a random possible world of the Markov sequence). In this work, we study transducers that hold the property of *deterministic emission*.[2] Our experience with LAHAR leads us to view string transducers as a natural means of expressing queries over Markov sequences.

In our running example, taken from RFID, we consider a scenario where sensors are installed in different locations of a hospital, and transmitters are attached to medical equipment (like crash carts) and personnel. Here, one Markov sequence may represent the locations of a particular crash cart at different times, and another the location of a particular doctor. Note that an actual location of a transmitting object is typically uncertain. For example, physical limitations may lead to erroneous reads or, more commonly, missed readings. As another example, the locations of sensors can easily introduce ambiguity (e.g., sensors located near passages, or close sensors that simultaneously read the same signal). More subtly, the antenna readings themselves are low level, and there may be no one-to-one mapping to higher-level events (e.g., the same sequence could correspond to entering either Room 1 or Room 2). We consider a specific task, where we want to detect the sequence of rooms that a crash cart has visited (e.g., to detect a source of infection). So, we formulate a transducer that reads the locations, and whenever the crash cart changes rooms the transducer emits the number of the new room. This simple transducer already exhibits some of the challenges in query evaluation. First, there may be a huge number of possible answers (sequences of rooms) with different probabilities, and it may well be the case that most of the answers have a confidence that is too low to be of interest. Second, many possible worlds can result (i.e., be transduced into) the same answer, as such worlds may differ in the duration of staying at each room as well as the different sub-locations inside each room (e.g., the main area versus the restroom).

In probabilistic relational databases and probabilistic XML databases [8, 9, 30, 49], evaluating a query entails two basic steps: First, enumerate all *possible* answers (i.e., those having a nonzero confidence). Second, compute the confidence of each possible answer. Later on, we argue that this approach, separating answer enumeration from confidence computation, is too weak to meet our needs. Nevertheless, to explore the computational aspects of query evaluation, we analyze the complexity of the two steps in our specific context.

The first step, enumerating the possible answers, is often simply an algorithm for query evaluation over standard, precise data. Typically, the algorithm terminates in polynomial time if one fixes the query (i.e., this step is efficient under *data complexity*). In our case, the goal is find all the output strings that are transduced into with a nonzero probability. However, even if the transducer is held fixed, the number of possible (different) answers may be exponential in the length of the Markov sequence. Furthermore, holding the transducer fixed is not always realistically justified (e.g., as in the above crash-cart example). Thus, one cannot expect the set of possible answers to be generated in polynomial time. To properly measure complexity, we adopt yardsticks from *enumeration complexity* [24] that relate the running time to the size of the output. We show that one can enumerate all the possible answers in polynomial time under *input-and-output complexity* (that is, the running time is polynomial in the combined size of the input and the output). Even better, enumeration can be done with *polynomial delay* (between every two consecutive answers) and polynomial space.

For the second step, we consider the computation of the confidence of a given answer. In general, this problem is $\mathrm{FP}^{\#\mathrm{P}}$-complete (in particular, #P-hard), which immediately follows from known results [28]. We show, however, that this problem is tractable if the transducer is deterministic. At a high-level, one may expect to be able to apply a determinization technique, such as the *subset construction*, to achieve an evaluation that takes time exponential in the size of the transducer and polynomial in that of the Markov sequence. For transducers with *uniform emission* (i.e., all emitted strings have the same length), this approach succeeds. Surprisingly, if one does not assume such uniformity, then determinization does not help: we construct a *fixed* nondeterministic transducer, such that computing the confidence, given a Markov sequence and an answer, is $\mathrm{FP}^{\#\mathrm{P}}$-hard.

In many applications, we are interested not in all answers, but in those with higher confidence; thus, in the above two-step approach, most of the produced answers may be irrelevant. Moreover, since the number of answers for a transducer over a Markov sequence may be huge (i.e., exponential in the length of the Markov sequence), the cost of producing even one valuable answer may be prohibitively high. So, we set the more realistic goal of *ranked enumeration of answers*. Our gold-standard result would be that we could enumerate the answers in decreasing confidence (in particular, the answers with the highest probability should appear first) with polynomial delay. Later on, we show that, in general, this gold-standard result is unlikely, even under strong restrictions on the transducer. So, we propose a heuristic approach: instead of scoring an answer with its confidence, we score it with the probability of its best evidence, namely, the probability of the most likely possible world that is transduced into the answer. We show that with the new scoring applied, ranked enumeration is possible. An instrumental tool in achieving this enumeration is the top-k technique of Lawler [38], Murty [43] and Yen [59].

While the above heuristic may seem practically reasonable, its theoretic performance is very poor (albeit provably better than that of the arbitrary order). Formally, if the score is again taken to be the confidence, then our heuristic provides an *approximately ranked enumeration* [16, 32] with the approximation ratio $|\Sigma|^n$, where n is the length of the Markov sequence and Σ is the sequence alphabet. Quite remarkably, it turns out that our heuristic has a worst-case optimal performance. In particular, we show that unless P = NP, it is intractable to approximate the top answer (let alone enumerate in ranked order) within any sub-

[2]Formally, this means that an emitted string is completely determined by the (possibly nondeterministic) state transition. See Section 3 for the exact details.

exponential factor of the form $2^{n^{1-\delta}}$ (for any $\delta > 0$). Furthermore, this lower bound holds even if the transducer belongs to the very restricted class of *Mealy machines*.

Motivated by the above daunting hardness of approximation, and drawing from our experience with LAHAR, we look for practical, tractable subclasses of transducers. We observe that, very often, practical queries simply extract substrings of their input. So, in Section 5 we consider *substring projectors* (called *s-projectors* for short). Essentially, an s-projector extracts from a string substrings that match a DFA, such that the prefix and suffix of the string, surrounding the extracted substring, satisfy constraints (also given as DFAs). We also look at *indexed s-projectors*, which are s-projectors that output, in addition to the extracted substring, the indices of the substring inside the input string. Thus, while an s-projectors selects substrings, an indexed s-projector selects *occurrences* of these substrings.

Among other things, we show that indexed s-projectors have an evaluation in the *exact* order of decreasing confidence, with polynomial delay and polynomial space. This is done through a reduction to the problem of enumerating paths of a directed graph in increasing weight [14]. Furthermore, we show how this result can be used to obtain an n-approximate ranked enumeration for s-projectors (which is exponentially better than what can be done for general transducers, or even for Mealy machines). Whether this approximation ratio is tight remains an open problem; however, we show a $\sqrt{n}$ lower bound (actually, $n^{1/2-\delta}$ for all $\delta > 0$); in particular, this lower bound rules out the existence of a constant-factor (or logarithmic-factor) approximation.

We first give preliminaries (§2) and describe the formal setting (§3). We then describe our results for the general problem (§4), and our exploration of s-projectors (§5). Finally, we discuss related work (§6) and conclude (§7).

2. PRELIMINARIES

In this section, we describe the basic concepts that are used throughout the paper.

2.1 Strings and Automata

Let U be a set. By U^* we denote the set of all finite strings of elements of U, that is, sequences of the form $u_1 \cdots u_n$ where $u_i \in U$ for all $1 \leq i \leq n$. For a string $\mathbf{u} \in U^*$, we use $|\mathbf{u}|$ to denote the length $\mathbf{u}$ (here n), and we use ϵ to denote the empty string. For an integer $n \geq 0$, the set of all the strings $\mathbf{u} \in U^*$ of length n is denoted by U^n. If $\mathbf{u} = u_1 \cdots u_n$ is a string, and i, j and k are integers such that $1 \leq i \leq j \leq n$ and $i \leq k \leq n+1$, then $\mathbf{u}_{[i,j]}$ denotes the substring $u_i \cdots u_j$ of $\mathbf{u}$, and $\mathbf{u}_{[i,k)}$ denotes the substring $u_i \cdots u_{k-1}$. Note that if $i = j$, then $\mathbf{u}_{[i,j]}$ is u_i whereas $\mathbf{u}_{[i,j)}$ is the empty string ϵ.

A *nondeterministic finite automaton* (*NFA*) A is a tuple $\langle \Sigma_A, Q_A, q_A^0, F_A, \delta_A \rangle$ where Σ_A is a finite *alphabet*, Q_A is a finite set of *states*, $q_A^0 \in Q_A$ is the *initial state*, $F_A \subseteq Q_A$ is the set of *accepting states* and $\delta_A : Q_A \times \Sigma_A \to 2^{Q_A}$ (i.e., $\delta_A(q, s)$ is a subset of Q_A for all $q \in Q_A$ and $s \in \Sigma_A$) is the *transition function*. We usually do not specify the whole tuple describing an automaton A, but rather implicitly assume that A is the automaton $\langle \Sigma_A, Q_A, q_A^0, F_A, \delta_A \rangle$ (i.e., the automaton symbol is consistently used as a subscript).

