Bringing Provenance to its Full Potential using Causal Reasoning

Alexandra Meliou Wolfgang Gatterbauer Dan Suciu
Department of Computer Science and Engineering,
University of Washington, Seattle, WA, USA
{ameli,gatter,suciu}@cs.washington.edu

Abstract

Provenance information is often used to explain query results and outcomes, exploit results of prior reasoning, and establish trust in data. The generality of the notion makes it applicable in a variety of domains, including data warehousing [7], curated databases [4], and various scientific applications. The recent introduction of causal reasoning in a database setting exploits provenance in ways that expand its applicability to more complex problems, and establish new directions, making a step towards achieving provenance's full potential. In this paper we explore through a variety of examples how causality improves on provenance information, discuss the challenges of building causality able systems, and propose some new directions.

1 Introduction

Understanding provenance ([9, 5, 11]) can lead to interesting facts, insights, or errors, but the task becomes harder as the data grows larger. In many cases, like sensor derived data ([16]), raw provenance is simply not adequate to provide explanations for surprising observations and errors without further analytical reasoning.

Causality is related to provenance, yet it is a more refined notion that can provide reasons and explanations for wrong or surprising results, by ranking provenance based on the notion of *responsibility*. It has been discussed in relation to data provenance [15, 16, 17, 8], and even workflow provenance [20, 19]. In this paper, we overview the improvements that causality offers over raw provenance, discuss through various examples several applications that can benefit from this type of reasoning, and deliberate over challenges and new directions.

2 Causality Preliminaries

Causality in databases [16, 17] came as an adaptation of established notions of causality from the AI and philosophy literature [21, 14, 12]. We briefly review here some basic notions of causality that map to database concepts.

Exogenous/endogenous. Variables can be partitioned into an endogenous and exogenous set. Exogenous variables define the context of the problem, and are considered as external factors that are not of interest to the current problem statement, whereas endogenous variables are judged on their effect to the outcome. This partitioning is a first step in filtering lineage: tuples, tables, and various parts of the transformations may be deemed irrelevant to a given provenance problem for various reasons. For example, data originating from trusted sources, or data that a user has no control over, and has no ability to change, can be regarded as exogenous, defining a context for the problem, but not being a part of interesting/relevant solutions. Thus the relevant parts of data lineage can be drastically reduced.

Counterfactuals. Counterfactuals express a strong dependency relationship between cause and effect, one that is defined by a counterfactual statement: "X is a counterfactual cause for Y, if Y would not have happened in the absence of X". The notion of counterfactuals can be generalized with the use of *contingencies*: a tuple is a cause, if there exists a hypothetical setting (contingency) that makes it counterfactual.

Responsibility. A notion first defined in [10], responsibility quantifies the causal relevance of a variable to an output, based on the size of the smallest contingency set. The responsibility ρ_t of a tuple t is a score in the range $[0, 1]$, with 1 representing counterfactual causes. The smaller the contingency set (empty for counterfactual causes), the higher the responsibility score. By convention, if t is not a cause, its responsibility is 0. Responsibility is a very useful notion for provenance, as it can quantify the contribution of tuples to a result.

3 Causality and Provenance

The notions of contingencies, responsibility, and the partitioning into exogenous and endogenous variables can enrich the raw provenance information, extending its applicability to a variety of applications. Note that contingencies and responsibility do not require extra information or input, but can be computed using the base lineage expressions. The exogenous/endogenous partitioning may be part of the user input, database or system specifications.

In this section, we present various examples that showcase the benefits that causality brings to provenance, and discuss their challenges and possible new directions.

3.1 Provenance Ranking

Using causality for explaining query answers [16, 17] relies on two components: discovering causes within the endogenous tuples, and *ranking* those causes based on the responsibility of the tuples in the lineage of a result. This responsibility-based ranking is one of the most important contributions of causality to lineage, and is relevant to any application, beyond explanations, that needs to quantify contributions of variables (e.g error detection/correction). Having such a metric is crucial when the lineage of results is large, and it becomes tedious and impractical to manually examine it.

