On Factorisation of Provenance Polynomials

Dan Olteanu and Jakub Závodný
Oxford University Computing Laboratory
Wolfson Building, Parks Road, OX1 3QD, Oxford, UK

1 Introduction

Tracking and managing provenance information in databases has applications in incomplete information and probabilistic databases, query evaluation under bag semantics, view maintenance and update, debugging and explanation, and annotation propagation [2]. A recurring observation is that provenance information tends to grow very large with the number of data operations: For instance, many records in the Gene Ontology public database have each over 10MB of provenance data [12]. Practical and theoretical tools are thus necessary to pave the way to efficient management of such large amounts of provenance data. A second observation is that the propagation of provenance information via queries in databases follows a rather regular pattern, which can be exploited for query-aware provenance compression.

These two observations are at the outset of our recently started project, whose goal is to provide a theoretical and practical framework for managing databases annotated with provenance information in the form of *provenance polynomials* [7]. Provenance polynomials represent a unifying formalism for much of recent work in the aforementioned areas. They capture as particular instances known types of provenance information ranging from data warehousing lineage, in which a result tuple is annotated with a set of identifiers of all contributing input tuples, to why-provenance, in which result tuples are annotated with a set of sets of contributing input tuples.

Provenance polynomials generated by query evaluation in relational databases can have a regular structure that can be exploited for a more succinct representation via algebraic factorisations [11]. The novelty of this project lies in the investigation and use of such factorised representations of provenance polynomials. In this paper we highlight key properties and potential benefits of factorised provenance polynomials. We also present a list of challenges and outline results obtained so far in managing factorised polynomials of query results.

2 Provenance Polynomials

We next exemplify the notions of provenance polynomials and their algebraic factorisations. Consider an office-supply warehouse database, showing current orders, stocks and employee availability. Each tuple in the input database is annotated with an indeterminate, or variable, which encodes here its provenance information.

`Order`

	id	item
o_1	01	Printer
o_2	02	Plotter
o_3	03	Ink
o_4	04	Printer
o_5	05	Ink

`Store`

	location	item
s_1	Depot1	Printer
s_2	Depot1	Plotter
s_3	Depot2	Printer
s_4	StoreA	Ink

`Emp`

	operator	location
e_1	Joe	Depot1
e_2	Bob	Depot1
e_3	Dan	Depot2
e_4	Dan	StoreA

In this scenario, data is aggregated from different extraneous or uncertain source databases: Orders could come from different vendors, storage information from different stock lists at each location, and employee availability could be produced by real-time tracking. When deciding on which employee should deliver what order from which location, the source of information supporting the decision, that is, its provenance, is essential, and should ideally be recorded for later reference.

Consider a query that finds all orders with their respective items, their possible locations and workers available to access and retrieve them:

Order $\bowtie_{\text{item}}$ Store $\bowtie_{\text{location}}$ Emp

	id	item	location	operator
$o_1s_1e_1$	01	Printer	Depot1	Joe
$o_1s_1e_2$	01	Printer	Depot1	Bob
$o_1s_3e_3$	01	Printer	Depot2	Dan
$o_2s_2e_1$	02	Plotter	Depot1	Joe
		...		

Each tuple in the result table arises as a combination of input tuples, and its provenance is the product of the variables of its contributing input tuples. For example, the tuples annotated with any of o_1, s_1 or e_1 contribute to the result tuple annotated with $o_1s_1e_1$. The sum of the products for all result tuples forms the provenance polynomial of the whole query result:

$$\begin{aligned}\Phi_1 = {} & o_1s_1e_1 + o_1s_1e_2 + o_1s_3e_3 + o_2s_2e_1 + o_2s_2e_2 + \\ & + o_3s_4e_4 + o_4s_1e_1 + o_4s_1e_2 + o_4s_3e_3 + o_5s_4e_4.\end{aligned}$$

The abstract product and sum operations on variables represent multiplication and summation in various commutative semirings. In the Boolean semiring $(\mathbb{B}, \vee, \wedge)$, we interpret the variables as Booleans and the product and sum as logical "and" $(\wedge)$ and "or" $(\vee)$ respectively. The Boolean semiring captures the semantics of positive relational algebra queries under set semantics: In the previous example, if the variables o_1, s_1, and e_1 are set to true, then the input tuples they annotate are in the database and the tuple annotated with $o_1s_1e_1$ is in the query result. The Boolean semiring is also used in the context of incomplete and probabilistic databases [7]. The semiring over natural numbers $(\mathbb{N}, +, \cdot)$ captures the semantics of positive queries under bag semantics: The variables are interpreted as tuple multiplicities. Individual monomials then give the multiplicities of result tuples and the whole polynomial yields the cardinality of the query result.