Let A be an NFA. A *run* of A on a string $\mathbf{s} = s_1 \cdots s_n \in {\Sigma_A}^*$ is a mapping $\rho : \{1, \ldots, n\} \to Q_A$, such that $\rho(1) \in \delta_A(q_A^0, s_1)$ and $\rho(i) \in \delta_A(\rho(i-1), s_i)$ for all $2 \leq i \leq n$. The run ρ is *accepting* if $\rho(n) \in F_A$. We denote by $\mathcal{L}(A)$ the subset of Σ_A^* that comprises all the strings that are *accepted* by A (i.e., strings $\mathbf{s}$ such that an accepting run on $\mathbf{s}$ exists).

A *deterministic finite automaton* (*DFA*) is an NFA A, such that the image of δ_A comprises singletons; that is, $|\delta_A(q, s)| = 1$ for all $q \in Q_A$ and $s \in \Sigma_A$. It is well-known that NFAs and DFAs, as well as regular expressions, recognize the same class of languages, called *regular*. If A is a DFA, then we may abuse the notation and view δ as a function $Q_A \times \Sigma_A \to Q_A$ (rather than a function from $Q_A \times \Sigma_A$ to singleton sets of 2^{Q_A}).

2.2 Probability Spaces

All the probability spaces that we consider in this paper are finite. Formally, here a *probability space* is a pair (Ω, p), where Ω is a finite set and $p : \Omega \to [0, 1]$ is a function satisfying $\sum_{o \in \Omega} p(o) = 1$. We say that (Ω, p) is a probability space *over* Ω. By a slight abuse of notation, we may identify a probability space (Ω, p) with the probability space (Ω', p') if Ω' and p' contain Ω and p, respectively (and then p' is zero over $\Omega' \setminus \Omega$).

2.3 Enumeration of Answers

In an *enumeration problem*, the result for a given input is a (possibly large) set of *answers*. An example is the problem of evaluating a query over a database, where the size of the result can be exponential in the size of the input. *Polynomial running time* in the size of the input may be a wrong yardstick of efficiency when analyzing an enumeration algorithm, because just writing the output may require exponential time. Several definitions of efficiency for enumeration algorithms have been proposed [24]. The most commonly used is *polynomial input-output complexity*, which means that the running time is polynomial in the combined size of the input and the output. A stronger definition is *polynomial delay*, which means that the time before printing the first answer, as well as the interval of time (delay) between every two consecutive answers, is polynomial only in the input size. (Between the two there is the notion of *incremental polynomial time*.)

2.3.1 Ranked Enumeration

As said above, in many applications the result of an enumeration problem (e.g., query evaluation) consists of an enormous number of answers. However, the answers are not equally valuable to the user; instead, we often have a *scoring function* that discriminates between answers. Formally, consider an algorithm E for an enumeration problem P, and let $\mathsf{score}(\cdot)$ be an underlying, real-valued scoring function over the answers. We say that E enumerates *in decreasing* score if for all inputs x for P with a set $P(x) = \{y_1, \ldots, y_m\}$ of answers, if E prints y_i before y_j then $\mathsf{score}(y_i) \geq \mathsf{score}(y_j)$. Often, though, computational limitations enforce one to use *approximations*. Formally, let $\theta \geq 1$ be a number. We say that E enumerates in *θ-approximately decreasing* score [16, 32] if for all inputs x for P with $P(x) = \{y_1, \ldots, y_m\}$, if E prints y_i before y_j then $\theta \cdot \mathsf{score}(y_i) \geq \mathsf{score}(y_j)$. In this work, θ is usually not a constant number but rather a function of the input.

Suppose that an enumeration algorithm E enumerates with polynomial delay. If E enumerates in decreasing score, then one efficiently obtains *top-k* answers by stopping E after k outputs. Similarly, if E enumerates in θ-approximately

decreasing score, then stopping E after k outputs yields an approximation of the top-k answers, as defined by Fagin et al. [16]. Hence, the problem of incrementally enumerating in decreasing (resp., approximately decreasing) score generalizes that of top-k (resp., approximate top-k) evaluation.

3. FORMAL SETTING

We consider probabilistic data that come in the form of a *Markov sequence*, and we study the problem of evaluating queries over such data; our queries belong to various classes of *finite-state transducers*. In this section, we formally define our data and query models, and describe the computational problems that are derived from the task of query evaluation. We begin with the data model.

3.1 Markov Sequences

A *Markov sequence* is essentially a sequence of random variables over a finite set of states, with the property that future states depend only on the current state[3] and not on the past states (namely, the *Markov* property). Formally, a Markov sequence μ of length n operates over a finite set Σ of *state nodes* (or just *nodes*),[4] and comprises an *initial-state distribution* $\mu_{0\rightarrow}$ and a *transition function* $\mu_{i\rightarrow}$ for all $1 \leq i < n$, where $\mu_{0\rightarrow}$ and $\mu_{i\rightarrow}$ are defined as follows.

- $\mu_{0\rightarrow} : \Sigma \rightarrow [0,1]$ is a function, such that $\sum_{s\in\Sigma} \mu_{0\rightarrow}(s) = 1$ holds.
- $\mu_{i\rightarrow} : \Sigma \times \Sigma \rightarrow [0,1]$ is a function, such that for all $s \in \Sigma$ it holds that $\sum_{t\in\Sigma} \mu_{i\rightarrow}(s,t) = 1$.

The set Σ of the state nodes of the Markov sequence μ is denoted by Σ_μ. We may write $\mu[n]$ instead of μ to denote that μ is a Markov sequence of length n.

Example 3.1. Our running example for this paper is in the context of RFID. In particular, we consider a hospital, where transmitters are attached to medical equipment. Each transmitter transmits discrete signals with fixed time intervals in between. Sensors are spread over the hospital in known locations (e.g., rooms, hallways, laboratories, etc.). To the data management system, a transmission is an object comprising a *transmitter identifier* and a *sensor identifier*.

In our example, we consider a specific crash cart. Based on the relevant transmissions, a prediction of the location of the cart at each point in time is made. Such a prediction is done by viewing the transmissions as a sequence of observations in a *hidden Markov model* (HMM), and then translating this HMM into a Markov sequence (which, in our case, incorporates both the HMM and the observed sensor readings). A detailed discussion on HMMs and their translation into Markov sequences is given in the extended version of this paper [31].

Figure 1 shows a tiny example μ of the resulting Markov sequence. In this figure, we consider two rooms, numbered 1 and 2, and a lab. Each of the three contains two locations. For example, Room 1 has the locations $\mathsf{r_{1a}}$ and $\mathsf{r_{1b}}$,

[3]We mention that all our results generalize to *k-order* Markov sequences, provided that k is fixed.

[4]Note that Σ denotes the set of state nodes of a Markov sequence, and the alphabet of an NFA. This is not accidental, since we later use the state nodes as the alphabet symbols of an NFA.

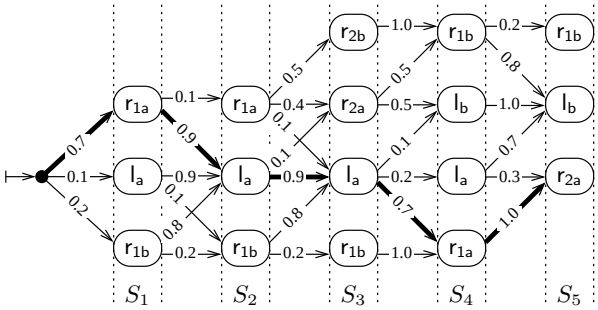

Figure 1: A Markov sequence μ describing locations of a hospital cart

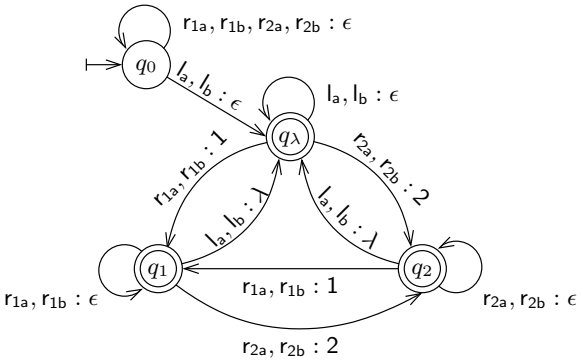

Figure 2: A transducer A^ω extracting a sequence of visited places

and the lab has the locations $\mathsf{l_a}$ and $\mathsf{l_b}$. The set Σ_μ comprises the six locations (i.e., $\mathsf{r_{1a}}$, $\mathsf{r_{1b}}$, etc.). The state nodes at different times are represented by rectangles with rounded corners. The functions $\mu_{0\rightarrow}$ and $\mu_{i\rightarrow}$ are represented by directed edges that are labeled with probabilities. As an example, $\mu_{0\rightarrow}(\mathsf{r_{1a}}) = 0.7$ is indicated by the upper edge emanating from the filled circle on the left. As another example, $\mu_{3\rightarrow}(\mathsf{l_a}, \mathsf{l_b}) = 0.1$ is indicated by the edge from the $\mathsf{l_a}$ rectangle to the $\mathsf{l_b}$ rectangle between variables S_3 and S_4 (which we discuss later). Note that edges with a zero probability are excluded from the figure. Finally, observe that the sum of edges emanating from each object is 1 (as we require in a Markov sequence). □

