Querying Probabilistic Business Processes for Sub-Flows

Daniel Deutch
Tel Aviv University *
danielde@cs.tau.ac.il

ABSTRACT

This paper studies top-k query evaluation for an important class of probabilistic semi-structured data: nested DAGs (Directed Acyclic Graphs) that describe possible execution flows of Business Processes (BPs for short). We consider queries with *projection*, that select portions (sub-flows) of the execution flows that interest the user and are most likely to occur at run-time. Retrieving common sub-flows is crucial for various applications such as targeted advertisement and BP optimization. Sub-flows are ranked here by the *sum* of likelihood of EX-flows in which they appear, in contrast to the max-of-likelihood semantics studied in previous work; we show that while sum semantics is more natural, it makes query evaluation much more challenging. We study the problem for BPs and queries of varying classes and present efficient query evaluation algorithms whenever possible.

Categories and Subject Descriptors

H.2.3 [**Database Management**]: [Languages]; F.4.2 [**Theory of Computation**]: [Grammars and Other Rewriting Systems]; G.3 [**Mathematics of Computing**]: [Probability and Statistics]

General Terms

Algorithms, Languages, Theory

1. INTRODUCTION

A Business Process (BP for short) consists of a set of activities which, when combined in a flow, achieve some business goal. A given BP may have a large, possibly infinite, number of possible execution flows (EX-flows for short), each having a certain probability to occur at run time. BPs are typically designed via high-level specifications [6] which are later compiled into an executable code. Since the BP logic is captured by the specification, tools for querying and analyzing the possible EX-flows of a BP specification, as well as their probability to occur at run time, are extremely valuable to companies [11, 12, 18].

This paper studies evaluation of top-k *projection* queries that select portions of EX-flows that are of interest to the user, and are most likely to occur at run-time. Before presenting our results, let us briefly recall our model (from [13]) for probabilistic BPs, and motivate the specific query semantics studied here.

BP specifications. A BP specification is abstractly modeled as a nested DAG consisting of activities (nodes), and links (edges) between them, that detail the execution order of the activities [11, 14]. For example, consider a BP of an on-line travel agency. The BP specification may include activities (nodes) for flight and hotel reservation, car rental, payment and confirmation services, and edges that describe their execution order. The DAG shape allows to describe parallel computations. For instance, advertisements may be injected in parallel to the search. BP activities may be either atomic, or compound. In the latter case their possible internal structures, called *implementations*, are also detailed as DAGs, leading to the nested DAG structure. A compound activity may have different possible implementations, corresponding to different user choices, variable values, servers availability, etc. An *Execution Flow* (abbr. EX-flow) is then an actual running instance of a BP, obtained by choosing a single implementation for each compound activity. A BP specification induces a set of such possible EX-flows; this set may be large, or even infinite when the BP specification contains *recursion*.

In practice, some EX-flows are more common than others. This is modeled by a probability distribution over the possible implementations of compound activities [12, 13]. A BP specification along with a description of such distribution is called a *Probabilistic BP*. We note that the probabilities of choices dictating the execution course are, in typical cases, *inter-dependent*. To simplify the presentation we first assume probabilistic independence between the choices made throughout the EX-flow and present our results in this setting. We then study how to extend the results to a settings where dependencies are allowed (in a bounded manner).

Top-k projection queries. Among all possible flows of a BP, analysts are often interested only in a part that is relevant for their analysis. This part is typically described via a query. Note, however, that among all qualifying execution

*Supported by the Israeli Ministry of Science through the Eshkol grant

ICDT 2011, March 21–23, 2011, Uppsala, Sweden.

(sub-) flows, some are often more "interesting" than others.

In particular, given a BP specification, identifying the top-k execution (sub-) flows that are most likely to occur in practice, out of those satisfying the query criteria, is crucial for various applications. It can be used, for instance, to adjust the BP web-site design to the needs of certain user groups, or to personalize on-line advertisements. The importance of top-k query evaluation is enhanced by the fact that the number of answers (qualifying sub-flows) to a simple query may be extensively large, or even infinite when the BP contains recursion [2].

Re-consider our example BP of an on-line, Web-based travel agency. An analyst of this BP may wish to find out *how is one likely to book a travel package containing a flight reservation?*, or *how is this likely to be done for travelers of a particular airline company, say British Airways?*. There may be many different ways for users to book such travel packages. But assume, for instance, that we obtain that a likely scenario for British Airways reservations is one where users first search for a package containing both flights and hotels, but eventually do not reserve an hotel. Such a result may imply that the combined deals suggested for British Airways fliers are unappealing, (as users are specifically interested in such deals, but refuse those presented to them), and can be used to improve the Web site.

We note that the score assigned to each sub-flow (query result) is defined here as the *sum* of probabilities of full EX-flows in which it appears. This coincides with the intuitive definition of likelihood of a sub-flow; but we also discuss other possible scoring functions below.

Our results. The present paper is the first to provide top-k query evaluation algorithms, for projection queries over probabilistic BPs, with the sum-of-likelihoods semantics. Our analysis offers a nearly complete picture of which combinations of BP and query features lead to PTIME algorithms and which to NP-hard or infeasible problems. We next describe the main results; in the following we refer to data complexity, i.e. complexity with respect to the size of the BP specification. This is because our query evaluation algorithms all incur time that is exponential in the query size (when considering a fixed-size BP specification).

To simplify the presentation we first assume probabilistic independence between the choices made throughout the execution flow and present our algorithms and complexity results in this setting.

First, observe that the number of possible execution flows of a given BP may be not only large but infinite, if the BP contains recursion. Hence enumerating them all, to sum up the likelihoods of relevant flows, is clearly not an option. We nevertheless show that even when the BP contains recursion, it is possible under plausible assumptions to identify the top-k query answers (sub-flows) that are most likely to occur at run-time. We present an EXPTIME algorithm for finding the top-k query answers and show that, unless P=NP, a PTIME algorithm does not exist.

Without recursion, the number of possible flows is finite and exact likelihoods can be computed. However, we show that query evaluation may still, in general, be impossible to perform in PTIME (unless P=NP). The exact complexity depends on the query characteristics. The query language that we consider selects execution (sub) flows of interest, using execution patterns. Execution patterns are an adaptation of the tree-patterns offered by existing query languages for XML, to BP nested DAGs. In particular, execution patterns may include regular/transitive edges, that are matched to simple edges/paths of the execution flow. We show that when the projection is not over transitive edges, query evaluation can be performed in PTIME under unit-cost RAM model with exact rational arithmetic (i.e., algebraic operations on rational numbers can be done in unit time) [5]. This computational model is used here for the same reasons it was used in [15]: to avoid worrying about representation of query result probabilities, since in general these probabilities may be exponentially small with respect to the input BP size.

Our results described so far referred to the case where all probabilistic choices are independent. Then, we also consider dependencies: we show that query evaluation is still possible, in the practical case where only a bounded level of dependency is allowed (to be formally defined below). In this case, an overhead which is (double-)exponential in the above bound is added to the complexity of query evaluation in the cases where query evaluation was PTIME (EXPTIME).

While our work is motivated by a particular application domain, our evaluation algorithms for projection queries are notably applicable to other important settings where the analysis of potentially infinite sets of graphs, generated probabilistically, is needed. This includes XML schemas with probabilities for generating synthetic test documents [8], Active XML [1] where embedded calls to Web services may have some probabilities for the possible returned answers, and other graph-based models for probabilistic processes, discussed below.

Difficulties and Novelty. We conclude the Introduction by contrasting our work with related work.

Recursive Markov Chains and Probabilistic XML. Recursive Markov Chains (RMCs) [15] are extensions of Markov Chains (MCs) [21], that allow for recursion. An RMC consists of a collection of Markov chains which can call each other in a possibly recursive manner. RMCs may also be thought of as an adaptation of Recursive State Machines [3] to a probabilistic settings. In a recent work [4], the authors show that RMCs can be used as a representation system for probabilistic XML, that subsumes most of the previous models for Probabilistic XML (e.g. [19, 9, 24]).

The analysis of RMCs was first studied in [15], in the context of reachability and termination; the authors showed that in the general case these problems are intractable (even approximation is at least as difficult as SQRT-SUM [17], which is conjectured not to have a PTIME solution). But they provide PTIME algorithms for a large class of restricted case (including the case of 1-exit RMC [15]). The work of [4] then studies algorithms for a very powerful query language, namely Monadic Second Order (MSO) Logic (again, considering various restricted cases of RMCs). They provide algorithms for MSO query evaluation that, while tractable w.r.t. data complexity (the RMC size), have non-elementary complexity in the query size and are thus impractical (but this is the best that can be done for MSO [10]). [4] further considers more restricted query languages such as tree patterns and XPath, and shows that they incur lower combined complexity.