In the following sections, we consider provenance polynomials of the *entire* result of a query. This is without loss of generality, since the analysis for this case can be extended in a standard way to the case of provenance polynomials for individual result tuples.

3 Factorisation of Provenance Polynomials

A useful property of provenance polynomials is the distributivity of product over sum. This enables the polynomial Φ_1 to be factorised into a nested expression:

$$\begin{aligned}\Phi_2 = {} & (o_1 + o_4)(s_1(e_1 + e_2) + s_3e_3) + o_2s_2(e_1 + e_2) \\ & + (o_3 + o_5)s_4e_4.\end{aligned}$$

The factorised form Φ_2 is equivalent to Φ_1 in the sense that it has the same monomials as Φ_1.

When compared to flat polynomials such as Φ_1, factorised polynomials can be both more informative and more succinct. Algebraic factorisations of provenance polynomials allow us to make explicit the structure in the query result. For instance, Φ_2 shows more directly than Φ_1 that orders annotated by o_1 and o_4 are joined with the same tuples from tables `Store` and `Emp`. Moreover, factorised polynomials of query results can be exponentially more succinct than their flat equivalents, with the exponent being the query size. Consider the query expressing a product of n relations $R_1 \times \cdots \times R_n$. The flat polynomial of the result enumerates explicitly all combinations of tuples from the input relations, and its size is the product of the sizes of the input relations. An equivalent factorised polynomial is the product of n sums $S_1 \cdots S_n$, where sum S_i is over all variables of relation R_i, and whose size is the sum of the sizes of the input relations. Moreover, this factorisation can be computed efficiently and directly from the input database. This means that in addition to space savings, we can also reduce the time complexity for computing the provenance polynomial by an exponential factor.

A downside of factorisations is that the monomials of a factorised polynomial are not any longer represented explicitly, and accessing them require more work than in the case of flat polynomials. We can nevertheless enumerate them with polynomial delay. That is, the time needed to output the first monomial (in some order) as well as the delay between the output of two subsequent monomials is polynomial. In particular, the delay and the space requirements for the enumeration are $O(S \log S)$, where S is the size of the factorised representation [11].

4 From Factorised Polynomials to Factorised Representations of Relations

Algebraic factorisations can be straightforwardly extended from provenance polynomials to entire query results and even to arbitrary relations by inlining tuples next to their variables. This extended factorisation forms a complete representation system for relational data with provenance. In particular, we obtain a succinct and efficiently computable representation of the query result and its provenance.

For instance, the polynomial Φ_2 can be extended to the following representation

$$\begin{aligned}& (o_1\langle \text{01,Printer}\rangle + o_4\langle \text{04,Printer}\rangle) \\ & \quad (s_1\langle \text{Depot1,Printer}\rangle(e_1\langle \text{Joe,Depot1}\rangle + e_2\langle \text{Bob,Depot1}\rangle) + \\ & \quad s_3\langle \text{Depot2,Printer}\rangle e_3\langle \text{Dan,Depot2}\rangle) + \\ & o_2\langle \text{02,Plotter}\rangle s_2\langle \text{Depot1,Plotter}\rangle \\ & \quad (e_1\langle \text{Joe,Depot1}\rangle + e_2\langle \text{Bob,Depot1}\rangle) + \\ & (o_3\langle \text{03,Ink}\rangle + o_5\langle \text{05,Ink}\rangle)s_4\langle \text{StoreA,Ink}\rangle e_4\langle \text{Dan,StoreA}\rangle.\end{aligned}$$

To correctly interpret this representation, we would need a mapping between the variables annotating the tuples, and the schemas of these tuples in the input database.

