Evaluating TOP-K Queries Over Business Processes

Daniel Deutch, Tova Milo
Tel Aviv University
{danielde,milo}@post.tau.ac.il

Abstract— **A Business Process (BP) consists of some business activities undertaken by one or more organizations in pursuit of some business goal. Tools for querying and analyzing BP specifications are extremely valuable for companies as they allow to optimize the BP, identify potential problems, and reduce operational costs. In particular, given a BP specification, identifying the top-k execution flows that are most likely to occur in practice out of those satisfying the query criteria, is crucial for various applications. To address this need, we introduce in this paper the notion of *likelihood* for BP execution flows, and study top-k query evaluation (finding the k most likely matches) for queries over BP specifications. We analyze the complexity of query evaluation in this context and present novel algorithms for computing top-k query results. To our knowledge, this is the first paper that studies such top-k query evaluation for BP specifications.**

I. Introduction

A Business Process (BP for short) is a collection of logically related activities that, when combined in a flow, achieve a business goal. A BP usually operates in a cross-organization, distributed environment and the software implementing it is fairly complex. To simplify software development, it is a common practice to provide a high level description of the BP operational flow (using a standard specification language such as BPEL [1]), and then have the software be automatically generated from this specification. Since the BP logic is captured by the specification, tools for querying and analyzing BP specifications are extremely valuable for companies [2]. They allow to optimize the BP, identify potential problems, and reduce operational costs.

As a simple example, consider a BP of an on-line, Web-based travel agency. An analyzer that wishes to examine the BP execution flows may issue queries such as "At which points of the flow is the user asked to relay her credit card details?", "In what ways may one reserve a travel package containing a flight and an hotel?", or "How is this done for travelers of a particular airline company, say British Airways?", etc.

A typical query engine is given as input the BP specification and an execution pattern of interest, and identifies, among the potential execution flows of the BP, all those having the structure specified by the pattern ([2], [3]).

Note, however, that the total number of qualifying execution flows may be very large, or even infinite in presence of recursion. Among these qualifying flows, some are typically more "interesting" than others. In particular, given a BP specification, identifying the top-k execution 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, to personalize on-line advertisements, or to provide appealing package deals.

For instance, say that we obtain that a popular execution flow containing a British Airways reservation is one where users first search for a package containing both flights and hotels, but eventually book a British Airways flight without reserving an hotel. Such result may strongly 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.

To address the need for such top-k analysis of BP executions, we introduce in this paper the notion of *likelihood* for BP execution flows, and study top-k (most likely matches) query evaluation for queries over BP specifications. We analyze the complexity of query evaluation in this context and present efficient algorithms for top-k query computation. To the best of our knowledge, this is the first paper that studies such top-k query evaluation over BP specifications.

Our contribution here is twofold. First, we present a simple generic probabilistic model that allows to describe the possible execution flows of a given BP, and their likelihoods, in presence of different sorts of dependencies, between events dictating the course of execution. We distinguish several classes of likelihood functions over execution flows of the process, according to the level of dependencies in-between events dictating the execution course (user choices, server states, etc.). Our model extend the model of [2], [3] to a probabilistic context. The model there is an abstraction of the BPEL (Business Process Execution Language [1]) standard.

Next, we consider queries. The query language that we consider selects execution flows of interest, using execution patterns [2]. We study the problem of identifying, for a given BP and a query, the top-k execution flows with highest likelihood out of these satisfying the query. We analyze the complexity of query evaluation for various classes of likelihood functions, and show that for a practically common class, the problem can be efficiently solved. We focus in particular on a practically common class of likelihood functions, namely functions that are of *bounded-memory*. Intuitively, with such function, the likelihood of a choice depends only on a bounded number of previously made choices. *The second contribution of this paper is a novel algorithm that computes the top-k execution flows conforming to a query, in presence of a bounded-memory likelihood function.*

This is a short version of our paper, summarizing the main contributions. The reader is referred to [4] for a full version containing exact details, along with an experimental study.

Related Work: We give next a brief review of related work, and refer the reader to [4] for further references.