When no dependencies between probabilistic choices made during execution of the process are allowed, our model for

probabilistic BPs can be syntactically translated to 1-exit RMCs (and vice versa). However, there are three important distinctions between the current work and the works described above. A first distinction is the query language used for analysis. The above works study only *boolean queries*, whose output is a single probability value. In contrast, our work studies *top-k projection queries*, where the output consists of a set of best ranked answers. We study evaluation of boolean queries as a tool, and show that the transition from boolean to projection queries in this context requires non-trivial development, reflected by inherently higher complexity of query evaluation. Second, evaluation of boolean queries in our context also requires additional development: the work of [19] cannot be used here as it does not allow for recursion, the work of [4] for MSO queries requires much higher (non-elementary vs. the EXPTIME in our work) combined complexity, and the weaker query languages (tree patterns and XPath) studied in [4] cannot capture the DAG-shaped patterns expressible in our query language. Third, previous work in this context assumes *independence* between probabilistic choices, while our model and query evaluation algorithms account for dependencies, which occur very regularly in the context of Business Processes.

Business Processes. Probabilistic BPs were introduced in [12, 13]. Query evaluation there was based on a particular *max* semantics: [12] suggested an algorithm for identifying *full flows* with maximal probability. In [13] we extended the algorithm of [12] to rank sub-flows based on the *maximal* probability of full flows where they appear. This choice of ranking was motivated in [13] by a specific application concerning the recovery of missing information. Recall that the current work ranks sub-flows based upon the *sum* of probabilities of flows in which they appear. To illustrate the difference between *max* and *sum* semantics, consider a case where a particular deal consisting of a flight and a car rental, is very popular, but where packages consisting of flight and hotel reservations are *overall* more common (even though each such offer is individually less popular than the specific flight+car deal). Now, consider the query above that wishes to identify how a package that includes a flight reservation is typically booked. With *sum* semantics, the flight+hotel option is ranked highest, as it appears in most EX-flows. But with *max* semantics, flight+car would be ranked highest, as there exists one very popular EX-flow where it appears.

The *sum* semantics is a common semantics for projection queries, employed also for probabilistic XML and probabilistic relational DB (e.g. [23, 16]). Yet, we show below that in the BP context it makes query evaluation *computationally much harder*, compared to *max* semantics (NP-hard and EXPTIME vs. the PTIME of [12, 13]). Intuitively, this is because the computation now has to implicitly compute sum-of-probabilities over all relevant flows (and there may be infinitely many).

Other Probabilistic Process Models. There are many works on analysis of various other models for probabilistic process specifications (e.g. Probabilistic Pushdown Automata [20] and Stochastic Context Free (Graph) Grammars [10]). The discussion above for works on RMCs holds also for the analysis work on (restricted versions of) these models: to our knowledge, these works consider only boolean queries; furthermore, they either study evaluation of MSO queries (as in e.g. [10]) and then suffer from non-elementary combined complexity, or consider very restricted kinds of analysis. In contrast, our query language suggests a reasonable tradeoff between expressibility and feasibility.

Paper Organization. In Section 2 we recall the model of probabilistic BPs and queries over such BPs; in Section 3 we study the complexity of query evaluation, providing lower and upper bounds. In Section 4 we extend our results to account for dependencies. We conclude in Section 5.

2. PRELIMINARIES

We start by recalling the basic model of [11] for probabilistic BPs, EX-flows and queries.

BP specification. At a high-level, a BP specification encodes a set of activities and the order in which they may occur. A BP specification is modeled as a set of node-labeled DAGs. Each DAG has a unique start node with no incoming edges and a unique end node with no outgoing edges. Nodes are labeled by activity names and directed edges impose ordering constraints on the activities. Activities that are not linked via a directed path are assumed to occur in parallel. The DAGs are linked through implementation relationships; the idea is that an activity a in one DAG is realized via the activities in another DAG. We call such an activity *compound* to differentiate it from *atomic* activities which have no implementations. Compound activities may have multiple possible implementations, and the choice of implementation is controlled by a a condition referred to as a *guarding formula*.

We assume the existence of two domains: $\mathcal{A} = \mathcal{A}_{atomic} \cup \mathcal{A}_{compound}$ of activity names and $\mathcal{F}$ of formulas in predicate calculus.

DEFINITION 2.1. *A* BP specification *is a triple* (S, s_0, τ)*, where* S *is a finite set of node-labeled DAGs,* $s_0 \in S$ *is a distinguished DAG consisting of a single activity, called the* root*,* $\tau : \mathcal{A}_{compound} \rightarrow 2^{S \times \mathcal{F}}$ *is the* implementation function*, mapping each compound activity name in* S *to a* set *of pairs, each consisting of an implementation (a DAG in* S*) and a guarding formula in* $\mathcal{F}$*.*

Each DAG d *in* S *has a unique start (end) node with no ingoing (outgoing) edges, denoted start*(d) *(end*(d)*).*

EXAMPLE 2.2. *Fig. 1 depicts a partial BP specification. Its root DAG consists of a single activity* `chooseTravel`*.* `chooseTravel` *is a compound activity having 3 possible implementations* F_2, F_3, F_4*. These correspond to different choices of travel search (flights only, flights + hotels, or flights + hotels + cars) and are guarded by corresponding formulas. The idea is that exactly one formula is satisfied at run-time (the user chooses one of the three search types) and thus* `chooseTravel` *is implemented either by* F_2*,* F_3 *or* F_4*. Consider for example* F_1*; it describes a group of activities comprising user login, the injection of an advertisement, the* `Flights` *activity, and the* `Confirm` *activity. Directed edges specify the order of activities occurrence, e.g. users must login before choosing a flight. Some of the activities (e.g.* `Advertise` *and* `Flights`*) are not in a particular order, and thus may occur* in parallel. `Login` *and* `Advertise` *are atomic whereas* `Flights` *and* `Confirm` *are compound. Note that the specification is recursive as e.g.* F_2 *may call* F_1*.*

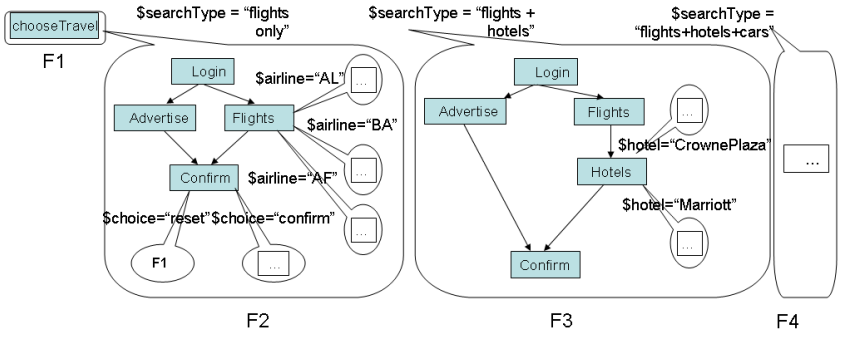

Figure 1: Business Process Specification

We note that satisfaction of guarding formulas is determined by external factors, such as user choices. We assume that exactly one guarding formula can be satisfied when determining the implementation of a given compound activity occurrence, but satisfaction of guarding formulas can change if activities occur several times. For instance, a user may choose to search for flights and hotels the first time she goes through F_1 and for flights only the second time.

Execution Flows. An EX-flow is modeled as a nested DAG that represents the execution of activities from a BP. Since, in real-life, activities are not instantaneous, we model each occurrence of an activity a by two a-labeled nodes, the first standing for the activity *activation* and the second for its *completion* point. These two nodes are connected by an edge. The edges in the DAG represent the ordering among activities activation/completion and the implementation relationships. To emphasize the nested nature of executions, the implementation of each compound activity appears in-between its activation and completion nodes. An EX-flow structure must adhere to the structure of the BP specification, i.e., activities must occur in the same order and implementation relationships must conform to τ.