Some attributes appear several times in the monomials of the above representation. For instance, the attribute `item` occurs with variables from both `Order` and `Store`. As in the relational case, we keep one of them in case of a natural join between `Order` and `Store` and project out the other one. One possible representation after dropping duplicate attributes is shown next:

$$\begin{aligned}
&(o_1\langle\text{01,Printer}\rangle + o_4\langle\text{04,Printer}\rangle)\\
&\quad(s_1\langle\text{Depot1}\rangle(e_1\langle\text{Joe}\rangle + e_2\langle\text{Bob}\rangle) + s_3\langle\text{Depot2}\rangle e_3\langle\text{Dan}\rangle)+\\
&o_2\langle\text{02,Plotter}\rangle s_2\langle\text{Depot1}\rangle(e_1\langle\text{Joe}\rangle + e_2\langle\text{Bob}\rangle)+\\
&(o_3\langle\text{03,Ink}\rangle + o_5\langle\text{05,Ink}\rangle)s_4\langle\text{StoreA}\rangle e_4\langle\text{Dan}\rangle.
\end{aligned}$$

In addition to the relational schema that captures the attributes of the tuples in the representation, we would like to capture the structure of the representation. A *factorised schema* defines both the relational schema of the tuples and the nesting structure of the factorisation. For our example above, the factorised schema would be `((id, item), (location, (operator)))`. This reads as follows: There can be several `(id, item)` values that are paired with several locations, and for each location there can be several operators. Factorised representations of query results always have a factorised schema, which can be inferred from the query. In particular, such schemas correspond to join orders in the query plans used to evaluate the query.

5 Challenges

Managing factorised representations poses new theoretical and practical challenges; we outline here a few of these challenges and focus mostly on provenance polynomials, although these challenges also apply to factorised representations of relations.

1. Characterisation of queries with bounded-size factorised provenance polynomials. Of paramount importance to the feasibility of our approach is to quantify how factorisable the (provenance polynomials of) query results are as a function of both input database and query. In particular, we would like a classification of relational queries based on the minimal possible size of the factorised provenance polynomials for any input database.

Consider the query `Store` $\bowtie_{\mathtt{location}}$ `Emp` on our example database. For each location l, all tuples in `Store` reporting items at location l join with tuples in `Emp` reporting employees serving location l. The polynomial for l can be expressed as a product of two sums, one sum for the items at location l and one sum for the operators at location l. To obtain the polynomial of the query result, we sum the polynomials for all locations:

$$\Phi_3 = (s_1+s_2)(e_1+e_2)+s_3e_3+s_4e_4.$$

A key observation is that the polynomial of `Store` $\bowtie_{\mathtt{location}}$ `Emp` can be factorised into a sum of products of sums for any input database and that the resulting factorisation is *read-once*, that is, it contains at most one occurrence of each variable. Its size is therefore at most linear in the size of the database. As part of the characterisation, it would be desirable to identify all queries whose results admit read-once factorisations. In contrast to the above query, the query from the introduction can have polynomials that do not have read-once factorisations: This is already the case for the polynomial Φ_1 obtained for the example database. For the original query, it turns out that the minimal possible number of occurrences of some variables in any factorised form depends on the input database and is therefore unbounded.

Specific semirings may introduce further equivalences between polynomials and hence more compact forms than those allowed by algebraic factorisations alone. In the Boolean semiring, for example, both multiplication and addition are idempotent, and we can use the identities $x \cdot x = x + x = x$ to further decrease the size of the polynomials. The query `Order` $\bowtie_{\mathtt{id}\leq\mathtt{id'}}$ $\delta_{\mathtt{id}\rightarrow\mathtt{id'}}$`Order` yields the polynomial

$$\begin{aligned}
\Phi_4 = {}& o_1o_1+o_1o_2+o_1o_3+o_1o_4+o_1o_5+o_2o_2+o_2o_3+\\
&+o_2o_4+o_2o_5+o_3o_3+o_3o_4+o_3o_5+o_4o_4+o_4o_5+o_5o_5.
\end{aligned}$$

We can factorise Φ_4 algebraically into

$$\begin{aligned}
\Phi_5 = {}& o_1(o_1+o_2+o_3+o_4+o_5)+o_2(o_2+o_3+o_4+o_5)+\\
&+o_3(o_3+o_4+o_5)+o_4(o_4+o_5)+o_5o_5,
\end{aligned}$$

but when interpreted in the Boolean semiring, it is also equivalent to the much simpler expression

$$\Phi_6 = o_1+o_2+o_3+o_4+o_5.$$

2. Efficient computation of factorised polynomials. Given a polynomial, how can we compute an optimal factorisation, that is, a factorisation with smallest possible size? This is computationally difficult. In the setting of Boolean factorisation (as opposed to algebraic factorisation considered here), [3] shows that for $k > 2$, it is NP-hard to decide for a given flat polynomial whether there exists a factorisation using at most k occurrences of each variable. The problem is open for $k = 2$.