Semantically, a Markov sequence $\mu[n]$ defines the probability space (Σ_μ^n, p) over n-long strings of nodes, where the

Table 1: Random strings and their output

string	value	probability	output
s	$\mathsf{r_{1a}}\ \mathsf{l_a}\ \mathsf{l_a}\ \mathsf{r_{1a}}\ \mathsf{r_{2a}}$	0.3969	1 2
t	$\mathsf{r_{1a}}\ \mathsf{r_{1a}}\ \mathsf{l_a}\ \mathsf{r_{1a}}\ \mathsf{r_{2a}}$	0.0049	1 2
u	$\mathsf{l_a}\ \mathsf{r_{1b}}\ \mathsf{r_{1b}}\ \mathsf{r_{1a}}\ \mathsf{r_{2a}}$	0.002	1 2
v	$\mathsf{r_{1a}}\ \mathsf{l_a}\ \mathsf{r_{2a}}\ \mathsf{r_{1b}}\ \mathsf{l_b}$	0.0315	2 1 λ
w	$\mathsf{r_{1b}}\ \mathsf{r_{1b}}\ \mathsf{l_a}\ \mathsf{l_b}\ \mathsf{l_b}$	0.0.0252	ϵ
x	$\mathsf{r_{1a}}\ \mathsf{r_{1a}}\ \mathsf{r_{2b}}\ \mathsf{r_{1b}}\ \mathsf{r_{1b}}$	0.007	N/A

probability $p(\mathbf{s})$, for a string $\mathbf{s} = s_1 \cdots s_n$, is given by

$$p(\mathbf{s}) = \mu_{0\rightarrow}(s_1) \times \prod_{i=1}^{n-1} \mu_{i\rightarrow}(s_i, s_{i+1})\,. \tag{1}$$

By $\mathbf{S}^\mu = S_1^\mu \cdots S_n^\mu$ we denote the random variables that correspond to the random string $\mathbf{s}$ and its nodes. If μ is clear from the context, we may omit it from the superscript. Recall that the Markov property states that node S_i (at time i) depends only on node S_{i-1} (at time $i-1$). Formally, we have the following equations (which result directly from (1)) for all i such that $1 \leq i < n$, nodes s, and strings $\mathbf{t} = t_1, \ldots, t_i$.

- $\Pr(S_1 = s) = \mu_{0\rightarrow}(s)$
- $\Pr\left(S_{i+1} = s \mid S_{[1,i]} = \mathbf{t}\right) =$
$$\Pr(S_{i+1} = s \mid S_i = t_i) = \mu_{i\rightarrow}(t_i, s)$$

Example 3.2. Recall the Markov sequence μ that is depicted in Figure 1 and discussed in Example 3.1. Table 1 shows five random strings of μ. For now, the rightmost column ("output") should be ignored. Take, for example, the string $\mathbf{s} = \mathsf{r}_{1a}\,\mathsf{l}_a\,\mathsf{l}_a\,\mathsf{r}_{1a}\,\mathsf{r}_{2a}$ of the top row. This string corresponds to a unique directed path of Figure 1, from the filled circle (on the left) to a node in the rightmost column. The probability of $\mathbf{s}$ (that is, $p(\mathbf{s})$ or $\Pr(\mathbf{S} = \mathbf{s})$) is obtained by multiplying the probabilities along this unique path, that is, $p(\mathbf{s}) = 0.7 \times 0.9 \times 0.9 \times 0.7 \times 1.0 = 0.3969$. □

Consider a Markov sequence $\mu[n]$. A string $\mathbf{s} \in \Sigma_\mu^n$ is viewed as a *data instance*, upon which one can evaluate meaningful queries. But, as mentioned above, μ defines a probability space over Σ_μ^n (captured by the random variable $\mathbf{S}$). Thus, we view μ as a *probabilistic database*, and apply the query according to the concept of querying probabilistic data (which we formally define later on). Next, we formalize the notion of a query over a string; later, we give the formalism of applying such a query to a Markov sequence.

3.1.1 Finite-State Transducers

The queries that we consider are *finite-state string transducers*, or just *transducers* for short. We restrict the discussion to transducers with deterministic emission; that is, each state transition deterministically produces a string of output symbols, and there are no *empty transitions* (i.e., transitions that do not read a symbol of the input string). We further discuss this restriction in Section 7. Formally, a *transducer* comprises an NFA A and an *output function* $\omega : Q_A \times \Sigma_A \times Q_A \rightarrow \Delta^*$, where Δ is the *output alphabet*. The transducer comprising A and ω is denoted by A^ω. We assume that Δ is the set of all the symbols that occur in the image of ω, and we denote it by Δ_ω.

The transducer A^ω *transduces* a string $\mathbf{s} \in \Sigma_A^n$ into a string $\mathbf{o} \in \Delta_\omega^*$, denoted $\mathbf{s} \rightarrow[A^\omega]\rightarrow \mathbf{o}$, if there exists an accepting run $\rho : \{1, \ldots, n\} \rightarrow Q_A$ on $\mathbf{s}$ such that

$$\mathbf{o} = \omega(q_A^0, s_1, q_1)\,\omega(q_1, s_2, q_2) \cdots \omega(q_{n-1}, s_n, q_n)\,,$$

where $q_i = \rho(i)$ for all $1 \leq i \leq n$. Recall that each $\omega(q, s, q')$ is a string over Δ_ω (which can be empty).

Let A^ω be a transducer. We say that A^ω is *deterministic* if A is deterministic (i.e., A is a DFA). We say that A^ω is *non-selective* if $F_A = Q_A$ (thus, A accepts every string); otherwise, A^ω is selective. The output function ω is said to be *k-uniform*, where k is a nonnegative integer, if the length of the string $\omega(q, s, q')$ is k for all $(q, s, q') \in Q_A \times \Sigma_A \times Q_A$. As an example, if ω is 1-uniform, then A^ω replaces each input symbol with an output symbol (given that the input string is accepted by A). As another example, if ω is 0-uniform, then A^ω simply tests whether the input string is accepted by A, and if so it outputs ϵ. We say that ω is *uniform* if it is k-uniform for some k. To show lower bounds on complexity, we have a special interest in the restricted class of Mealy machines, where a *Mealy machine* is a deterministic and non-selective transducer A^ω, such that ω is 1-uniform.

Example 3.3. Figure 2 shows a transducer A^ω over the alphabet $\Sigma_A = \{\mathsf{r}_{1a}, \mathsf{r}_{1b}, \mathsf{r}_{2a}, \mathsf{r}_{2b}, \mathsf{l}_a, \mathsf{l}_b\}$ (which, not coincidentally, is the language Σ_μ of the Markov sequence μ of Figure 1). Each state is represented by a circle, an accepting state has an inner circle, and the initial state has an incoming arrow. Thus, the set Q_A is $\{q_0, q_\lambda, q_1, q_2\}$, the set F_A is $\{q_\lambda, q_1, q_2\}$, and the initial state q_A^0 is q_0. The functions δ_A and ω are represented by the labels on the directed edges of the figure. Generally, the notation $\sigma : \mathbf{o}$ on the edge from q and q' means that $q' \in \delta_A(q, \sigma)$ and $\omega(q, \sigma, q') = \mathbf{o}$. We use the notation $\sigma_1, \ldots, \sigma_k : \mathbf{o}$ as a shorthand for $q' \in \delta_A(q, \sigma_i)$ and $\omega(q, \sigma_i, q') = \mathbf{o}$ for all $i = 1, \ldots, k$. For example, reading l_a or l_b moves A from q_1 to q_λ (i.e., $\delta_A(q_1, \mathsf{l}_a) = \delta_A(q_1, \mathsf{l}_b) = \{q_\lambda\}$), and λ is emitted in this transition (i.e., $\omega(q_1, \mathsf{l}_a, q_\lambda) = \omega(q_1, \mathsf{l}_b, q_\lambda) = \lambda$).