Probabilistic Databases (PDBs) [5], [6] and Probabilistic Relational Models (PRMs) [7] allow representation of uncertain information, but consider relational data and do not capture the dynamic nature of flow and the possibly unbounded number of recursive (possibly dependent) invocations. In terms of the *possible worlds semantics*, the number of worlds in our model is infinite (rather than large, yet finite, in PDBs). Extensions of PRMs to a dynamic setting, called Dynamic PRMs [8], do not allow for practically efficient algorithms.

Probabilistic XML [9], [10] bears some resemblance to our model: the data is graph (tree) shaped, and it allows some dependencies between probabilistic events. However, our model is more complex: first, it represents nested DAG structures, rather than trees, entailing more intricate dependencies between events. Second, potentially infinite number of such nested DAGs are represented, due to possible recursive calls.

We assume that the BP and a description of the likelihood function are readily given. The design/inference of both has been the focus of several previous works (see e.g. [11], [12]), and is outside the scope of this work.

Paper organization: In Section II we informally define our model and the notion of top-k execution flows. In section III we present our algorithms for finding the top-k execution flows conforming to a query, and we conclude in section IV.

II. Preliminaries

We start by reviewing (informally) the main concepts discussed in the paper.

Business Processes: A Business Process (BP) is an abstraction of the BPEL standard for specification of processes. We show its merits via an example, as follows.

Example 2.1: The business logic of a Web-based travel agency is given as a BP in Figure 1. A BP describes a process as a nested DAG (Directed Acyclic Graph) consisting of activities (nodes), and links (edges) between them. Links detail the execution order of the activities. Each activity is represented by a pair of *nodes*, the first (having darker background) standing as the activity's *activation point* and the second as its *completion point*. *Edges* represent *execution flow* relations; multiple edges going out of a single node stand for *parallelism* (hence the DAG structure). Activities may be either *atomic* (like the $Login$ activity) or *compound* (like $start$, $chooseTravel$ and $Flights$). In the latter case, each of their possible internal structures (called *implementation*) is also detailed as a DAG (depicted as a "bubble" and leading to the nested structure). For instance, at the $start$ activity, the user may choose between three possible ways of usage. By setting $\$usage$ to be "search travel", a $chooseTravel$ activity is invoked; it has three possible implementations $F3$, $F4$, $F5$. Each implementation is guarded by a *guarding formula*, in this case testing the value of the $\$searchType$ variable. This value may either be "flights only", "flights+hotels", or "flights+hotels+cars", depending on the user's choice. At run-time, exactly one implementation will be chosen, determined