In our setting we are specifically concerned with the problem of finding factorisations for polynomials of query results. On one hand, this restricts the class of polynomials under consideration. On the other hand, we would like to compute an optimal factorisation of the provenance polynomial directly from the query and the input database, that is, without first computing the flat polynomial and then factorising it.

The optimality requirement may be relaxed for practical reasons. Also, it may be desirable to trade the optimality for a simpler nesting structure of the factorisation. Although all factorisations of a given polynomial are equivalent, particular nesting structures may facilitate more sophisticated manipulation, better storage, or faster enumeration of monomials in a desired order.

For example, consider the two equivalent polynomials

$$\Phi_7 = (s_1+s_2)(e_3+e_4)+(s_3+s_4)(e_1+e_2)+ \\ +s_1e_2+s_2e_1+s_3e_4,$$

$$\Phi_8 = s_1(e_2+e_3+e_4)+s_2(e_1+e_3+e_4)+ \\ +s_3(e_1+e_2+e_4)+s_4(e_1+e_2).$$

Assume here that the polynomials are enriched with the input tuples from our example database. Although Φ_7 has smaller size (that is, less variable occurrences) than Φ_8, the latter has a structure that allows a quick enumeration of tuples with variables s_1 to s_4; in our database, that would be tuples from relation Store. Also, a join on location or item values, as provided by these tuples, is better supported by Φ_8.

3. Queries over factorised representations of relations and over factorised provenance polynomials. To what extent is it possible to use factorisations of provenance polynomials, or even of entire query results, for further processing? The ability to evaluate queries on factorised forms would greatly amplify the advantage of their succinctness and low time complexity, turning them into a truly advantageous representation system for relational data and its provenance.

In particular, this challenge is to identify classes of queries that can be evaluated on factorised representations without unfolding them into a flat form and with minimal re-structuring. For illustration, consider a scenario in which the result of our query $Q = \texttt{Order} \bowtie_{\texttt{item}} \texttt{Store} \bowtie_{\texttt{location}} \texttt{Emp}$ has the provenance polynomial

$$\Phi_9 = (o_1+o_2)(s_1(e_1+e_2)+s_2(e_3+e_4))+ \\ +(o_3+o_4)(s_3(e_1+e_2)+s_4(e_3+e_4)).$$

An equivalent factorisation is

$$\Phi_{10} = ((o_1+o_2)s_1+(o_3+o_4)s_3)(e_1+e_2)+ \\ +((o_1+o_2)s_2+(o_3+o_4)s_4)(e_3+e_4).$$

The expressions Φ_9 and Φ_{10} have equal sizes but different structures. Suppose that the result of Q is to be joined with a relation `Shipping` on order IDs. Then Φ_9 is more suitable to carry out this join, since each order tuple, which occurs with one of the variables o_1 to o_4, appears only once, whereas the order tuples appear multiple times in Φ_{10}. For a further example, consider that each relation `Store` and `Emp` in our example database has an additional attribute `time`, which records the time an item and respectively an operator are at a location. We would like to further refine the query Q by adding the equality condition on time: `Store.time = Emp.time`. This constraint can lead to dropping some combinations of tuples from `Store` and `Emp` that already exist in Q's result. In the polynomials Φ_9 and Φ_{10}, this means that some combinations of variables s_i and e_j are dropped. While this can be performed locally in case of Φ_9, it can require re-structuring in case of Φ_{10}.

4. Approximation of provenance polynomials by more succinct factorised polynomials. For query results, whose polynomials Φ are poorly factorisable, we would like to find factorised polynomials Φ_L and Φ_U that represent lower and upper bound approximations of Φ in a given provenance semiring. For instance, if Φ serves as an explanation of a query result, then the lower bound Φ_L represents a necessary, but possibly not sufficient explanation of the query result, whereas the upper bound Φ_U represents a sufficient, but possibly not necessary explanation of the query result.