DEFINITION 2.3. *Given a BP specification $s = (S, s_0, \tau)$, e is an execution flow (EX-flow) of s if:*

- **Base EX-Flow:** *e consists only of the activation and completion nodes of the root activity s_0 of s, connected by a single edge, or,*
- **Expansion Step:** *e' is an EX-flow of s, and e is obtained from e' by choosing some activation-completion pair (n_1, n_2) of an activity a in e', choosing a pair of implementation and guard $(e_a, f_a) \in \tau(a)$, adding to e' the nodes and edges of e_a as well as two new edges, called* implementation edges, *$(n_1, \texttt{start}(e_a))$ and $(\texttt{end}(e_a), n_2)$, and finally annotating the pair (n_1, n_2) with the formula f_a.*

  We require that (n_1, n_2) do not have any implementation edges already connected to them in e', whereas all its ancestor compound activities in e' do have such implementation edges.

  In the attached implementation e_a, each node is replaced by a corresponding pair of activation and completion nodes, connected by an edge.

  We call e_a a direct implementation *of (n_1, n_2) and call e an* expansion *of e', denoted $e' \to e$.*

We use $e' \to^* e$ to denote that e was obtained from e' by a sequence of expansions. An activity pair in e is *unexpanded* if it is not the source of an implementation edge. We say that an EX-flow is *partial* if it has unexpanded activities, and *full* otherwise, and denote the set of all full flows of a BP specification s by $flows(s)$.

For a graph e, e is an EX-flow if it is a (partial or full) flow of some BP specification s. Last, an *abstract* EX-flow e' is obtained from an EX-flow e by deleting some occurrences of guarding formulas from e.

EXAMPLE 2.4. *Two EX-flows of the travel agency BP are given in Fig. 2. Ordering edges (implementation edges) are drawn by regular (resp. dashed) arrows. Each EX-flow describes a sequence of activities that occurred during the BP execution. In Fig. 2(a) the user chooses a "flights+hotels" search, reserving a "British Airways" flight and a "Marriott" hotel, then confirms. Fig. 2 (b) depicts another possible EX-flow, where the user chooses a "flights only" search, followed by a choice of British Airways flight, but then resets and makes other choices (omitted from the figure).*

Likelihood. Some user choices / variable values are more common than others, and thus EX-flows vary in their likelihood of occurring in practice. To model this we use two likelihood functions. The first, named c–likelihood (choice likelihood), associates a value with each guarding formula (implementation choice), describing the probability that the formula holds. The second, named f–likelihood (flow likelihood) reflects the joint likelihood of satisfaction of guarding formulas occurring along the flow. Formally,

DEFINITION 2.5. *Given a BP s with root s_0 and a c–likelihood function δ, the* f–likelihood *Δ of an EX-flow e of s (w.r.t. δ) is defined as follows:*

1. *If e consists only of the activation and completion nodes of the root activity s_0, $\Delta(e) = 1$.*
2. *Else, if $e' \to e$ for some EX-flow e' of s, then $\Delta(e) = \Delta(e') \times \delta(f)$, where f guards the implementation that is added to e' to form e.*

Note that the likelihood functions considered here assume *independence* between choices; that is, the likelihood of every formula to hold is constant, regardless of truth values of other formulas / implementation choices taken. In practice, choices may be dependent: for instance, the choice of hotel may depend on the choice of airline that preceded it. In Section 5 we explain how our results extend to the general context where choices may be dependent.

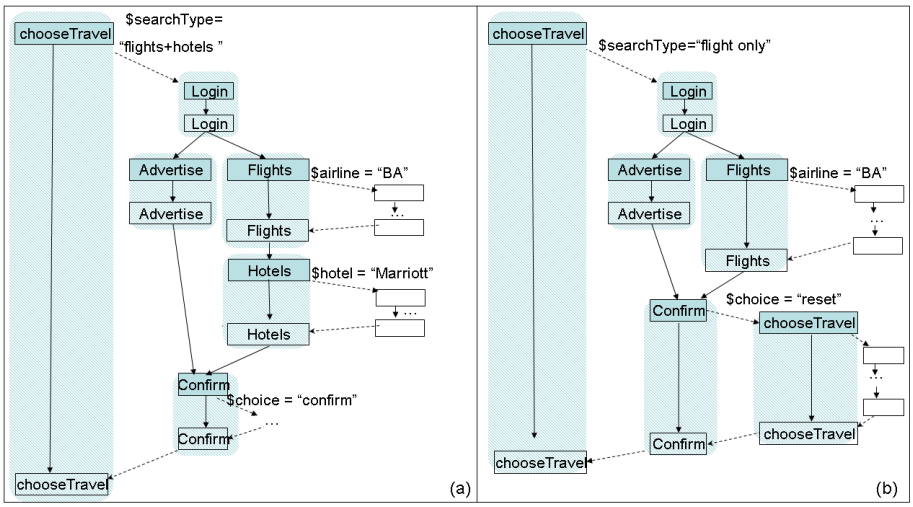

Figure 2: Execution Flows

EXAMPLE 2.6. *Consider Table 1 that depicts the likelihoods of value assignments for the different variables of the travel agency BP, and consequently the c–likelihood function for the corresponding guarding formulas. The f–likelihood of each EX-flow may be computed according to this c–likelihood function, For instance, the f–likelihood of the EX-flow in Fig.2(a) is computed as the multiplication of the c–likelihood values of* $\$searchType$ = "$flights + hotels$", $\$airline$ = "$BritishAirways$", $\$hotel$ = "$Marriott$", *and* $\$choice$ = "$confirm$", *that is* $0.25 * 0.7 * 0.6 * 0.2 = 0.021$.

Queries. Queries are defined using *execution patterns*, an adaptation of the tree-patterns of existing XML query languages, to nested EX-flow DAGs. Such patterns are similar in structure to EX-flows, but contain *transitive edges* that match any EX-flow *path*, and *transitive activity nodes*, for searching deep within the implementation subgraph, of the corresponding compound activities, at any level of nesting. Nodes/ formulas in the pattern may be labeled by the wildcard ANY and then may match any EX-flow node/formula.

DEFINITION 2.7. *An* execution pattern, *(abbr. EX-pattern), is a pair* $p = (\hat{e}, T)$ *where* $\hat{e}$ *is an abstract EX-flow with node labels in* $\mathcal{A} \cup \{\text{ANY}\}$ *and* T *is a set of activity pairs and edges in* $\hat{e}$, *called* transitive *activities and edges, resp.*

EXAMPLE 2.8. *An example EX-pattern is given in Figure 3(a) (ignore for now the rectangle surrounding a sub-graph of the pattern). It describes EX-flows that contain some "British Airways" (abbr. "BA") flight search that resulted in a confirmation. The dashed edges are* transitive edges. *The doubly bounded chooseTravel nodes are* transitive nodes. *Intuitively, the transitive edge connected to chooseTravel may match any sequence of searches and resets, and the transitive edge connecting the Flights and Confirm activities may match any sequence of search activities (for hotels and/or cars). The transitivity of the chooseTravel node allows this matching to include indirect implementation of the corresponding composite node, at any nesting level.*

The matching of an EX-pattern to an EX-flow is called an *embedding*.

$searchType	$P(\$searchType)$	$airline	$P(\$airline)$
flights only	0.5	BA	0.7
flights+hotels	0.25	AF	0.2
flights+hotels+cars	0.25	AL	0.1
$hotel	$P(\$hotel)$	$choice	$P(\$choice)$
Marriott	0.6	reset	0.6
HolidayInn	0.3	confirm	0.2
CrownePlaza	0.1	cancel	0.2

Table 1: c–likelihood function

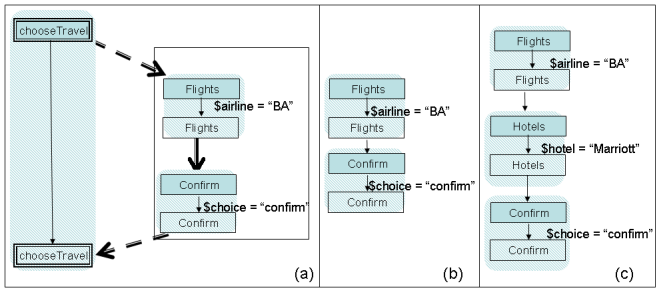

Figure 3: Query

DEFINITION 2.9. *Let* p *be an EX-pattern and let* e *be an EX-flow. An* embedding *of* p *into* e *is a homomorphism* ψ *from nodes and edges in* p *to nodes, edges and paths in* e *s.t.*

1. **[root]** *the root of* p *is mapped to the root of* e.
2. **[nodes]** *activity pairs in* p *are mapped to activity pairs in* e, *preserving node labels and formulas; a node labeled by* ANY *may be mapped to nodes with any activity name. For non-transitive compound activity pairs in* p, *nodes in their direct implementation are mapped to nodes in the direct implementation of the corresponding activity pair in* e.
3. **[edges]** *each (transitive) edge from node* m *to* n *in* p *is mapped to an edge (path) from* $\psi(m)$ *to* $\psi(n)$ *in* e.