It is easy to verify that the transducer A^ω is deterministic. It is selective, since A does not accept every string (in particular, A accepts the strings that have at least one occurrence of either l_a or l_b). It is not uniform, since different emissions may have different lengths (among the lengths 1 and 0). □

3.1.2 Querying a Markov Sequence

Recall that data represented as a Markov sequence $\mu[n]$ defines a probability space over Σ_μ^n. To evaluate a query (transducer) over μ, we adopt the traditional concept of querying probabilistic data where the goal is to produce each possible answer along with its *confidence* (probability) [8, 30, 48].

Formally, let $\mu[n]$ be a Markov sequence, and let A^ω be a transducer. Throughout the paper, we make the implicit assumption that $\Sigma_A = \Sigma_\mu$; that is, the strings read by A^ω are in the same language as that of the random strings generated by μ. We denote by $A^\omega(\mu)$ the set of all the strings that have a nonzero probability of being transduced; that is,

$$A^\omega(\mu) \stackrel{\text{def}}{=} \{\mathbf{o} \in \Delta_\omega^* \mid \Pr(\mathbf{S} \rightarrow[A^\omega]\rightarrow \mathbf{o}) > 0\}\,.$$

A string of $A^\omega(\mu)$ is called an *answer*. The *confidence* of an answer is its probability of being transduced by a random instance of μ. The result of evaluating A^ω over μ is the mapping that assigns to each answer $\mathbf{o} \in A^\omega(\mu)$ its confidence. More formally, query evaluation is the task of producing the mapping $conf : A^\omega(\mu) \rightarrow [0, 1]$ where $conf(\mathbf{o})$ (the confidence of $\mathbf{o}$) is equal to $\Pr(\mathbf{S} \rightarrow[A^\omega]\rightarrow \mathbf{o})$ for all $\mathbf{o} \in A^\omega(\mu)$.

Example 3.4. We continue our running example. Recall the Markov sequence of Figure 1 (described in Example 3.1). Consider a scenario, where we identify that the particular cart is contaminated. Suppose also that we know the cart was not contaminated in its first visit to the lab. Our goal is to trace the sequence of places (i.e., Room 1, Room 2

and lab) the cart has been in after that first visit. So, we construct the transducer A^ω of Figure 2. This transducer emits, after the first visit to the lab, a symbol representing a place whenever this place is being entered to from another place. As an example, for both the strings $\mathbf{u} = \mathsf{l_a}\ \mathsf{r_{1b}}\ \mathsf{r_{1b}}\ \mathsf{r_{1a}}\ \mathsf{r_{2a}}$ and $\mathbf{s} = \mathsf{r_{1a}}\ \mathsf{l_a}\ \mathsf{l_a}\ \mathsf{r_{1a}}\ \mathsf{r_{2a}}$ of Table 1, A^ω emits the output $\mathsf{1\,2}$.

The rightmost column of Table 1 shows the output that A^ω emits for the strings of the corresponding rows. An exception is the last row, where the corresponding string $\mathbf{x}$ is not acceptable by A and then the output is marked as "N/A." Recall from Example 3.2 that the strings of the table are random strings of μ. Note that the table does not contain all the random strings of μ; however, from the rightmost column it follows that $A^\omega(\mu)$ contains (at least) the strings $\mathsf{1\,2}$, $\mathsf{2\,1}\,\lambda$, and ϵ. Also, it can be verified that the table contains all the random strings of μ that are transduced into $\mathsf{1\,2}$; these strings are $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{u}$, which have the probabilities 0.3969, 0.0049 and 0.002, respectively. Hence, $conf(\mathsf{1\,2})$ is $0.3969 + 0.0049 + 0.002 = 0.4038$. □

3.2 Computational Problems

Naïvely speaking, query evaluation entails two tasks: computing the set of answers, and computing the confidence of each answer. The input for each of the two tasks consists of a Markov sequence μ and a transducer A^ω, and in the second task we also get a string $\mathbf{o} \in \Delta_\omega^*$. We could also require $\mathbf{o}$ to be an answer of $A^\omega(\mu)$, since it can be shown that whether a string $\mathbf{o} \in \Delta_\omega^*$ is an answer (i.e., has a nonzero probability) can be decided efficiently. We assume that μ and A^ω are represented in a straightforward manner; in particular, the representation of $\mu[n]$ consists of a transition matrix for each index $1 \leq i \leq n$, and an array for $\mu_{0\rightarrow}$.

We use the convention that each probability in a Markov sequence is a rational number represented by a pair of integers (each of which is represented in the standard binary encoding): the numerator and the denominator.

Evaluating a query naïvely by separately solving the above two tasks is often impractical, since for given $\mu[n]$ and A^ω, the set $A^\omega(\mu)$ can be huge (in the worst case its size can be exponential in n, even if A^ω is fixed). Moreover, many (even the majority) of the answers $\mathbf{o} \in A^\omega(\mu)$ may be such that the confidence of $\mathbf{o}$ is tiny—too small to be of practical interest. The desired kind of evaluation is that of a *ranked enumeration* of $A^\omega(\mu)$, namely, an incremental algorithm (e.g., one that runs with polynomial delay) that enumerates $A^\omega(\mu)$ in decreasing confidence; if decreasing confidence is infeasible, we may settle for an approximation thereof. An efficient procedure for computing the confidence of an answer is still required if the user desires the confidence to be given along with each answer.

Finally, to explore the complexity of ranked enumeration, we study the restricted problem of finding a *top answer*. Formally, the input consists of a Markov sequence μ and a transducer A^ω, and the goal is to find an answer with a maximal confidence, that is, a string $\mathbf{o}$, such that $\Pr(\mathbf{S} \rightarrow\!\![A^\omega]\!\!\rightarrow \mathbf{o}) \geq \Pr(\mathbf{S} \rightarrow\!\![A^\omega]\!\!\rightarrow \mathbf{o}')$ for all $\mathbf{o}' \in \Delta_\omega^*$. We also consider the problem of finding a *θ-approximate top answer* for a number (or function) $\theta \geq 1$; that is, a string $\mathbf{o} \in \Delta_\omega^*$, such that $\theta \cdot \Pr(\mathbf{S} \rightarrow\!\![A^\omega]\!\!\rightarrow \mathbf{o}) \geq \Pr(\mathbf{S} \rightarrow\!\![A^\omega]\!\!\rightarrow \mathbf{o}')$ for all $\mathbf{o}' \in \Delta_\omega^*$. An important observation, which we repeatedly use, is that hardness of finding a top answer (resp., a θ-approximate top answer) rules out the existence of an evaluation in decreasing (resp., θ-approximately decreasing) confidence with polynomial delay, as such an answer can be obtained by taking the first output in the enumeration.

4. THE COMPLEXITY OF QUERY EVALUATION

We now study the complexity of evaluating transducers over Markov sequences. We start with the problem of unranked enumeration of the answers.

4.1 Unranked Enumeration

Consider a transducer A^ω and a Markov sequence μ. In this section, we consider the problem of enumerating $A^\omega(\mu)$ in an unranked fashion, namely, the order of answers disregards confidence values. The following theorem shows that this problem is tractable. More specifically, given a transducer and a Markov sequence, one can enumerate all the answers with polynomial delay and polynomial space.

THEOREM 4.1. *Given a Markov sequence μ and a transducer A^ω, the set $A^\omega(\mu)$ can be enumerated with polynomial delay and polynomial space.*

To prove Theorem 4.1, we present a polynomial-space algorithm that runs with a delay that is, roughly, polynomial in A^ω, linear in the length of μ, and quadratic in the length of the two answers before and after the delay. The algorithm uses a general technique [34] that reduces an enumeration problem into the problem of testing satisfiability of *constraints*, which are used for recursively partitioning the space of solutions. The challenge in applying this technique is to find a proper class of (tractable) constraints. Here, we use what we call *prefix constraints*, and we show that such a constraint can be enforced by efficiently transforming the input transducer into a new one; thus, we are left with the problem of testing whether $A^\omega(\mu) \neq \emptyset$, or equivalently $\Pr(\mathbf{S} \in \mathcal{L}(A)) > 0$, given A^ω and μ. The latter problem is shown to be tractable by deploying dynamic programming.

Theorem 4.1 shows tractability of unranked enumeration. In the next section, we consider the problem of enumerating the answers in ranked order.

4.2 Ranked Enumeration

Ideally, we would like the answers to be enumerated efficiently (i.e., with polynomial delay) in decreasing confidence. Unfortunately, as we later show, this problem is highly intractable. So, we first propose an enumeration in a *heuristic order* by deploying a different scoring function: the maximal probability of an *evidence*. Next, we give a formal explanation.

Let μ be a Markov sequence, and let A^ω be a transducer. Let $\mathbf{o} \in A^\omega(\mu)$ be given. We denote by $E_{\max}(\mathbf{o})$ the maximal number p, such that there exists a string $\mathbf{s} \in \Sigma_\mu^n$ (an *evidence*) where $\mathbf{s} \rightarrow\!\![A^\omega]\!\!\rightarrow \mathbf{o}$ and $p = \Pr(\mathbf{S} = \mathbf{s})$.