We would like the bounds Φ_L and Φ_U to have better properties than Φ, such as being of size linear in the input or having read-k factorisations, i.e., factorisations where each variable occurs at most k times. Also, the bounds should be optimal in the sense that there are no other bounds that are "closer" to Φ and have the same good properties. The definitions of "closer," as well as of upper and lower bounds, depends on the provenance semiring. It nevertheless holds for any semiring that lower and upper bounds can be obtained by removing and respectively adding monomials to the polynomial Φ.

We illustrate this with our example query and database. We have seen that the polynomial Φ_1 does not have a read-once factorisation. However, by removing two monomials from Φ_1 (corresponding to two tuples in Q's result), we can factorise the polynomial into

$$\Phi_L = (o_1+o_4)(s_1(e_1+e_2)+s_3e_3)+(o_3+o_5)s_4e_4.$$

By adding seven new monomials to Φ_1 (corresponding to seven new tuples in Q's result), we obtain

$$\Phi_U = (o_1+o_2+o_4)((s_1+s_2)(e_1+e_2)+s_3e_3)+ \\ +(o_3+o_5)s_4e_4.$$

The polynomials Φ_L and Φ_U represent lower and upper bounds of Φ_1 respectively. They are optimal with respect to the class of read-once polynomial factorisations.

The interpretation and optimality of bounds change in case of specific semirings. For the Boolean semiring, an optimal lower bound polynomial for Φ in a given class of polynomials $\mathscr{P}$ would be a polynomial $\Phi_L \in \mathscr{P}$ such that the set of satisfying assignments of Φ_L is included in that of Φ and there is no other polynomial $\Phi'_L \in \mathscr{P}$ such that its set of satisfying assignments strictly includes that of Φ_L and is included in that of Φ. The case of upper bounds is analogous. As an example, consider the polynomial

$$\Phi_{11} = s_1e_1+s_1e_2+s_2e_1.$$

By adding one monomial to Φ_{11}, we obtain an optimal upper bound that is a read-once factorisation:

$$\Phi'_U = (s_1+s_2)(e_1+e_2) = s_1e_1+s_1e_2+s_2e_1+s_2e_2.$$

However, the polynomial

$$\Phi''_U = s_1+s_2e_1$$

is also an optimal upper bound of Φ_{11}, is a read-once factorisation, and is incomparable with Φ'_U. The polynomial Φ''_U is nevertheless no valid upper bound in case of the semiring over natural numbers: Indeed, if the variables are replaced by natural numbers, then $s_1e_1+s_1e_2+s_2e_1 \geq s_1+s_2e_1$.

This challenge can be shifted from approximation by polynomials to approximation by queries: Given a query Q whose polynomial Φ is poorly factorisable, find lower and upper bound queries Q_L and Q_U with polynomials Φ_L and Φ_U that are better factorisable, such that Φ_L and Φ_U are lower and upper bounds, respectively, of Φ in a given provenance semiring.

5. Factorised representations of relational data. For this challenge, we depart from the central point of this paper, namely factorisation of provenance polynomials. If all we are interested in is compact factorised representation of relational data (under bag or set semantics) and not of provenance polynomials, then the representation discussed in Section 4 can be made more compact by aggregating the variables that annotate empty tuples in the query result. Such aggregates are necessary for computing correctly multiplicities of tuples in case of bag semantics. Under set semantics, these aggregates are not necessary and can be dropped altogether. This challenge is to compute compact representations of relational data using such reduced factorisations.