We are now ready to define projection queries and their results.

DEFINITION 2.10. *A projection query* $q = (p, P)$ *consists of an execution pattern* p *accompanied by a sub-graph* P *of*

the pattern, itself forming an execution pattern, called the projected *part of the pattern.*

Given a BP specification s and a projection query $q = (p, P)$, let Ψ be the set of all possible embeddings ψ of the pattern p to EX-flows in $flows(s)$, and let $\Psi{\downarrow_P}$ be the set obtained from Ψ by restricting each embedding to the nodes and edges of P. Two (restricted) embeddings in $\Psi{\downarrow_P}$ are considered *equivalent* if for every node/edge x in P, $\psi(x)$ and $\psi'(x)$ are isomorphic up to node identifiers. The result of q on s, denoted $q(s)$, is a set consisting of one representative for each such equivalence class. (The result is unique up to the equivalence relation defined above).

We note that $q(s)$ can be infinite. Also, each answer $\alpha \in q(s)$ may have infinitely many *origins*, namely, there may be an infinite number of EX-flows $e \in flows(s)$ with an embedding ψ (of p to e) whose restriction to the output node and edges is equivalent to α.

The *likelihood* of an answer $\alpha \in q(s)$ is the sum of likelihood of its origins, namely

$$likelihood(\alpha, q, s) = \sum\{\Delta(e) \mid e \in flows(s) \wedge e \text{ is an origin of } \alpha\},$$

where Δ is the f–likelihood function for s. When s is clear from the context we will omit it and simply use $likelihood(\alpha, q)$. If α is not in the set $q(s)$ we say that $likelihood(\alpha, q) = 0$.

EXAMPLE 2.11. *Let us consider again the EX-pattern in Figure 3(a), this time as a query, with the rectangle denoting its projected part. Note that the projection focuses on the execution sub-flows that may occur between the point where a user chooses a "BritishAirways" flight and the final confirmation of her reservation. Note that, due to recursive nature of the BP, there are in general an infinite number of such possible sub-flows (query answers) - a user may reset and restart the search an unbounded number of times. Two possible answers to the query appear in Fig. 3 (b) and (c). The first answer corresponds to users that choose at some point a "flightsOnly" search, pick a "BA" flight and then immediately confirm. The second answer corresponds to users that choose at some point a "flights+hotels" search, pick "BA" as airline and Marriott as hotel, and confirm.*

Observe that each of these answers may have infinitely many origin EX-flows. The likelihood of each answer is the sum of likelihoods of all these origins. Let us compute for instance the likelihood of answer (b). To compute the likelihood here, we define a variable $x_{\texttt{ans}}$ *that reflects the likelihood of a match for the answer. Intuitively,* $x_{\texttt{ans}}$ *is the chance that* `ans` *can be matched already in the first reservation choice of the user, or alternatively that the user cancels and then a match is obtained in the following reservation choices. Thus*

$$\begin{aligned} x_{\texttt{ans}} = \;& c\text{–likelihood}\ (\$searchType = \text{“}flightsOnly\text{”}) \\ & * \ c\text{–likelihood}\ (\$airline = \text{“}BA\text{”}) \\ & * \ c\text{–likelihood}\ (\$choice = \text{“}confirm\text{”}) \\ & + \ c\text{–likelihood}\ (\text{“}reset\text{”}) * x_{\texttt{ans}}. \end{aligned}$$

Namely, $x_{\texttt{ans}} = 0.5 * 0.7 * 0.2 + 0.6 * x_{\texttt{ans}} = 0.07 + 0.6 * x_{\texttt{ans}}$. *Thus* $x_{\texttt{ans}} = 0.07/0.4 = 0.175$.

Given a BP specification s, a projection query q and a number k, we use $top{-}k(q, s)$ to denote the set of k answers in $q(s)$ having highest likelihood values.[1] `TOP-K-ANSWERS` is then the problem of identifying, given q,s and k, the set $top{-}k(q, s)$.

[1]Observe that since distinct EX-flows may have the same likelihood value, this set may not be unique, in which case we pick one such set arbitrarily.

3. QUERY EVALUATION

We study in this section the complexity of evaluating projection queries.

We start by presenting *lower bounds*: first, we show that top-k query evaluation is $\sharp P$-hard even for non-recursive BPs. Then, since our techniques will be based on computing (or approximating) the score of individual answers and then identifying the top-k out of them, we consider lower bounds specific to the first part. We show that for general recursive specification, computing the exact scores of individual answers may be impossible (since these scores may be irrational), and even their approximation is hard in general.

Then we consider query evaluation algorithms, and show that an EXPTIME query evaluation algorithm is possible, in restricted cases when computing the exact scores of individual answers is possible, or when they can be approximated and the different answer scores are different by at least some fixed ϵ. Under additional constraints we then show a PTIME query evaluation algorithm.

3.1 Lower Bounds

Unfortunately, we can show that the problem is hard, even for the restricted case where the Business Process is non-recursive. This stems from the unique nested DAG structure of BP specifications.

The following theorem holds:

THEOREM 3.1. `TOP-K-ANSWERS` *is* $\sharp P$*-hard (under Turing computation model) in the BP size,* even for non-recursive specifications and for queries with no transitive nodes.

PROOF. The proof works by reduction from 3-SAT. Given a 3NF formula with C clauses and n variables, $x_1, \ldots, x_n$, we generate a BP specification s and a query q as follows: the BP root activity r has C implementations, each with c–likelihood of $1/C$. Each implementation represents a clause, and has a start-node labeled by an activity S, an end-node labeled by an activity E, and three sub-graphs connecting these two nodes, each corresponding to one literal of the clauses. The first activity in each such subgraph is labeled X_1. If x_1 appears in the corresponding literal positively (negatively), the node labeled X_1 has one outgoing edge, whose target is labeled T (F). Otherwise, the node labeled X_1 has two outgoing edges, to nodes labeled T and F. Each of these nodes has a single outgoing edge to a node labeled by an activity X_2, and this node again has one or two outgoing edges, depending if the variable x_2 appears in the given literal or not, and so on. (Thus all other X_i's in the subgraph have two children). The last T/F-labeled activities are the sources of edges whose targets are the end-node labeled by E. The EX-pattern of the query q consists of a root activity labeled r with implementation containing two activities, S and E, connected by a transitive edge. The projected part of q consists of all nodes and edges of the EX-pattern.

Note that an answer of q with respect to s is a path, and this path uniquely defines a truth assignment A for the variables $x_1, ..., x_n$: $A(x_i) = true$ ($= false$) if the X_i-labeled node is followed in the query answer by a node labeled by T (F). We claim that for any value of k, there exist at least k satisfying assignments to the formula if and only if there are at least k answers to q all having probability 1 (i.e. the top-k answers of q all have a probability of 1). To observe that this hold, note that every path that describes a satisfying assignment appears in all implementations of the

root (since each implementation includes all paths that are consistent with the corresponding claus), hence its likelihood is 1. On the other hand, for every path that describes a non-satisfying assignment, there is at least one implementation that does not contain it (corresponding to a non-satisfied clause), thus the likelihood value of such a path is less than 1. □

We will show in the sequel restricted cases that allows for a PTIME (data complexity) algorithm for TOP-K-ANSWERS in the unit-cost rational arithmetic model [5]. However, the above theorem indicates that the existence of a *general* PTIME solution in the unit-cost model is questionable, as 3-SAT (used in the above reduction) is a problem not known to be solvable in PTIME under the unit-cost rational arithmetic model.

The design of top-k algorithm requires a method for comparing the score of two possible answers. In principle, one possible such method may be based upon explicit computation of these scores, which in our case corresponds to computing the likelihood of a given answer (sub-flow) to appear in a random EX-flow. However, exact computation of the score turns out to be impossible here, as the following proposition holds:

PROPOSITION 3.2. *Given a BP specification s and an EX-flow ans, $likelihood(ans, s)$ may be irrational, even if all likelihoods in s are rational.*

The proof follows directly from results in [15], and is omitted for lack of space.

Interestingly, we will show below that (under plausible assumptions on the input), exact identification of top-k projection result is possible, using an *approximation* of the likelihoods of individual answers. However, we note that in general such an approximation cannot be done in PTIME (the proof follows from [15], and is omitted for lack of space):

PROPOSITION 3.3. *The existence of a PTIME (in the BP specification size) algorithm for approximating $likelihood(ans, s)$ up to an additive error of a given ϵ for every ans, s, implies that* $\texttt{SQRT}-\texttt{SUM} \in PTIME$.