Example 4.2. Consider again Example 3.1, and let $\mathbf{o}$ be the string $\mathsf{1\,2}$. Recall that the set of random strings of μ that are transduced into $\mathbf{o}$ are exactly $\mathbf{s}$, $\mathbf{t}$ and $\mathbf{u}$ of Table 1. Since $\mathbf{s}$ has the highest probability among the three, 0.3969, we conclude that $E_{\max}(\mathbf{o}) = 0.3969$. □

The following theorem shows that the answers for a transducer over a Markov sequence can be enumerated in decreasing $E_{\max}$ with polynomial delay. Note that unlike unranked

enumeration (Theorem 4.1), here polynomial space is not guaranteed; indeed, the used space can grow proportionally to the number of printed answers.

THEOREM 4.3. *$A^\omega(\mu)$ can be enumerated in decreasing $E_{\max}$ with polynomial delay, given a transducer A^ω and a Markov sequence μ.*

In the proof of Theorem 4.3, we devise an algorithm that uses the ranked-enumeration technique of Lawler-Murty [38, 43]. This technique is essentially a reduction from enumeration of results in ranked order to optimization (i.e., finding the best result) under constraints. As in the proof of Theorem 4.1, the class of constraints we use here is that of our prefix constraints. As stated above, a prefix constraint can be enforced over the given transducer. Hence, we obtain a reduction to the problem of finding a top answer w.r.t. $E_{\max}$, and we give an efficient algorithm for that.

From a formal point of view, when considering the enumeration of Theorem 4.3 as an approximation of decreasing confidence, the approximation ratio is poor: if the length of μ is n, then the (worst-case) approximation ratio is $|\Sigma_A|^n$. Surprisingly, the following theorem shows that one cannot do significantly better than Theorem 4.3, namely, a top answer (and, hence, the whole enumeration) cannot be efficiently approximated within any sub-exponential factor. Moreover, this inapproximability holds already for Mealy machines, such that the underlying DFA comprises exactly one state!

THEOREM 4.4. *Assume $\mathrm{P} \neq \mathrm{NP}$. For all $\delta > 0$, no polynomial-time algorithm finds a $2^{n^{1-\delta}}$-approximate top answer, given a Mealy machine A^ω and a Markov sequence $\mu[n]$. Furthermore, it holds even if $|Q_A| = 1$ is assumed.*

Theorem 4.4 shows hardness for Mealy machines with a DFA that comprises one state; however, the alphabet Σ_A is unbounded. Next, we show the existence of a *fixed* deterministic transducer (with a fixed alphabet), such that the same inapproximability holds. This transducer has the special form of a *projector*, which is a transducer A^ω such that each $\omega(q, s, q')$ is either the input symbol s or ϵ.

THEOREM 4.5. *Assume $\mathrm{P} \neq \mathrm{NP}$. There exists a (fixed) deterministic projector A^ω, such that for all $\delta > 0$, no polynomial time algorithm finds a $2^{n^{1-\delta}}$-approximate top answer, given a Markov sequence $\mu[n]$.*

The transducer A^ω of Theorem 4.5 is tiny: $|\Sigma_A| = 4$, $|\Delta_\omega| = 2$ and $|Q_A| = 1$. The proofs of Theorems 4.4 and 4.5 use two different reductions from the problem max-3-DNF, which is that of finding a truth assignment that maximizes the number of satisfied clauses in a given 3-DNF formula (this problem does not have an efficient 7/8-approximation, unless $\mathrm{P} = \mathrm{NP}$ [15]). More specifically, each reduction gives a constant-ratio lower bound on approximability. Then, we amplify a $2^{n^{1-\delta}}$-approximation to any desirable constant-factor approximation, by essentially concatenating a polynomial number of copies of the given Markov sequence.

4.3 Confidence Computation

We now consider the problem of computing the confidence of an answer. Our results for deterministic transducers differ from those for nondeterministic ones, and we begin with the deterministic case.

The following theorem shows a polynomial-time complexity for the problem of computing the confidence of an answer for deterministic transducers. It also shows that we have a faster algorithm in the case of uniform emission.

THEOREM 4.6. *Computing $\Pr(\mathbf{S} \twoheadrightarrow [\![A^\omega]\!] \twoheadrightarrow \mathbf{o})$, given a deterministic transducer A^ω, a Markov sequence $\mu[n]$ and a string $\mathbf{o} \in \Delta_\omega^*$, is in $O(|\mathbf{o}|n|\Sigma_\mu|^2|Q_A|^2)$ time. Furthermore, it is in $O(kn|\Sigma_\mu|^2|Q_A|^2)$ time if ω is k-uniform.*

Theorem 4.6 is proved by providing dynamic-programming algorithms (assuming or not assuming k-uniformity) for computing the confidence of an answer. Theorem 4.6, 4.1 and 4.3 together show that for deterministic transducers, there is an efficient query evaluation, either unranked or ranked by $E_{\max}$, where the confidence of each answer is given along with the answer itself (e.g., as soon as the answer is printed).

We now consider general (nondeterministic) transducers. Unfortunately, Theorem 4.6 does not generalize to the class of all transducers (under standard complexity assumptions), as it is intractable to compute the confidence of an answer. Furthermore, this holds even if we assume uniformity and non-selectivity. This is shown in the following proposition, which follows rather straightforwardly from the fact that computing $|\mathcal{L}(A) \cap \Sigma_A^n|$, given an NFA A and a natural number n in unary representation, is #P-complete [28].

PROPOSITION 4.7. *Computing $\Pr(\mathbf{S} \twoheadrightarrow [\![A^\omega]\!] \twoheadrightarrow \mathbf{o})$, given as input a Markov sequence μ, a transducer A^ω, and a string $\mathbf{o} \in \Delta_\omega^*$, is $\mathrm{FP}^{\#\mathrm{P}}$-complete.[5] It remains $\mathrm{FP}^{\#\mathrm{P}}$-hard even if A^ω is non-selective and ω is 1-uniform.*

Whether there exists an efficient approximation for the computation of a confidence (for nondeterministic transducer) is unknown. The left gap is large, since we do not even have a sub-exponential efficient approximation. We leave the resolution of this gap to future work. We mention that an FPRAS (i.e., *fully polynomial-time randomized approximation scheme*) for this problem would straightforwardly imply an FPRAS for computing $|\mathcal{L}(A) \cap \Sigma_A^n|$ (as described above), which is a long-standing open problem.[6] Nevertheless, the following theorem shows that in the case of uniform emission, the confidence of an answer can be computed in time that is polynomial in $2^{|Q_A|}$ and in the size of rest of the input (which is often practical, and in line with the notions of *data complexity* [57] and *fixed-parameter tractability* [12, 44]).

THEOREM 4.8. *Computing $\Pr(\mathbf{S} \twoheadrightarrow [\![A^\omega]\!] \twoheadrightarrow \mathbf{o})$, given a transducer A^ω where ω is k-uniform, a Markov sequence μ, and a string $\mathbf{o} \in \Delta_\omega^*$, is in $O(nk \cdot |\Sigma_\mu|^2 \cdot 4^{|Q_A|})$ time.*

The proof of Theorem 4.8 combines dynamic programming with some form of *subset construction*. Observe that this result cannot be obtained by just straightforwardly determinizing A; to see that, observe that a nondeterministic

[5] $\mathrm{FP}^{\#\mathrm{P}}$ is the class of functions that are polynomial-time computable using an oracle to some function in #P, where #P [56] is the class of functions that count the number of accepting paths for the input of an NP machine. Using an oracle to a #P-hard (or $\mathrm{FP}^{\#\mathrm{P}}$-hard) function, one can efficiently solve every problem in the polynomial hierarchy [54].

[6] To the best of our knowledge, the best approximation is that of Kannan et al. [28], which gives a quasi-polynomial *FRAS* for the computation of $|\mathcal{L}(A) \cap \Sigma_A^n|$.

transducer can transduce a string $\mathbf{s}$ into multiple strings $\mathbf{o}$ whereas a deterministic transducer transduces $\mathbf{s}$ into at most one $\mathbf{o}$ (thus, a transducer cannot be determinized in a straightforward sense). Another argument is that Theorem 4.8 does not hold without the assumption of uniformity, which we show next. (Recall from Theorem 4.6 that deterministic transducers do not require uniformity for allowing efficient confidence computation.)