We next give an example. The query $\pi_{\mathtt{id,location}}(Q)$, where Q is the query from the introduction, reports the IDs and locations of all retrievable orders. Under bag semantics, each such pair appears as many times as there are employees available to retrieve the order from that location. Its factorised representation is

$$\begin{aligned}&(o_1\langle 01\rangle + o_4\langle 04\rangle)(s_1\langle \text{Depot1}\rangle(e_1\langle\rangle + e_2\langle\rangle) + s_3\langle \text{Depot2}\rangle e_3\langle\rangle)+\\ &+o_2\langle 02\rangle s_2\langle \text{Depot1}\rangle(e_1\langle\rangle + e_2\langle\rangle) + (o_3\langle 03\rangle+\\ &+o_5\langle 05\rangle)s_4\langle \text{StoreA}\rangle e_4\langle\rangle.\end{aligned}$$

The variables for tuples from `Emp` annotate the empty tuple. We can interpret it as the multiplicative unit, and aggregate sums of such units:

$$\begin{aligned}&(o_1\langle 01\rangle + o_4\langle 04\rangle)(\mathbf{2}s_1\langle \text{Depot1}\rangle + s_3\langle \text{Depot2}\rangle)+\\ &+\mathbf{2}o_2\langle 02\rangle s_2\langle \text{Depot1}\rangle + (o_3\langle 03\rangle + o_5\langle 05\rangle)s_4\langle \text{StoreA}\rangle.\end{aligned}$$

The coefficients represent multiplicities of tuples: For instance, there are two employees available to retrieve orders 01 and 04 from Depot1.

6. Provenance data management system. We plan to build an open-source scalable management system whose key ingredient are factorised representations at the physical layer, but annotated relations at the logical layer. The challenges that we want to address in particular are the design and implementation of a storage manager aware of the factorised polynomials and of disk-resident query evaluation techniques.

6 Results So Far

We have recently addressed facets of some of the challenges mentioned above [11, 4].

Result 1. A contribution to the first challenge is a tight characterisation of conjunctive queries by bounds on the minimal possible size of their factorised provenance polynomials. The tightness of the characterisation is with respect to factorisations whose nesting structure is independent of the database. For any query we define so-called factorisation trees, which are essentially query join trees, that define nesting structures of provenance polynomials independent of the input database.

To measure the size of a factorisation, we use the total number of occurrences of its variables: The flat polynomial Φ_1 has size $10 \times 3 = 30$, while the factorised form Φ_2 has size 15 only. In fact, Φ_2 is the smallest possible factorised polynomial equivalent to Φ_1.

A different measure for the complexity of a polynomial is its *readability*, which is the minimum over all of its equivalent factorised forms of the maximum number of occurrences of any variable. In the polynomial Φ_1, the variable s_1 occurs 4 times. However, s_1 occurs only once in Φ_2, and no variable occurs more than twice. Also, there is no equivalent factorisation in which each variable occurs at most once. The readability of these equivalent polynomials is thus two. The readability of a polynomial Φ is an important tool that can be used to compute bounds on the minimal possible size of Φ's factorisation.

The notion of readability is borrowed from earlier work on Boolean functions [5]. That strand of work differs from ours in that we only consider polynomials of query results, and classify queries based on worst-case analysis of the readability of their result polynomials.

Expressed in terms of readability, our result states that for any conjunctive query Q we can find a rational number $f(Q)$ derived from the query, such that for any database $\mathbf{D}$ the provenance polynomial of the result $Q(\mathbf{D})$ has readability at most $|\mathbf{D}|^{f(Q)}$. Moreover, there exist arbitrarily large databases for which this bound is asymptotically tight with respect to factorised polynomials whose structures are defined by factorisation trees.

This bound on readability implies an upper bound of $|\mathbf{D}|^{f(Q)+1}$ on the size of the smallest possible factorisation of the provenance polynomial for any database $\mathbf{D}$, which is also tight with respect to factorisation trees, if repeated tuples are allowed in the input database. Our result extends an existing result shown for conjunctive queries under bag semantics [1]. In our context, that result corresponds to the case of flat, that is, non-factorised, polynomials.

Result 2. A further contribution to the first challenge is a dichotomy for conjunctive queries based on whether their provenance polynomials have readability either independent or dependent on the database size. It turns out that these queries are precisely the hierarchical ones [13] that play a central role in studies with seemingly disparate focus, including the present one, probabilistic databases, streamed query evaluation, and parallel query evaluation. Our characterisation of query readability essentially revolves around how far the query is from its hierarchical subqueries. The rational number $f(Q)$ discussed under the first contribution above can thus be seen as the *hierarchy width* of the query, similar in spirit to existing width measures that capture the complexity of conjunctive query, such as the *(hyper)tree width* [6].