SQRT-SUM is the problem of deciding, given natural numbers $(d_1, ..., d_n)$ and a natural number k, whether $\sum_{i=1,...,n} \sqrt{d_i} \leq k$, strongly believed not be solvable in PTIME under Turing Computation Model [17].

We note here that query evaluation was shown in [11] to be NP-hard with respect to the query size (i.e. when the BP specification size is fixed), even for testing emptiness of queries results, even for very simple queries and even for non-recursive BP specifications (see [11] for details).

We next consider upper bounds. We will construct a top-k query evaluation algorithm that uses, as a black box, an algorithm for approximating $likelihood(ans, s)$; we provide such an approximation algorithm in Section 3.3.

3.2 Upper Bounds

We start by describing the general framework for our evaluation algorithm.

General Framework. Given a projection query $q = (p, P)$ and a BP specification s, consider some (restricted) embedding α that assigns (1) activities names from s to the ANY labels of P, and (2) a sequence of labeled nodes to each transitive edge of P. Let $\alpha(p)$ denote the pattern obtained from p (including its non-projected part) by replacing each ANY label with the label dictated by α, and replacing each transitive edge in P with the path assigned to it by α (and treating each transitive node as a regular one). $\alpha(p)$ is possibly an answer to the query q (it may not be a sub-flow of any EX-flow of s, in which case it will not be an answer). It is easy to observe that every possible query answer may be obtained in such manner, for some embedding α. Thus, one may consider the following query evaluation algorithm: generate all possible assignments α; for each such α, generate the possible answer $\alpha(p)$, and compute the likelihood of $\alpha(p)$ to appear in a random EX-flow. Then retrieve the k answers having the greatest likelihood.

We note that this is the standard way of evaluating projection queries, employed for XML [19]. However, there are two challenges here, absent from the XML settings: (1) the number of possible assignments (and consequently the number of possible answers) is possibly infinite, and (2) as indicated by Proposition 3.2, exactly computing the likelihood of a given sub-flow is impossible in general, and we must use an approximation.

We next consider these two obstacles and explain how we tackle them.

Possibly infinite number of answers. Recall that the embeddings α considered above are assignments over (1) ANY-labeled nodes and (2) transitive edges. While the number of combinations for the first case (ANY labels) is bounded by $\mid s \mid^{\mid q \mid}$, the number of possible paths in a recursive BPs, and consequently the number of path assignments to transitive edges, may be *infinite*. This leads to infinitely many possible projection answers. To bound the number of assignments considered for transitive edges, we use the following "small world" Lemma:

LEMMA 3.4. *Given a BP s and a projection query $q = (p, P)$, there exists a set of top-k answers of q w.r.t. s where in each answer all the paths assigned to transitive edges in P are of length bounded by $\mid s \mid * k$.*

PROOF. By induction on k. Consider first $k = 1$. Assume that q is embedded by some embedding α in some EX-flow e, and let M be a path in e such that a transitive edge T of q is matched to M. Further assume that $|P| > |s|$. In particular, this means that M contains a recursive invocation of at least one activity $a \in S$, otherwise the length of M may not exceed the total number of nodes in implementation graphs of S. We construct an EX-flow e' that is obtained from e by subsequently omitting sub-flow of e that are rooted at recursive invocation of activities: first, we omit the sub-flow rooted at the recursive invocation of a. If there still exists a recursive invocation of some activity a' within M, we then omit the sub-flow rooted at a', and so forth, until any activity name appears at most once among the remaining nodes of the path M. We denote the remaining nodes and edges that originally were in M as M', and observe that M' is still a path (as we only removed connected sub-paths of M). Clearly, e' is a flow of S, and $f\!likelihood(e') \geq f\text{-}likelihood(e)$ due to the monotonicity of $f\!likelihood$. There exists an embedding α' of q in e', obtained from α by replacing the path M assigned to T by the new path M'. The same construction may be employed for each such path M,

and consequently we obtain the existence of a top-1 answer where each path assigned to T is of length bounded by $\mid s \mid$.

For $k > 1$, assume that there exists $k-1$ results with transitive edges mapped to paths of length bounded by $|s| * (k-1)$. Consider an embedding that assigns a path M of length greater than $|s| * k$. Then in particular, it contains more than k recursive invocations of compound activities. We may employ the same technique as above to shorten the path into a path M' that contains at most k such recursive invocations. M' does not appear as one of the $k-1$ results as its length is greater than $|s| * (k-1)$, but it is better weighed than M due to the monotonicity of *f-likelihood*, and may thus be used as the k'th-best result. □

Consequently, we can enumerate a set of EX-flows including all possible query answers.

No exact computation for scores of individual answers. As stated above (Proposition 3.2), in general there is no exact algorithm for computing the likelihood of a given answer. Still, under plausible assumptions on the input, we are able to perform exact evaluation of projection queries in such cases. We start by assuming the existence of some known ϵ such that for every two answers $\alpha, \alpha' \in q(s)$, either $likelihood(\alpha, q) = likelihood(\alpha', q)$, or $\mid likelihood(\alpha, q) - likelihood(\alpha', q) \mid > \epsilon$. In this case we say that the answers of q (w.r.t. s) have *separated likelihoods*, and refer to ϵ as the *separation factor* of q with respect to s. We note that similar assumptions are made in the context of top-k query evaluation over probabilistic *relational* data [23]. We consider below the implications of withdrawing the assumption.

The following Lemma holds

LEMMA 3.5. *For a BP s and a query q with separation factor ϵ, a set of top-k answers according to approximated likelihood values, up to $\epsilon/2$ precision, is also a set of top-k answers for to the exact likelihood values.*

We now have an algorithm for `TOP-K-ANSWERS`, in present of exact or approximated likelihood values for the possible answers: generate the (exponentially large) set of candidate answers based upon Lemma 3.4, then compute or approximate the likelihood of each such answer up to $\epsilon/2$, where ϵ is the separation factor, and declare the top-k answers.

Formally, we define `RANK-ANSWER` as the problem of computing, given a BP specification s, a query q and a possible projection answer α, $likelihood(\alpha, q, s)$. `APPROX-RANK-ANSWER` is the problem of approximating this value up to, at most, a given additive error. Then, following common practice, let F^L be the class of problems solvable in time complexity F, when given an oracle solving a problem L. We obtain:

THEOREM 3.6. *1. If there exists an oracle for* `RANK-ANSWER`*, then* $\texttt{TOP-K-ANSWERS} \subseteq EXPTIME^{\texttt{RANK-ANSWER}}$.

2. If separated likelihoods *are guaranteed with respect to the input BP and query, and there exists an algorithm for* `APPROX-RANK-ANSWER`*, then* $\texttt{TOP-K-ANSWERS} \subseteq EXPTIME^{\texttt{APPROX-RANK-ANSWER}}$.

Furthermore, we note that the exponential overhead w.r.t. the BP specification size in our top-k projection algorithm was only due to the large number of assignments to transitive edges, that had to be considered. For queries that do not project over such edges, it only remains to consider assignments to *Any*-labeled nodes; the number of such assignments is only exponential in the query size (and polynomial in the BP size). Consequently,

THEOREM 3.7. *For queries that do not project over transitive edges, Theorem 3.6 holds, with the EXPTIME complexity class replaced by PTIME with respect to the BP size (with exponential dependency over the query size).*

This will be useful when we provide, in the sequel, a PTIME algorithm for `RANK-ANSWER`; plugging-in this algorithm, we will obtain a case where `TOP-K-ANSWERS` can be solved in PTIME.

Withdrawing the assumption. Recall that we have assumed above that the projection results bear separate likelihoods, i.e. that we are given some bound on the separation in-between answers. If this does not hold then we may easily adapt our algorithm to find the top-k results up to an error of μ, for any given μ. I.e, if α is ranked by the algorithm above α', it is guaranteed that $likelihood(\alpha, q) > likelihood(\alpha', q) - \mu$.

To complete the picture, we next present algorithms for `APPROX-RANK-ANSWER`, and, where possible, for `RANK-ANSWER`.

3.3 Ranking a Single Answer

We next show that an EXPTIME approximation for the likelihood of a given sub-flow (answer). We note here that, prior work cannot be directly employed for this task: while our technique is inspired by [19], more work is required to consider the possibly recursive nested-DAG structure that we have here; using the algorithm of [4] for MSO query evaluation would result in a non-elementary combined complexity.