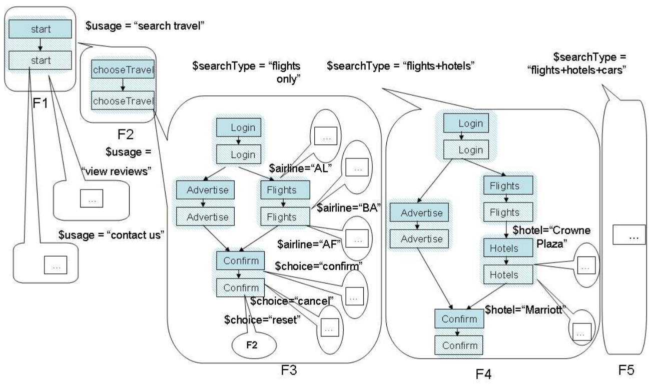

Fig. 1. Business Process

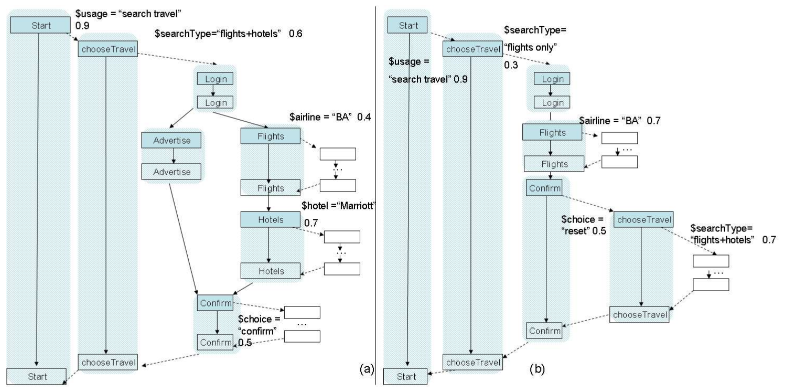

Fig. 2. Ranking Implementation Choices

by the truth value of the guarding formulas (determined, in turn, by the user choice). Focusing on the $F3$ implementation, the $Flights$ activity has a set of possible implementations, corresponding to choices of $\$airline$ ("BA" stands for British airways, "AF" for Air France, "AL" for Aer Lingus). Last, the $Confirm$ activity allows the option of *reset*, recursively going back to $F2$, or alternatively to confirm or cancel.

Execution Flows: An execution flow is an actual running instance of a Business Process. It may be abstractly viewed as a nested DAG, containing node-pairs that represent the activation and completion of activities, and edges that represent flow and zoom-in (implementation) relationships among activities, along with a record of guarding formulas corresponding to chosen implementations of compound activity nodes. Figure 2(a) depicts an example execution flow of the travel agency process, in which the user chose to search for travel, then chose a "flights+hotels" search, and finally made a reservation consisting of (ignore, for now, the numbers annotating the different choices). Zoom-in edges (denoted by dashed arrows) connect activation and completion nodes of compound activities to the start and end nodes of the chosen implementation.

Likelihood: We define two sorts of likelihood functions: (1) the c–likelihood function, that determines the likelihood of the different *implementation choices*, and (2) the f–likelihood function, that determines the likelihood of flows. Recall that each execution flow corresponds to a unique sequence of implementation choices taken throughout the execution, and we define the f–likelihood of a flow as the multiplication of c–likelihood values along the execution.

A c–likelihood function receives as input both a guarding formula and a partial flow representing the "history", i.e. flow thus far. Intuitively, a c–likelihood function is simple if it's "local", i.e. it is in fact a function of the guarding formula solely, or a function of the formula along with a small amount of history that affect its value. To capture this notion, we define three classes of functions, with decreasing level of simplicity.

Memory-less functions: The simplest class of c–likelihood functions is the *memory-less* class. δ is *memory-less* if for each formula f and for each two partial flows (histories) e, e', $\delta(f, e) = \delta(f, e')$. This means that choices are all independent.

Example 2.2: An example for such independent likelihoods annotates the choices in Figure 2(a), and the corresponding f–likelihood of the flow is computed as their multiplications, that is $0.9 * 0.3 * 0.7 * 0.5 * 0.7 = 0.06615$.

Bounded memory functions: A more general and more realistic class of c–likelihood functions captures the common case where the c–likelihood of any given choice may depend on the execution history, but only in a bounded manner. Intuitively, the bound b is on the maximal number of past choices, made for any activity, that affect the likelihood of some choice. That is, at any point of the execution, and given the implementation choices taken for the last b occurrences of each activity, the likelihood of every implementation choice is indifferent to choices made further in the past.

Example 2.3: Consider for example the execution flow depicted in Figure 2(b). Note that the likelihood of the different search types depends upon the choices preceding it: at first, the likelihood of a "flights only" search type is 0.8, but given that the user was unsatisfied with the results and chose to reset, the likelihood of him making the same choice again decreases to 0.3. For a given flow, however, the entire history is known at each point. Thus computation of f–likelihood values is again obtained as a multiplications of the c–likelihood values along the flow, that is $0.9 * 0.6 * 0.4 * 0.7 * 0.5 = 0.0756$.

Unbounded-memory functions: In general, there exist c–likelihood functions that do not fall under the bounded-memory category. We refer to these as *unbounded-memory*.

Example 2.4: Consider a scenario where the likelihood of each hotel choice depends on the exact number of "resets" previously chosen. Here, an unbounded number of choices for the same activity name must be known for likelihood computation. In practice, such scenario is rare (see e.g. [13]).