The positive result for hierarchical queries draws on earlier work in the context of probabilistic databases [10, 13], where polynomials of readability one in the Boolean semiring and over random variables are useful since their exact probability can be computed in polynomial time. For larger readabilities, probability computation quickly becomes #P-hard [14]. In our case, however, a readability that is polynomial in the sizes of the input database and query is acceptable.

Mirroring the dichotomies in the contexts of probabilistic data and provenance readability, it has been shown recently that the hierarchical property separates queries that can be evaluated in one pass from those that cannot in the finite cursor machine model of computation [8]. In this model, queries are evaluated by first sorting each relation, followed by one pass over each relation. It would be interesting to investigate the relationship between the readability of a query Q and the number of passes necessary in this model to evaluate Q.

Furthermore, in the Massively Parallel computation model, any conjunctive query that can be evaluated (under bag semantics) with one synchronisation step is hierarchical [9].

Result 3. For the second challenge, we have developed algorithms that, for a given conjunctive query Q and any database $\mathbf{D}$, can directly compute factorised provenance polynomials of asymptotically optimal size, that is, of size $O(|\mathbf{D}|^{f(Q)+1})$ for the result of Q on $\mathbf{D}$. Moreover, the running time of the algorithms is bounded above by the same bounds as the size of the factorised polynomials, that is, it runs it time $O(|Q| \cdot |\mathbf{D}|^{f(Q)+1})$ for $f(Q) > 0$ (with an additional $\log |\mathbf{D}|$ factor for hierarchical queries for which $f(Q) = 0$). Therefore, in the cases when the factorised polynomials are exponentially more succint compared to the flat polynomials, these algorithms also bring exponential savings in computation time.

Result 4. For the fourth challenge, we have considered so far the case of approximations of provenance polynomials over the Boolean semiring, where the lower and upper bound approximations are expressed as factorised polynomials with readability one [4]. The key contributions are equivalences of syntactic and model-theoretic characterisations of optimal bounds for provenance polynomials of results to positive relational algebra queries, as well as algorithms to enumerate such bounds with polynomial delay. The bounds can also be computed by first-order queries extended with transitive closure and a choice construct.

Acknowledgment. The authors would like to thank Robert Fink for his useful comments on earlier drafts of this work.

References

[1] Atserias, A., Grohe, M., and Marx, D. Size bounds and query plans for relational joins. In *FOCS* (2008), pp. 739–748.

[2] Cheney, J., Chiticariu, L., and Tan, W. C. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases 1*, 4 (2009).

[3] Elbassioni, K. M., Makino, K., and Rauf, I. On the readability of monotone boolean formulae. In *COCOON* (2009).

[4] Fink, R., and Olteanu, D. On the Optimal Approximation of Queries Using Tractable Propositional Languages. In *ICDT* (2011).

[5] Golumbic, M. C., Peled, U. N., and Rotics, U. Chain graphs have unbounded readability. Tech. rep., University of Haifa, 2006.

[6] Gottlob, G., Leone, N., and Scarcello, F. The complexity of acyclic conjunctive queries. *J. ACM 48* (May 2001), 431–498.

[7] Green, T. J., Karvounarakis, G., and Tannen, V. Provenance semirings. In *PODS* (2007).

[8] Grohe, M., Gurevich, Y., Leinders, D., Schweikardt, N., Tyszkiewicz, J., and den Bussche, J. V. Database query processing using finite cursor machines. *TCS 44*, 4 (2009).

[9] Koutris, P., and Suciu, D. Parallel evaluation of conjunctive queries. In *PODS* (2011). to appear.

[10] Olteanu, D., and Huang, J. "Using OBDDs for Efficient Query Evaluation on Probabilistic Databases". In *SUM* (2008).

[11] Olteanu, D., and Závodný, J. Factorised representations of query results. Tech. rep., Oxford, April 2011. Also arXiv report 1104.0867, available at http://arxiv.org/abs/1104.0867.

[12] Ré, C., and Suciu, D. Approximate lineage for probabilistic databases. *PVLDB 1*, 1 (2008).

[13] Suciu, D., Olteanu, D., Ré, C., and Koch, C. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[14] Vadhan, S. The Complexity of Counting in Sparse, Regular, and Planar Graphs. *SIAM J. Comput. 32*, 2 (2001).