THEOREM 3.8. *We may solve* `APPROX-RANK-ANSWER`*, approximating the probability up to j bits of precision, in time exponential in the size of the BP specification, exponential in the size of the query, and linear in j.*

Combined with (part 2) of Theorem 3.6, we obtain:

COROLLARY 3.9. `TOP-K-ANSWERS` *may be solved in EXPTIME (w.r.t. both the BP specification size and the query size), for: (1) non-recursive BP specifications, and (2) recursive BP specifications, when the query has separated likelihoods w.r.t. the specification.*

While proving Theorem 3.8, we in fact solve here a more general problem: we consider the approximation of the likelihood of *boolean queries*. A boolean query is simply an EX-pattern; we use $likelihood^b(p, s)$ to denote the sum of all EX-flows of s in which there exists an embedding of p.

We present the approximation algorithm in two steps. We first consider a restricted case of non-recursive BP specifications, and show that an *exact* computation is possible here; then, we explain why the algorithm does not apply to the general case, and show the changes required to obtain an approximation algorithm for this setting.

The non-recursive case. Our algorithm is based on the following intuition. Recall that when a pattern p is embedded into an EX-flow e, parts of the pattern are matched

to parts of the EX-flow. Our algorithm will compute the likelihood of the full answer (w.r.t. the given BP) as an arithmetic combination of the likelihoods of its parts (w.r.t. parts of the BP).

To that end, we denote by $Parts(p)$ the set of all boolean (sub)queries obtained from an EX-pattern p by removing one or more nodes and edges, and all the conjunctions of such queries. The semantics of a conjunction is defined in a natural manner. Further recall that a query (EX-pattern) may include *simple* and *transitive* activities, where implementations of simple (transitive) pattern activities are matched to *direct (possibly indirect) implementations* of the corresponding EX-flow activities. We extend, correspondingly, our definition of $Parts(p)$: each query q appears in it in two forms: as q^{direct} and as $q^{indirect}$. Finally, we have defined above $likelihood^b(p)$ for the likelihood that an EX-flow starting from the BP *root* satisfies the EX-pattern p; we can define $likelihood^b(q^{direct}, a)$ (resp. $likelihood^b(q^{indirect}, a)$) for the likelihood that a sub-flow starting from an implementation of a satisfies q (in)directly, by extending the definition of f–likelihood (Def. 2.5) to sub-flows rooted at any compound activity a.

Given a query p and a BP s, Algorithm `EVAL-BOOL-QUERY` computes $likelihood^b(p)$ via Dynamic Programming. Observe that the non-recursive nature of the BP specification induces a *partial order* $>_s$ over its compound activities, such that $a_1 >_s a_2$ if a_2 may appear in an EX-flow originating from a_1. The algorithm first completes this partial order to a total one and processes the compound activities of s, in reversed order, from the most internal activities to the root activity. `EVAL-BOOL-QUERY` (gradually) fills in a table T of likelihoods whose rows and columns correspond to sub-queries (direct and indirect) and compound activities, resp. For each compound activity a and for all direct (resp. indirect) queries q^{direct} ($q^{indirect}$) in $Parts(p)$, the algorithm computes $likelihood^b(q^{direct}, a)$ ($likelihood^b(q^{indirect}, a)$), using the likelihoods computed for the preceding activities. The indirect likelihoods are computed only as auxiliaries, as will be explained below.

Let $\hat{p}$ denote the query pattern p with its root activity removed, and annotated as *indirect*, if p's root activity is transitive, and otherwise as *direct*. Note that with the notations introduced above, $likelihood^b(p) = likelihood^b(\hat{p}, r)$, with r being the root activity of the BP s. At the last iteration of `EVAL-BOOL-QUERY` the root activity r is reached, and (among others) $likelihood^b(\hat{p}, r)$ is computed. Then $T[\hat{p}, r]$, which contains this value, is returned.

We next explain the two functions responsible for the computation of likelihoods, namely `ComputeDirectLikelihood` and `ComputeIndirectLikelihood`.

ComputeDirectLikelihood. Given $q^{direct} \in Parts(p)$ and a compound activity a of s, `ComputeDirectLikelihood` computes $likelihood^b(q^{direct}, a)$. Recall that q has a nested-DAG shape. The "upper level" of q refers to the outer most nodes and edges of q, reachable by paths that do not include implementation edges. A matching of q^{direct} corresponds to (1) matching its upper level to some direct implementation of a, and then (2) matching the implementations of the compound activities nodes $N_1, ..., N_k$ appearing in the upper level to implementations of the corresponding BP activities (a direct/indirect match for the simple/transitive compound activities).

Ignore for now the matching of the upper level, and consider the compound activities nodes $N_1, ..., N_k$. We denote the sub-queries rooted at these nodes by $q_1, ..., q_k$. *If N_i is transitive, then q_i appears in an "indirect" form, otherwise in a "direct" form.* We then consider the matching of all sub-queries, i.e. $\bigwedge_{i=1,...,k} q_i$.

The following identity holds:

$$likelihood^b(\bigwedge_{i=1,...,k} q_i, a) = 1 - likelihood^b(\bigvee_{i=1,...,k} \neg q_i, a) \quad (1)$$

We thus focus on computing $likelihood^b(\bigvee_{i=1,...,k} \neg q_i, a)$. Using the principle of inclusion and exclusion, this term can be represented as sum of terms, all having the form $likelihood^b(\bigwedge_{i\in J} \neg q_i, a)$ for some subsets J of $\{1, ..., n\}$. Note that the exponential blow-up here is only in the size of the query, and not in that of the BP specification.

Now, re-consider the possible embeddings of the query upper level, denoted $embs(q, imp)$ for each implementation imp of a. Each such conjunction must hold (1) in the chosen implementation of a and (2) for *all* embeddings of the query within the chosen implementation. As we require that for all embeddings, all nodes are *not* matched, we may simply consider a single set of specification nodes, that contain all nodes that participated in any embedding. Moreover, the fulfillment of the negated expression is independent in-between nodes. Thus we conclude (recall that for a node n of a BP specification, $\lambda(n)$ is the activity name labeling n):

$$likelihood^b(\bigwedge_{i\in J} \neg q_i, a) = \sum_{imp\in\tau(a)} \text{c–likelihood } (imp) * \prod_{n\in emb\in embs(q,imp)} likelihood^b(\bigwedge_{i\in J} \neg q_i, \lambda(n)) \quad (2)$$

Note that now each of the expressions $likelihood^b(\bigwedge_{i\in J} \neg q_i, \lambda(n))$ satisfies $\lambda(n) <_s a$, as all of these nodes appeared in implementations of a. However, $\bigwedge_{i\in J} \neg q_i$ does not belong to $Parts(q)$, so they do not appear in T. We thus apply the following manipulation over it. First, we apply negation:

$$likelihood^b(\bigwedge_{i\in J} \neg q_i, \lambda(n)) = 1 - likelihood^b(\bigvee_{i\in J} q_i, \lambda(n))) \quad (3)$$

Now we apply again the principle of inclusion and exclusion over $\bigvee_{i\in J} q_i$ (again, only dependent on the query size) and obtain expressions of the form $likelihood^b(\bigwedge_{i\in J'} q_i, \lambda(n))$. The expressions of the sort $\bigwedge_{i\in J'} q_i$ are conjunctions of sub-queries, hence belong to $Parts(p)$, and thus the required likelihood values already appear in the Dynamic Programming table T and can be used.

ComputeIndirectLikelihood. The computation here is similar, but slightly more complicated due to the possibly indirect matches. Recall that when embedding a query indirectly, query parts may be matched to different levels of the implementation nesting. Thus, instead of dividing the query simply by its compound nodes, we define the notion of *query splits*. $\{q_1, ..., q_m\}$ is a split of $q^{indirect}$ if each q_i is a sub-graph of $q^{indirect}$, and every node or edge of $q^{indirect}$ appear in exactly one of the q_i's. We denote the set of all splits of $q^{indirect}$ by $splits(q^{indirect})$, and consider the likelihood of $\bigvee_{sp\in splits(q^{indirect})} \bigwedge_{q_i\in sp} q_i$.

By applying the principle of inclusion and exclusion, we obtain expressions of the form $\bigwedge_{sp \in SP} \bigwedge_{q_i \in sp} q_i$ for some subsets SP of *splits*. (Note that the number of splits is, once again, only a function of the query size). We can now unite the two $\bigwedge$ expressions and obtain $\bigwedge_{q_i \in sp'} q_i$ for some sp'. From this point on the computation proceeds exactly as for `ComputeDirectLikelihood` (with the only difference being that the q_i's are now not necessarily partial flows rooted at compound activities).