THEOREM 4.9. *There exists a (fixed) non-selective transducer A^ω with $|Q_A| = |\Delta_\omega| = 3$ and $|\Sigma_A| = 5$, such that computing $\Pr(\mathbf{S} \twoheadrightarrow[A^\omega]\rightarrowtail \mathbf{o})$, given a Markov sequence μ and a string $\mathbf{o} \in \Delta_\omega^*$, is $\mathrm{FP}^{\#\mathrm{P}}$-complete.*

The proof of Theorem 4.9 is by a reduction from counting the number of satisfying assignments for a monotonic bipartite 2-DNF formula, which is known to be #P-complete [45].

5. SUBSTRING PROJECTORS

In this section, we study a restricted class of transducers, which we view as common in practice, and so, practically important. We will show that this class is more tractable than general transducers in terms of ranked enumeration of answers. We call a machine of our restricted class a *substring projector*, or *s-projector* for short.

Intuitively, when an s-projector runs over an input string, it looks for a substring that matches a pattern (automaton), and this substring is emitted. In addition, an s-projector can specify constraints, in the form of automata, over the prefix and suffix of the input string that occur before and after the matched substring, respectively. Formally, an *s-projector* P is represented by DFAs B and E, called a *prefix constraint* and a *suffix constraint*, respectively, and a deterministic 1-uniform projector A^ω (that is, A^ω is a deterministic transducer, such that $\omega_A(q_1, s, q_2) = s$ for all $q_1, q_2 \in Q_A$ and $s \in \Sigma_A$). The s-projector given by B, E and A^ω is denoted by $[B]A[E]$ (note that ω is not needed). For the s-projector $P = [B]A[E]$, we require that $\Sigma_B = \Sigma_A = \Sigma_E$, and we denote this alphabet by Σ_P. For strings $\mathbf{s}, \mathbf{o} \in \Sigma_P^*$, we denote by $\mathbf{s} \twoheadrightarrow[[B]A[E]]\rightarrowtail \mathbf{o}$ the fact that $\mathbf{o} \in \mathcal{L}(A)$, and there exist two strings $\mathbf{b}$ and $\mathbf{e}$, such that $\mathbf{b} \in \mathcal{L}(B)$, $\mathbf{e} \in \mathcal{L}(E)$, and $\mathbf{s}$ is the concatenation $\mathbf{boe}$. An *s-projector* $P = [B]A[E]$ is *simple* if B and E accept *every* string of Σ_P^*; in this case, P is denoted by $[*]A[*]$.

Example 5.1. Continuing with the hospital RFID example, we may want to know the path that a patient takes from the operating room to the recovery unit. For that, we use an *s*-projector that outputs the path from the point when the patient is in the operating room, specified as a prefix constraint, and when the patient arrives at the recovery room. In other domains, such as handwritten-form data, s-projectors are particularly useful, as they allow us to perform tasks of *data extraction* (e.g., for finding names and email addresses). As a concrete (simplistic) example, consider the s-projector $P = [B]A[E]$, where B, A and E correspond to the Perl-syntax expressions "`.*Name:`", "`[a-zA-Z,]+`", and "`\s.*`", respectively. When applied to a textual string, P produces `Hillary` if `Name:Hillary`, followed by the whitespace character, occurs (anywhere) in the text. □

As in the case of transducers, when given a Markov sequence μ and an s-projector P, we assume that $\Sigma_\mu = \Sigma_P$. An easy observation is that, given an s-projector P (represented by B, A and E), one can efficiently construct a (nondeterministic) transducer $\hat{A}^{\hat{\omega}}$, such that for all $\mathbf{s} \in \Sigma_P^*$ and $\mathbf{o} \in \Delta_\omega^*$ it holds that $\mathbf{s} \twoheadrightarrow[P]\rightarrowtail \mathbf{o}$ if and only if $\mathbf{s} \twoheadrightarrow[\hat{A}^{\hat{\omega}}]\rightarrowtail \mathbf{o}$. Thus, we can view P as a special case of a transducer. In particular, Theorem 4.1 holds for s-projectors; that is, the set $P(\mu)$ (i.e., the set of strings $\mathbf{o} \in \Delta_\omega^*$ that are transduced into with a nonzero probability) can be enumerated with polynomial delay and polynomial space.

Recall from Theorem 4.4 that for (deterministic) transducers, it is intractable to achieve an enumeration in approximately decreasing confidence with a ratio that is subexponential in the length of the Markov sequence μ. In contrast, the following theorem shows that for s-projectors, we can get an efficient enumeration where the approximation ratio is the length of μ.

THEOREM 5.2. *The set $P(\mu)$ can be enumerated in n-approximately decreasing confidence with polynomial delay, given an s-projector P and a Markov sequence $\mu[n]$.*

Theorem 5.2 is a consequence of results we give later on, and we discuss the proof in Section 5.2.

It is yet unknown whether the upper bound of Theorem 5.2 is tight. However, the following theorem gives a square-root (rather than linear) lower bound, under the assumption that NP ≠ ZPP.[7] The lower bound holds already in a case where a simple s-projector is held fixed. In particular, this theorem rules out the existence of an efficient ranked enumeration, and a constant (or logarithmic) approximation thereof.

THEOREM 5.3. *Assume* NP ≠ ZPP. *There exists a (fixed) simple s-projector $[*]A[*]$, such that for all $\delta > 0$, there is no polynomial-time algorithm for $(n^{\frac{1}{2}-\delta})$-approximating a top answer, given a Markov sequence $\mu[n]$.*

The proof of Theorem 5.3 is by a reduction from finding a *maximum independent set* of a graph, which is known to be inapproximable within a $|V|^{1-\delta}$ factor, for all $\delta > 0$, where V is the set of nodes [19].

We now consider the problem of computing the confidence of an answer. Theorem 4.6 shows that this problem is tractable for deterministic transducers. One might expect the same for s-projectors, since an s-projector is defined by means of DFAs. Unfortunately, this is not the case, and the following theorem shows that the problem is #P-complete for s-projectors. Furthermore, this problem remains hard even if we fix the alphabet Σ, and we fix the DFAs B and A to be those accepting every string and only the empty string, respectively.

THEOREM 5.4. *Computing $\Pr(\mathbf{S} \twoheadrightarrow[P]\rightarrowtail \mathbf{o})$, given a Markov sequence $\mu[n]$, an s-projector $P = [B]A[E]$ and a string $\mathbf{o} \in \Delta_\omega^*$, is $\mathrm{FP}^{\#\mathrm{P}}$-complete. It remains $\mathrm{FP}^{\#\mathrm{P}}$-hard, even if the alphabet Σ_P is fixed, B accepts every string, and A accepts only ϵ.*

The next theorem shows that for an s-projector $[B]A[E]$, the confidence of an answer can be computed in time that

[7] ZPP is the class of decision problems that can be solved by a randomized algorithm with polynomial expected time. NP = ZPP is believed to be unlikely [60].

is polynomial in $2^{|Q_E|}$ and in the size of the rest of the input. Combined with Theorem 5.4, we learn that hardness of confidence computation stems solely from the size of the suffix constraint E.

THEOREM 5.5. *Computing* $\Pr(\mathbf{S} \twoheadrightarrow [P] \twoheadrightarrow \mathbf{o})$*, given a Markov sequence* $\mu[n]$*, an s-projector* $P = [B]A[E]$ *and a string* $\mathbf{o} \in \Delta_\omega^*$*, is in* $O(n|\mathbf{o}|^2|\Sigma_P|^2|Q_B|^2 4^{|Q_E|})$ *time.*

The proof of Theorem 5.5 uses the fact that the *state complexity* of the concatenation $M.N$ of two DFAs M and N is exponential $|Q_N|$ while polynomial in $|Q_M|$ [23].

5.1 Indexed s-Projectors

An s-projector extracts substrings that match a DFA (subject to prefix and suffix constraints). Often, it is required to provide the user with not only the matched substring, but also an actual occurrence of that substring in the input. For example, when extracting names from text, we may want to identify each *occurrence* of a name (rather than each name) as an answer. To support this need, in this section we require the s-projector to specify the *index* within $\mathbf{S}$ of where emission begins. Formally, for an s-projector $P = [B]A[E]$ and a string $\mathbf{s}$, an answer is now a pair $(\mathbf{o}, i)$, such that $\mathbf{s}$ is the concatenation $\mathbf{bue}$, where $\mathbf{b} \in \mathcal{L}(B)$, $\mathbf{e} \in \mathcal{L}(E)$, $\mathbf{u} \twoheadrightarrow [A] \twoheadrightarrow \mathbf{o}$, and $|\mathbf{b}| = i - 1$. We call such an s-projector an *indexed s-projector*, and denote it by $[B]{\downarrow}A[E]$ (that is, we add the down-arrow to the notation of an s-projector). We next show that this twist results in a significant increase of tractability.