Queries: Queries are defined using *execution patterns*, whose structure is similar to that of execution flow. A subset of the pattern's edges may be marked as transitive, seeking for a path connecting the edge end-nodes, rather than a single edge; similarly, composite nodes may be marked as *transitive*, seeking for *possibly indirect* implementations.

To evaluate a query over a BP specification s, we search for occurrences of the execution pattern in execution flows of s, represented by *embeddings*. An embedding is a homomorphism from all nodes and edges of the pattern to some nodes and edges of the flow, such that for matched nodes, their activity names coincide, and composite (atomic) nodes are mapped to composite (atomic) nodes. Non-transitive edges (both regular and zoom-in) of the query are mapped to corresponding edges of the trace, such that the end-points in the pattern are mapped to the corresponding end-points in the flow. Transitive edges may be mapped to paths (containing flow or zoom-in edges), and implementations of transitive query nodes may be mapped to indirect implementations of the corresponding flow nodes.

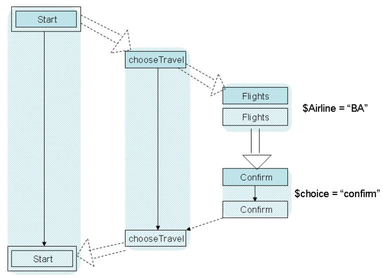

Fig. 3. Query

Example 2.5: An example execution pattern is depicted in Fig. 3. It zooms-in transitively into `Start`, searching for a `chooseTravel` activity (that may follow any number of resets). Then, it requires a reservation of a BritishAirways flight, followed by a confirmation.

Note that double-bounded nodes (double-lined edges) denote transitive nodes (resp. edges).

TOP-K results: Given a BP specification s, a query q and a number k, the top-k results of q with respect to s (denoted $top\!-\!k(q, s)$) are defined as the k most likely flows of s, out of those in which an embedding of q exists.

III. Query Evaluation

We now turn to describing our results and algorithms for query evaluation over BP specifications.

A. General Framework

Given a BP s, a c–likelihood function δ, and a query q, the top-k query results are computed in two steps.

1) First, we construct a BP s' and a c–likelihood function δ' such that the flows set of s' is identical, up to activities renaming, to the subset of s flows that match q, and have correspondingly identical likelihoods.
2) Next, depending on the type of δ (memory-less, bounded-memory), we use s' and apply a particular algorithm that generates a specification s'' whose set of flows corresponds exactly to the top-k query results.

The first step is an adaptation of the evaluation algorithm given in [14]. We thus omit its details for brevity, and focus on the second step, namely finding the top-k flows of a given specification, with respect to a given likelihood function, of each of the different classes.

B. Memory-less c–likelihood functions

For the case of memory-less c–likelihood functions, we are able to give an efficient algorithm for computing a compact representation of the top-k qualifying flows.

Theorem 3.1: Given a BP s, a memory-less c–likelihood function, and a query q, we may compute a compact representation of $top\!-\!k(q, s)$, in polynomial time (data complexity).

Proof: [Sketch]

The algorithm is based on the following lemma:

Lemma 3.2: For every memory-less c–likelihood function δ, a BP s and a compound activity a in s,

1) There exists a best ranked (top-1) EX-flow originating at a that does not contain another occurrence of a.
2) There exists a $j+1$'th ranked EX-flow originating at a such that for any occurrence of a in it, the sub-flow rooted at the latter is one of the top-j EX-flows of a.

The above lemma implies that when searching for the top-k EX-flows, it suffices to examine a *finite number* of EX-flows. Thus, a simple algorithm for finding the top-k EX-flows is to enumerate all these EX-flows, compute their f–likelihood , and pick out the top-k ones. Note that the number of EX-flows examined by this naive algorithm may be *exponential* in the size of the BP s. To avoid examining all of them we use a Dynamic Programming approach, that materializes in each step only the essential front line of flows. ■

We can also show that, unless $P = NP$, no algorithm polynomial in the query size is possible for computation of top-k results. First, we define the BEST-MATCH decision problem: given a BP specification, a c–likelihood function, a query, and a bound B, decide the existence of a qualifying flow whose f–likelihood is greater than B.