Complexity. The number of arithmetic operations performed by the algorithm is polynomial in the BP size, with the exponent depending on the query size. The number of bits of the computed likelihood values may become, however, exponential in the size of s. However, if we have a unit-cost RAM model with exact rational arithmetic (i.e., algebraic operations on arbitrary rationals can be done in unit time) [5], we do not have to worry about the size of the numbers. Consequently,

THEOREM 3.10. *For non-recursive BP specifications,* `RANK-ANSWER` *is in EXPTIME (combined complexity) under Turing computation model and in PTIME (data complexity, with exponential dependency on the query size) with unit-cost exact rational arithmetic.*

Combined with Theorem 3.7 and part 1 of Theorem 3.6, we obtain:

COROLLARY 3.11. *For non-recursive BP specifications, and queries that do not project over transitive edges,* `TOP-K-ANSWERS` *may be solved in PTIME (data complexity, with exponential dependency on the query size) with unit-cost rational arithmetic.*

The recursive case. To see that the previous algorithm `EVAL-BOOL-QUERY` cannot be directly applied over recursive BPs, observe that it assumed a *total order* over the BP activity names. Likelihood of queries with respect to a given activity a were computed out of previously computed likelihoods for "smaller" activities (according to the assumed order). Such order does not exist for recursive BPs. Thus, instead of using simple arithmetic, our refined algorithm generates an equations set whose solution corresponds to the query likelihood. We explain this in more details next.

Refined Algorithm. Given a BP specification s and an EX-pattern p, recall that `EVAL-BOOL-QUERY` gradually computed likelihoods for each [sub-query $q \in Parts(p)$, activity name a]. We create a variable $X_{q,a}$, whose value will reflect $likelihood^b(q, a)$, for each such pair $[q, a]$. We then choose some *arbitrary* order over the BP activities, and using this order, we follow the computation of `EVAL-BOOL-QUERY`, attempting to gradually compute likelihood values. However, in contrast to the non-recursive case, the computation of likelihood for some $[q, a]$ (i.e. computation of value for $X_{q,a}$) may require some value $likelihood^b(q', a')$ that was not computed yet (possibly $[q', a']$ is $[q, a]$ itself). To account for that, we create an equation with $X_{q,a}$ on its left-hand side. The right-hand side will contain an arithmetic expression similar to that obtained in the non-recursive case, but with $likelihood^b(q', a')$ replaced by $X_{q',a'}$, and so on. For each pair $[q, a]$, this process results in a single equation; repeating the computation for all pairs of activities names and sub-queries, the result is a set of polynomial equations.

We denote the obtained equations system by $ES[p]$ and show the following proposition.

PROPOSITION 3.12. *The solution of $ES[p]$, with all variables in $[0, 1]$, restricted to $X_{p,r}$ (r is the root activity of s), is exactly $likelihood^b(p, s)$. If more than one such solutions exist, we use the Least Fixed Point (LFP) solution.*

PROOF. To prove the proposition we need to show that a Least Fixed Point (LFP) solution exists and captures the correct likelihood values.

We first note that [15] also uses a set of equations to describe the termination probability of Recursive Markov Chains. An important property of the equations in [15] is that all the coefficients in the equations are positive. The consequent monotonicity of the polynomials is then used to prove the existence of an LFP.

In contrast, in our case, the equations may have negative coefficients (due to the use of the inclusion-exclusion principle). Thus, the proof of [15] cannot be directly applied here. Nevertheless, the system is "piece-wise" monotone, in the following sense: consider a "part-of" partial order over the sub queries q of p (including p itself). For each such q and an activity a, the computation of $likelihood^b(q, a)$ uses either likelihood values computed for queries that are "smaller" than q, or values of the form $likelihood^b(q, b)$ for some activity b (possibly $b = a$). We thus solve the equations for $likelihood^b(q, a)$ in an increasing order of such q (and for all compound activities a). Now, terms in the polynomial that correspond to queries smaller than q may be simply replaced by constants (computed in previous iterations). After substitution, the formula contains only variables for some $likelihood^b(q, b)$. It follows from the construction that these variables appear with positive coefficients. The least fixed point solution for the system may thus be computed in a bottom-up fashion (dictated by the order over query parts) using in each step the Algorithm of [15]; this least fixed point solution constitutes the correct probabilities.

□

We can further show that the following Lemma holds.

LEMMA 3.13. *Given an equations system $ES[p]$ as above, its LFP solution may be approximated up to j bits of precision, in time exponential in the number of variables in $ES[p]$ and linear in j.*

The proof follows that of Thm. 4.2 in [15] that uses the existential theory of reals [7] to approximate the LFP solution of an equations set. Combined with Proposition 3.12, this allows for an EXPTIME approximation algorithm (under both Turing computational model and the unit-cost RAM model with exact rational arithmetic), and concludes the Proof of Theorem 3.8.

4. DEPENDENCIES

Throughout the paper we assumed full independence between c–likelihood values of choices dictating the EX-flow. In practice, user choices (variables values) are often correlated. We next introduce dependencies between implementation choices, and revisit query evaluation for this settings.

We start by recalling the definition of [13] for bounded-history c–likelihood functions. To introduce dependency,

we first extend the definition of c–likelihood to consider not only a given guarding formula, but also a partial EX-flow e' representing the *history* that had occurred before the formula was evaluated. Namely, c–likelihood is now a function of both e' and f, where e' is a partial EX-flow and f is a guarding formula for some implementation of the activity that is next to be expanded in e'. For clarity of presentation, we assume a total order on the expansions order (but our results stay intact even if this is not the case).

We can now define *bounded-history* c–likelihood functions. Recall that we assumed that given an EX-flow e, the expansion sequence leading to e is well defined. The last choice preceding (the expansion of) a node n in this sequence, denoted $PrevChoice(e, n)$, is the guarding formula of the implementation selected for the last compound activity node $\hat{n}$ in this sequence that preceded n (in the above defined sense). Similarly, $PrevChoice^2(e, n)$ are the last two preceding choices, and more generally $PrevChoice^i(e, n)$ is a *vector* consisting of the i last preceding choices. We are now ready to define bounded-history c–likelihood functions.

DEFINITION 4.1. *We say that a c–likelihood function δ is bounded-history, with history bound b, if for every activity name a, every guarding formula f of a, and every two pairs of [EX-flow,next-to-be-expanded-node] $[e, n]$, $[e', n']$ where $\lambda(n) = \lambda(n') = a$ and $PrevChoice^b(e, n) = PrevChoice^b(e', n')$, it holds that c–likelihood $(e, f) =$ c–likelihood (e', f).*

Such bounded-history dependencies are common in practice, and moreover studies on the behavior of typical Web applications indicate the history size to be in practice relatively small (approximately 4) [22]. When no such bound exists, even the problem of testing, for a given EX-pattern p and a BP s, the existence of an EX-flow of s with likelihood > 0, to which p may be embedded becomes undecidable (proof by reduction from the halting problem, see [13]).

For bounded-history c–likelihood functions, we may show that one cannot hope to obtain an algorithm whose complexity is polynomial in the history-bound b, as the following theorem holds:

THEOREM 4.2. *Given a BP specification with a bounded-memory c–likelihood function,* `TOP-K-ANSWERS` *is ♯P-hard w.r.t. the history size, even for non-recursive BP specifications and queries that do not project out transitive edges.*

The proof (adapted from [13]) is by reduction from 3-SAT, showing that testing for the existence of a flow with likelihood > 0 that matches a given EX-pattern is NP-hard in b.

We next show that all our upper bounds extend to the setting of bounded-memory c–likelihood functions with bounded-history, property. The complexity now is also dependent on the history bound

THEOREM 4.3. *All of our above algorithms may be extended to consider BPs with bounded-history c–likelihood function, with an overhead that is (double-)exponential in the history size b for algorithms of PTIME (EXPTIME) data complexity.*

PROOF. We describe Algorithm `COMPILE-HISTORY-INDEPENDENT` that, given a probabilistic BP specification s with activities $a_1, \ldots, a_n$ and a bounded-history c–likelihood function, generates a new probabilistic BP specification s' with history-independent c–likelihood function, and a renaming function π mapping activities names in s' to activities names in s such that the EX-flows of s and s' are the same up to applying π, and bear the same f–likelihood value. We then explain how to apply each of the algorithms depicted in the previous section over s'.