REMARK 5.6. *Technically, emitting the index is not directly captured by a transducer. Nevertheless, we can simulate it by requiring the s-projector* $P = [B]A[E]$ *to emit a special symbol* $\perp$ *whenever* ϵ *should to be emitted before* A *is reached. Thus, the index is obtained from the number of* $\perp$*s in the answer.* □

In the following theorem, we show that for an indexed s-projector P, query evaluation can be done in the *exact* ranked order with polynomial delay and polynomial space; in contrast, recall that the enumeration of Theorem 4.3 guarantees only an exponential-factor approximation, and does not guarantee polynomial space (and, in fact, its space usage may grow linearly with the output).

THEOREM 5.7. $P(\mu)$ *can be enumerated in decreasing confidence with polynomial delay and polynomial space, given a Markov sequence* μ *and an indexed s-projector* P*.*

To prove Theorem 5.7, we use a reduction to the enumeration of the directed paths between two nodes of an edge-weighted DAG [14].

Finally, the following theorem shows that the confidence of an answer can be computed in polynomial time for indexed s-projectors $P = [B]{\downarrow}A[E]$. Note that, unlike the running time of Theorem 5.5, the dependence of the running time on P is polynomial.

THEOREM 5.8. *Given an indexed s-projector* P*, a Markov sequence* $\mu[n]$ *and an answer* $(\mathbf{o}, i)$*, the confidence of* $(\mathbf{o}, i)$ *can be computed in* $O(n|\Sigma_P|^2|Q_P|^2)$ *time.*

5.2 Proof of Theorem 5.2

We now discuss the proof of Theorem 5.2. Let $P = [B]A[E]$ be an s-projector. Let μ be a Markov sequence, and let $\mathbf{o} \in P(\mu)$ be answer for P. Then, there exists one or more indices i, such that $(\mathbf{o}, i)$ is an answer for $[B]{\downarrow}A[E]$. We denote by $I_{\max}(\mathbf{o})$ the maximal number p, such that there exists some i where $(\mathbf{o}, i) \in [B]{\downarrow}A[E](\mu)$ and $p = \Pr(\mathbf{S} \twoheadrightarrow [[B]{\downarrow}A[E]] \twoheadrightarrow (\mathbf{o}, i))$. The proof of the following proposition is straightforward.

PROPOSITION 5.9. *For all answers* $\mathbf{o} \in P(\mu)$ *we have:*

$$I_{\max}(\mathbf{o}) \leq \Pr(\mathbf{S} \twoheadrightarrow [P] \twoheadrightarrow \mathbf{o}) \leq n \cdot I_{\max}(\mathbf{o}).$$

We conclude that an enumeration of $P(\mu)$ in decreasing $I_{\max}$ is an enumeration in n-approximately decreasing confidence. This observation, combined with Theorem 5.7, implies the following algorithm: execute the enumeration of Theorem 5.7, and whenever an answer $(\mathbf{o}, i)$ should be printed, print $\mathbf{o}$ instead. But, we must avoid printing duplicates, and thus need to remember the set of printed strings and consult this set before each print. Since a large chunk of duplicates may be encountered, polynomial delay is not guaranteed (although *incremental polynomial time* [24] is). Nevertheless, polynomial delay can be obtained by combining the strategy used for Theorem 4.3, and the corollary of Theorem 5.7 that finding a top answer under $I_{\max}$ is tractable. Thus, we obtain the following lemma.

LEMMA 5.10. $P(\mu)$ *can be enumerated with polynomial delay in decreasing* $I_{\max}$*, given an s-projector* P *and a Markov sequence* $\mu[n]$*.*

Finally, Theorem 5.2 follows immediately from Lemma 5.10 and Proposition 5.9.

6. RELATED WORK

Markovian-stream databases have seen interest in the last couple of years with several distinct research projects. Kanagal and Deshpande [25,26] advocate an approach to answering queries on sequential databases where the data and the query are transformed into a graphical model. They transform queries that are expressed in a sequence algebra [50] to probabilistic inference on graphical models. Their approach allows them to handle more general correlation structures than HMMs [27]. Their querying setting and complexity results appear to be inherently different from ours (e.g., they do not consider hardness of approximation), and it would be interesting to perform a detailed comparison with their approach. The focus of the CLARO project [11, 55] is on high-volume data streams, where even simple models like HMMs may not be able to keep up with overwhelming data. There, query processing is a streaming variant of SQL. The LAHAR system [39,40,47] was originally motivated by RFID data, but has been applied to other sensor data and speech data as well. There, queries are essentially linear DFAs combined with aggregation across many streams. Such a query is Boolean, and at each time period it returns the probability that it is evaluated to true. In all of these systems, the queries have small result sizes, and so they do not have to deal with the central technical problem posed in this paper: coping with queries (namely transducers) that return huge numbers of answers.

Table 2: Complexity of transducing Markov sequences

Measure	*general*	*uniform emission*	*deterministic*	*s-projectors*	*indexed s-projectors*
Complexity of confidence computation	$\frac{\text{FP}^{\#\text{P}}\text{-complete}}{\text{FP}^{\#\text{P}}\text{-complete}}$	$\frac{\text{FP}^{\#\text{P}}\text{-complete}}{\text{PTIME}}$	$\frac{\text{PTIME}}{\text{PTIME}}$	$\frac{\text{FP}^{\#\text{P}}\text{-complete}}{\text{PTIME}}$	$\frac{\text{PTIME}}{\text{PTIME}}$
Order for ranked evaluation with polynomial delay	$E_{\max}$: $\lvert\Sigma_A\rvert^n$	$E_{\max}$: $\lvert\Sigma_A\rvert^n$	$E_{\max}$: $\lvert\Sigma_A\rvert^n$	$I_{\max}$: n	*conf* (PSPACE)
	no order (PSPACE)				
Inapprox. $\forall\delta > 0$	$2^{n^{1-\delta}}$	$2^{n^{1-\delta}}$	$2^{n^{1-\delta}}$	$n^{\frac{1}{2}-\delta}$	N/A

In recent years, there has been a flurry of activity in discrete probabilistic relational databases, notably the Mystiq project [3], the Trio project [58], and MayBMS [22]. These approaches all consider the query language to be SQL. A notable exception is the more expressive language of Koch [35, 36], which allows the query to introduce further uncertainty. The problem of enumerating maximal answers to relational joins over probabilistic relations has also been studied [33]; that problem is inherently different from those studied here, and its analysis required different algorithmic and proof techniques (with *hereditary properties* [6] playing a central role). None of the above systems deal with sequential probabilistic models. There has also been work on managing continuous uncertainty, notably the Orion project [5, 53] and the BBQ project [10]. Extending our work to continuous Markov sequences (or HMMs) is a challenging direction.

Querying probabilistic data by automata has been explored recently [7] in the context of probabilistic XML, where the studied problem is that of computing the probability of acceptance by a tree automaton. There are inherent differences between that work and ours. First, it is not at all clear whether their XML model can (efficiently) capture Markov sequences. Second, as exhibited in this paper, transducers introduce challenges that do not exist in automata (e.g., the data complexity of determining the probability of an answer). Finally, enumeration of answers does not arise there.

Towards the goal of tagging parts of speech, the work of Kempe [29] has combined HMMs and transducers in a different way: the HMM is *converted* into a transducer that heuristically approximates the behavior of the HMM within the task of tagging. In principle, such a conversion could be used as an alternative approach to querying HMMs, with the advantage that query answering could be done by means of composition of transducers. Note, however, that the notion of querying there is inherently different from the one of this paper, since the conversion into a transducer eliminates the probabilistic aspect of the setting. In contrast, our query evaluation is heavily based on the probabilistic behavior of the Markov sequence (or the HMM)—the probabilities are translated into confidence values of answers, and ranking thereof.

7. CONCLUSIONS

Towards the goal of obtaining strong querying capabilities in a Markov-sequence database, we studied the complexity of evaluating a transducer over a Markov sequence. In particular, we considered answer enumeration and confidence computation. We showed that the former is tractable, if one poses no restrictions on the order of answers. The latter, however, can be #P-hard in the case of nondeterministic transducers. Our main focus has been on the enumeration of answers in ranked order (by decreasing confidence), which in practice is much more desirable than unranked enumeration. Unfortunately, ranked enumeration is intractable, and even highly inapproximable; theoretically, the best one can do is to apply our heuristic for enumerating by decreasing $E_{\max}$. The class of s-projectors allows for better approximations, and in the case of indexed s-projectors, ranked enumeration is actually tractable.

Table 2 summarizes our complexity results. Each column corresponds to a class of transducers. The first row shows the complexity of computing the confidence of an answer. In each cell, the upper part is the query-and-data complexity, and the bottom one refers to the data complexity. Here, hardness under data complexity means *existence* of a transducer for which the problem is hard. The second row shows the best approximation (i.e., scoring function and ratio) presented for enumerating with polynomial delay. For example, "$I_{\max}$: n" means enumeration by decreasing $I_{\max}$, where the guaranteed approximation ratio is n. The term "PSPACE" means that the enumeration is with polynomial space. Finally, the third row shows the lower bounds we gave for the problem of approximating the top answer. In this row, all the lower bounds hold already under data complexity, except for "uniform emission," where hardness applies to transducers with only one state, but with an unbounded alphabet Σ_A.