Theorem 3.3: BEST-MATCH is NP-complete w.r.t. the query size, even for memory-less c–likelihood functions.

C. Bounded-memory functions

Bounded-memory c–likelihood functions pose further challenges, as the c–likelihood of each choice may depend on a number of other choices. We may show the following theorem:

Theorem 3.4: Given a BP s', a bounded-memory c–likelihood function δ', and a query q, we may compute a compact representation of the top-k results, in EXPTIME.

Proof: [sketch] The general idea of the algorithm is to create, given s' and δ' with a memory bounded by m, a new BP s'', with a new, *memory-less*, c–likelihood function δ'', such that s' and s'' have essentially the same set of flows with the same f–likelihood . Then, we apply the algorithm from the proof of Theorem 3.1. To create this memory-less function, we annotate the activity names in s', "factoring" within the names all information required for the computation of c–likelihood of formulas, namely a pre-condition vector, including the m last choices for all activities. Additionally, the new activities names also contain post-condition vectors, necessary to assure consistencies between pre-conditions assumed by activities and what has happened in their predecessors. The algorithm is polynomial in k and exponential in $|s'| \times m$. ■

Unfortunately, no polynomial algorithm is likely to exist in this setting, as the following theorem holds.

Theorem 3.5: Given a BP s with a bounded-memory c–likelihood function δ and a query q BEST-MATCH is NP-complete w.r.t. the size of s.

The theorem is proved by a reduction from probability computation over bayesian networks, known to be NP-hard.

However, the pathological scenarios that lead to this NP-hardness are not necessarily typical. We may optimize our EXPTIME algorithm, by identifying common cases where more efficient processing is possible, namely by exploiting (conditional) independencies between implementation choices. Our experiments (omitted for lack of space) show that the optimizations succeed in achieving feasible execution times even for large-scaled specification and queries.

D. Unbounded-memory c–likelihood functions

Last, we may show that for general c–likelihood functions that do not adhere to any of the previously discussed classes, computation of the top-k results is impossible.

Theorem 3.6: The BEST-MATCH problem is undecidable for unbounded-memory c–likelihood functions.

The proof (omitted here) is by reduction from the halting problem of a Turing Machine, known to be undecidable.

IV. Conclusion

This paper studies, for the first time, the problem of top-k query evaluation over BPs. We have studied the complexity of the problem for various classes of likelihood functions over the execution course, and presented efficient algorithms for query evaluation, where possible, for each class. Important applications of our results include adjusting web-sites design to the needs of certain user groups, personalization of on-line advertisements, focusing on particular target audience, and enhancing business logic. We intend to study such applications as future research. The development of dedicated query optimization techniques is another intriguing future challenge.

References

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

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

[3] C. Beeri, A. Eyal, T. Milo, and A. Pilberg, "Monitoring business processes with queries," in *Proc. of VLDB*, 2007.

[4] D. Deutch and T. Milo, "Evaluating top-k queries over business processes (extended version)," http://www.cs.tau.ac.il/~danielde/icde09Extended.pdf.

[5] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," in *Proc. of VLDB*, 2004.

[6] P. Sen and A. Deshpande, "Representing and querying correlated tuples in probabilistic databases," in *ICDE*, 2007.

[7] N. Friedman, L. Getoor, D. Koller, and A.Pfeffer, "Learning probabilistic relational models," in *Proc. of IJCAI*, 1999.

[8] S. Sanghai, P. Domingos, and D. Weld, "Dynamic probabilistic relational models," in *Proc. of IJCAI*, 2003.

[9] S. Abiteboul and P. Senellart, "Querying and updating probabilistic information in xml," in *Proc. of EDBT*, 2006.

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

[11] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. on Knowl. and Data Eng.*, vol. 16, no. 9, 2004.

[12] R. Silva, J. Zhang, and J. G. Shanahan, "Probabilistic workflow mining," in *KDD*, 2005.

[13] P. L. T. Pirolli and J. E. Pitkow, "Distributions of surfers' paths through the world wide web: Empirical characterizations," *World Wide Web*, vol. 2, no. 1-2, 1999.

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