`COMPILE-HISTORY-INDEPENDENT`. The algorithm constructs a BP s' as follows:

- Activities Names. The activities names in s' are tuples of the form $(a, pre = [pre_1^1, \ldots, pre_n^1, \ldots, pre_1^m, \ldots, pre_n^m], post = [post_1^1, \ldots, post_n^1, \ldots, post_1^m \ldots post_n^m])$ where a is an activity name, pre_j^i denotes a formula guarding the implementation chosen for a_i in its previous j expansions, prior to expanding a, and $post_j^i$ denotes a formula guaranteed to guard the implementation chosen for a_i in its next j expansions. We use $pre_j^i = \bot$ if a_i was not expanded j steps before the flow reaches a, and $post_j^i = \bot$ if a_i will not be expanded j times before the execution of a terminates.

- Guarding Formulas and Likelihood Function. Let $f_1, ..., f_l$ be the guarding formulas appearing in s, then guarding formulas in s' are of the form (f_i, pre) where pre is a vector of formulas of size b. The c–likelihood function is defined as c–likelihood $(f_i, pre) =$ c–likelihood (f_{pre}, f_i) where f_{pre} is some arbitrary partial EX-flow of s for which the next node to be expanded is guarded by f_i, and in which the last implementation were guarded by the sequence of formulas in pre (This is uniquely defined, due to the history bound of size b).

- Implementation Function. Next we construct the implementations of $(a, pre, post)$ in s'. For each implementation F_i of a in s (guarded by f_i), we create a set of new implementations, all guarded by f_i. Each implementation is obtained by annotating each activity b in F_i with vectors of pre and post conditions. The construction is as follows: (a) if r is the root of an implementation F_i of $(a, pre_a, post_a)$ guarded by f_i (note that many such activity names are created for any activity name a, differing in their pre- and post-condition vectors), then the pre-condition of r is obtained from pre_a, shifted by one step, recording F_i (and possibly deleting some formula from the vector, if reached the bound), (b) if there exists an edge from some node n to some node n' in F_i, the post-condition annotating n complies with the pre-condition annotating n', and (c) if e is the end node of F_i, the post-condition annotating it complies with $post_a$.

- Renaming Function. We define $\pi(a, pre, post) = a$ for each activity name $(a, pre, post)$.

LEMMA 4.4. *Given an EX-flow e and a renaming function π, let $\Pi(e)$ be the EX-flow obtained from e by replacing each activity name a with $\Pi(a)$. For every EX-flow $e \in flows(s)$ if and only if $\Pi(e) \in flows(s')$; additionally, f–likelihood $(e, s) =$ f–likelihood $(\Pi(e), s')$*

PROOF SKETCH. Note that every implementation of $a' = (a, pre, post)$ in s' was obtained from an implementation of a in s by replacing all activity names b in this implementation by some activity names $b, pre', post'$. Thus, by applying π over all activities in the corresponding implementation of

a' one obtains every possible implementation of a, and only such implementations. As for likelihood of flows, we show, by induction on the size of e, that f–likelihood (e, s) = f–likelihood $(\Pi(e), s')$: for the induction basis, a flow consisting only of the root activity of s (s') has a weight of 1; now, assume that f–likelihood (e_1) = f–likelihood (e'_1) for e_1 (e'_1) that bears exactly the same structure as e (e') except for its last implementation choice. Then this last implementation bears, in s, a weight of f–likelihood $(e_1)*$ c–likelihood (e_1, f) where f is its guarding formula. The corresponding implementation in s' bears a weight of f–likelihood $(e'_1)*$ c–likelihood $((f, pre))$ where pre is the pre vector encoded within the activity in e'_1 whose implementation was chosen to form e. But c–likelihood (e_1, f) = c–likelihood $((f, pre))$, because the algorithm construction of the implementation function defines a pre-condition vector that is consistent with the sequence of implementation choices made in the course of the flow (in this case e_1). □

We can now apply the top-k algorithms depicted in the previous section over s'. The algorithms need to be adapted to account for the newly created activities names of s', in the following manner: the original algorithms had matched each query node n to specification nodes having the same activity name as n. In s', the nodes activities names encode both the original activity name of the corresponding node in s, as well as some additional information on flow history. Thus, we extend the notion of an embedding: a query node with activity name a may match a specification node that encodes a, along with any history information. We then run the algorithms using this definition of an embedding instead of the original one.

Complexity. The complexity of Algorithm `COMPILE-HISTORY-INDEPNDENT` is polynomial in the size of its input BP specification s but exponential in the history bound b, as the number of pre and $post$ vectors in the algorithm construction is exponential in b. The same holds for the size of the BP specification s' outputted. Now, when applying a PTIME top-k algorithm over s', the overall complexity is polynomial in the size of s and exponential in b; when we apply an algorithm that incurs exponential time in the size of s', this translates into time exponential in the size of s but double-exponential in b.

It is open whether the double exponential dependency on b can be avoided (and reduced to a single exponent).

5. CONCLUSION

This paper studies the complexity of query evaluation for top-k projection queries over probabilistic Business Processes, where the query results are parts of possible Execution Flows that are of interest to the analyst; the query results are ranked by their likelihood of appearing in a random execution flow of the given process. We have studied the complexity of query evaluation for varying classes of queries and BP specifications, and presented restricted cases that allow for efficient query evaluation.

Future research includes the development of dedicated optimization techniques, especially in light of the high worst case complexity of query evaluation in some cases. We also intend to analyze the complexity algorithms that support richer query features such as joins and aggregates, combined with projection. Additionally, we have assumed above monotonicity of the weight function, which may not hold in some cases. We intend to consider relaxations of this assumption, and to study their effect on query evaluation.

Acknowledgments. The author thanks the anonymous reviewers of ICDT for their insightful advice.

6. REFERENCES

[1] S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active xml systems. In *Proc. of PODS*, 2008.

[2] C. Beeri, A. Eyal, S. Kamenkovich, and T. Milo. Querying business processes. In *Proc. of VLDB*, 2006.

[3] M. Benedikt, P. Godefroid, and T. Reps. Model checking of unrestricted hierarchical state machines. In *ICALP*, 2001.

[4] Michael Benedikt, Evgeny Kharlamov, Dan Olteanu, and Pierre Senellart. Probabilistic XML via Markov chains. *PVLDB*, 3(1), September 2010.

[5] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and real computation*. Springer-Verlag, 1998.

[6] Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/library/ws-bpel/.

[7] J. Canny. Some algebraic and geometric computations in pspace. In *Proc. of STOC*, 1988.

[8] S. Cohen. Generating xml structure using examples and constraints. *PVLDB*, 1(1):490–501, 2008.

[9] Sara Cohen, Benny Kimelfeld, and Yehoshua Sagiv. Running tree automata on probabilistic xml. In *PODS*, pages 227–236, 2009.

[10] B. Courcelle. The monadic second-order logic of graphs. *Inf. Comput.*, 85(1), 1990.

[11] D. Deutch and T. Milo. Type inference and type checking for queries on execution traces. In *Proc. of VLDB*, 2008.

[12] D. Deutch and T. Milo. Evaluating top-k queries over business processes (short paper). In *Proc. of ICDE*, 2009.

[13] D. Deutch and T. Milo. Top-k projection queries for probabilistic business processes. In *Proc. of ICDT*, 2009.

[14] P. Diniz. Increasing the accuracy of shape and safety analysis of pointer-based codes. In *LCPC*, 2003.

[15] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *JACM*, 56(1), 2009.

[16] J. N. Foster, T. J. Green, and V. Tannen. Annotated xml: queries and provenance. In *PODS*, 2008.

[17] M. R. Garey, R. L. Graham, and D. S. Johnson. Some np-complete geometric problems. In *Proc. of STOC*, 1976.

[18] R. Hull and J. Su. Tools for composite web services: a short overview. *SIGMOD Rec.*, 34(2), 2005.

[19] B. Kimelfeld and Y. Sagiv. Matching twigs in probabilistic xml. In *Proc. of VLDB*, 2007.

[20] Antonín Kucera, Javier Esparza, and Richard Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1), 2006.

[21] S. P. Meyn and R. L. Tweedie. *Markov Chains and Stochastic Stability*. Springer-Verlag, 1993.

[22] P. L. T. Pirolli and J. E. Pitkow. Distributions of surfers' paths through the world wide web: Empirical characterizations. *World Wide Web*, 2(1-2), 1999.

[23] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proc. of ICDE*, 2007.

[24] P. Senellart and S. Abiteboul. On the complexity of managing probabilistic xml data. In *PODS*, 2007.