This work has been restricted to transducers with deterministic emission. In the course of our research, we found that without this restriction almost every basic problem is computationally hard (even in the presence of other, quite severe restrictions). We postpone to future work the challenge of identifying interesting cases of nondeterministic emission to where our positive results extend. This work gives rise to quite a few additional future directions. Important subjects are approximating the confidence of an answer, and an average-case analysis of approximate ranked evaluation. There is the practical challenge of effectively implementing the proposed algorithms. In particular, we intend to combine these algorithms in the Lahar system, so as to strengthen its querying capabilities with transducers.

Acknowledgments

We thank Sara Cohen for providing valuable comments and suggestions. We also thank Nimrod Megiddo and Evan Welbourne for useful discussions.

8. REFERENCES

[1] A. J. Bonner and G. Mecca. Sequences, datalog, and transducers. *J. Comput. Syst. Sci.*, 57(3):234–259, 1998.

[2] A. J. Bonner and G. Mecca. Querying sequence databases with transducers. *Acta Inf.*, 36(7):511–544, 2000.

[3] J. Boulos, N. N. Dalvi, B. Mandhani, S. Mathur, C. Ré, and D.Suciu. MYSTIQ: a system for finding more answers by using probabilities. In *SIGMOD Conference*, pages 891–893, 2005.

[4] M.-Y. Chen, A. Kundu, and J. Zhou. Off-line handwritten word recognition using a hidden Markov model type stochastic network. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):481–496, 1994.

[5] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating probabilistic queries over imprecise data. In *SIGMOD Conference*, pages 551–562, 2003.

[6] S. Cohen, B. Kimelfeld, and Y. Sagiv. Generating all maximal induced subgraphs for hereditary and connected-hereditary graph properties. *J. Comput. Syst. Sci.*, 74(7):1147–1159, 2008.

[7] S. Cohen, B. Kimelfeld, and Y. Sagiv. Running tree automata on probabilistic XML. In *PODS*, pages 227–236. ACM, 2009.

[8] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[9] N. N. Dalvi and D. Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *PODS*, pages 293–302, 2007.

[10] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[11] Y. Diao, B. Li, A. Liu, L. Peng, C. Sutton, T. Tran, and M. Zink. Capturing data uncertainty in high-volume stream processing. In *CIDR*, 2009.

[12] R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

[13] R. Durbin, S. R. Eddy, A. Krogh, and G. J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[14] D. Eppstein. Finding the k shortest paths. *SIAM J. Comput.*, 28(2):652–673, 1998.

[15] B. Escoffier and V. T. Paschos. Differential approximation of min sat, max sat and related problems. In *ICCSA (4)*, volume 3483 of *Lecture Notes in Computer Science*, pages 192–201. Springer, 2005.

[16] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[17] C. H. Fosgate, H. Krim, W. W. Irving, W. C. Karl, and A. S. Willsky. Multiscale segmentation and anomaly enhancement of SAR imagery. *IEEE Transactions on Image Processing*, 6(1):7–20, 1997.

[18] J. F. Gantz, D. Reinsel, C. Chute, W. Schlichting, J. Mcarthur, S. Minton, I. Xheneti, A. Toncheva, and A. Manfrediz. The expanding digital universe: A forecast of worldwide information growth through 2010, March 2007.

[19] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *FOCS*, pages 627–636. IEEE Computer Society, 1996.

[20] HMMER. Biosequence analysis using hidden Markov models. `http://hmmer.janelia.org/`.

[21] HTK. The hidden Markov toolkit, October 2009. `http://htk.eng.cam.ac.uk/`.

[22] J. Huang, L. Antova, C. Koch, and D. Olteanu. MayBMS: a probabilistic database management system. In *SIGMOD Conference*, pages 1071–1074, 2009.

[23] G. Jirásková. State complexity of some operations on binary regular languages. *Theor. Comput. Sci.*, 330(2):287–298, 2005.

[24] D. Johnson, M. Yannakakis, and C. Papadimitriou. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.

[25] B. Kanagal and A. Deshpande. Online filtering, smoothing and probabilistic modeling of streaming data. In *ICDE*, pages 1160–1169, 2008.

[26] B. Kanagal and A. Deshpande. Efficient query evaluation over temporally correlated probabilistic streams. In *ICDE*, pages 1315–1318. IEEE, 2009.

[27] B. Kanagal and A. Deshpande. Indexing correlated probabilistic databases. In *SIGMOD Conference*, pages 455–468, 2009.

[28] S. Kannan, Z. Sweedyk, and S. R. Mahaney. Counting and random generation of strings in regular languages. In *SODA*, pages 551–557, 1995.

[29] A. Kempe. Finite state transducers approximating hidden Markov models. In *ACL*, pages 460–467, 1997.

[30] B. Kimelfeld, Y. Kosharovsky, and Y. Sagiv. Query efficiency in probabilistic XML models. In *SIGMOD Conference*, pages 701–714, 2008.

[31] B. Kimelfeld and C. Ré. Transducing Markov sequences (extended version). Accessible from the second author's home page (`http://pages.cs.wisc.edu/~chrisre/`), 2010.

[32] B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pages 173–182. ACM, 2006.

[33] B. Kimelfeld and Y. Sagiv. Maximally joining probabilistic data. In *PODS*, pages 303–312. ACM, 2007.

[34] B. Kimelfeld and Y. Sagiv. Efficiently enumerating results of keyword search over data graphs. *Inf. Syst.*, 33(4-5):335–359, 2008.

[35] C. Koch. Approximating predicates and expressive queries on probabilistic databases. In *PODS*, pages 99–108, 2008.

[36] C. Koch. A compositional query algebra for second-order logic and uncertain databases. In *ICDT*, pages 127–140, 2009.

[37] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, pages 282–289, 2001.

[38] E. L. Lawler. A procedure for computing the k best solutions to discrete optimization problems and its

application to the shortest path problem. *Management Science*, 18:401–405, 1972.

[39] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Access methods for Markovian streams. In *ICDE*, pages 246–257, 2009.

[40] J. Letchner, C. Ré, M. Balazinska, and M. Philipose. Lahar demonstration: Warehousing Markovian streams. *PVLDB*, 2(2):1610–1613, 2009.

[41] B. Ludäscher, P. Mukhopadhyay, and Y. Papakonstantinou. A transducer-based XML query processor. In *VLDB*, pages 227–238. Morgan Kaufmann, 2002.

[42] W. Martens and F. Neven. Typechecking top-down uniform unranked tree transducers. In *ICDT*, volume 2572 of *Lecture Notes in Computer Science*, pages 64–78. Springer, 2003.

[43] K. G. Murty. An algorithm for ranking all the assignments in order of increasing costs. *Operations Research*, 16:682–687, 1968.

[44] C. H. Papadimitriou and M. Yannakakis. On the complexity of database queries. *Journal of Computer and System Sciences*, 58(3):407–427, 1999.

[45] J. S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12(4):777–788, 1983.

[46] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[47] C. Ré, J. Letchner, M. Balazinska, and D. Suciu. Event queries on correlated probabilistic streams. In *SIGMOD Conference*, pages 715–728, 2008.

[48] C. Ré and D. Suciu. Approximate lineage for probabilistic databases. *PVLDB*, 1(1):797–808, 2008.

[49] A. D. Sarma, M. Theobald, and J. Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *ICDE*, pages 1023–1032, 2008.

[50] P. Seshadri, M. Livny, and R. Ramakrishnan. SEQ: A model for sequence databases. In *ICDE*, pages 232–239. IEEE Computer Society, 1995.

[51] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *HLT-NAACL*, 2003.

[52] F. Sha and L. K. Saul. Large margin hidden Markov models for automatic speech recognition. In *NIPS*, pages 1249–1256, 2006.

[53] S. Singh, C. Mayfield, R. Shah, S. Prabhakar, S. E. Hambrusch, J. Neville, and R. Cheng. Database support for probabilistic attributes and tuples. In *ICDE*, pages 1053–1061, 2008.

[54] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 21(2):316–328, 1992.

[55] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. J. Shenoy. Probabilistic inference over RFID streams in mobile environments. In *ICDE*, pages 1096–1107, 2009.

[56] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[57] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146. ACM, 1982.

[58] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[59] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17:712–716, 1971.

[60] S. Zachos. Probabilistic quantifiers and games. *Journal of Computer and System Sciences*, 36(3):433–451, 1988.