We will discuss an example that demonstrates how this ranking aids in explanations of surprising results, and error discovery. The example is based on the publicly available Never Ending Language Learning (NELL) dataset [6, 1]. The dataset contains information extracted automatically from various online sources, and is comprised of one large relation (`nell`) with the following attributes: entity, relation, value, iteration, probability, source, candidate source, entity literal strings, value literal strings. For specifications on all the fields the reader can see [1]. For our example we can focus on the first three fields, which specify for various diverse entities, their value under a specific context (relation). Due to the nature of its construction, the NELL dataset contains quite a few errors and dirty data. Figure 1 gives some example entries from the `nell` table.

EXAMPLE 3.1 (NELL DATASET). *A user has downloaded the NELL dataset [6, 1], and issues various exploratory queries. She wants to find a list of all languages that serve as an official language to at least one country that has multiple official languages. For example, Belgium has both French and German as official languages (Fig. 1), so both French and German should appear in the query result. She issues the following query over the* `nell` *relation:*

```
select distinct n1.value
from   nell n1, nell n2
where  n1.relation = 'countrylanguage'
       and n2.relation = n1.relation
       and n2.entity = n1.entity
       and n1.value <> n2.value;
```

entity	relation	value
brett_johnson	generalizations	athlete
tim_burton	directordirectedmovie	beetlejuice
mark_zuckerberg	ceoof	facebook
barak_obama	politicianholdsoffice	president
belgium	countrylanguage	french
belgium	countrylanguage	german
greece	countrylanguage	greek
greece	countrycities	chania

Figure 1: Some sample tuples from the NELL database. The actual tuple data is abbreviated for presentation purposes (e.g. `athlete` instead of `concept:athlete`).

and gets the following languages in the result:

afrikaans	french	persian
arabic	german	polish
cantonese	greek	portuguese
english	mandarin	spanish

The user is surprised to see Greek and Persian in the result set, as she believes these languages to be only spoken as the single official language in Greece and Iran respectively. The user wants to understand the reason for this outcome, and correct any possible errors.

The responsibility ranking can provide the user with a meaningful measure of the importance of various tuples in a result, which can translate in more meaningful explanations, or error likelihood. This is important when the provenance of a tuple becomes large, and impractical to manually explore. Even for the simple query of Example 3.1, the suspicious results map to 30 base tuples. We explain the resulting ranking in more detail only for greek, whose provenance is much smaller.

EXAMPLE 3.2 (EXAMPLE 3.1 CONTINUED). *The user issues a causality query on the result* `greek`*, and receives the responsibility ranking of Fig. 2. As expected by the user, the tuple* `(greece, greek)` *(we omit the relation value for brevity) appears as a top result in the ranking, as it is an expected fact. However, the user can also easily notice that a second language (English) is also listed as an official language of the country. Even though many Greeks do speak English, the tuple is factually wrong. Moreover, the fact that the tuple is not counterfactual ($\rho_t < 1$) means that there are additional errors: apparently Greek is also listed as an official language for Egypt, which is also a mistake. Note that the tuple* `(egypt, greek)` *is ranked higher than* `(egypt, arabic)` *and* `(egypt, persian)`*. This is important, as the other languages spoken in Egypt are not as relevant to this result (only the fact that there is at least on of them).*

ρ_t	tuple
0.5	(greece, countrylanguage, greek)
0.5	(greece, countrylanguage, english)
0.5	(egypt, countrylanguage, greek)
0.3	(egypt, countrylanguage, arabic)
0.3	(egypt, countrylanguage, persian)

Figure 2: Responsibility ranking for the result `greek`.

The responsibility ranking successfully identifies a relevant fact (first tuple), and two errors (second and third tuples). The details on how responsibility is computed, as well as complexity results, can be found in [16]. Note that the last tuple `(egypt, persian)` also happens to be wrong, but the error is not relevant to this particular causality query ("why is `greek` in the result?").

Responsibility is a meaningful metric for explanations in the presence of few errors. For example, if there were multiple wrong entries for Greece, their individual responsibility would be lower. This is a natural behavior, as we cannot assume the system to have any knowledge of the actual semantics of the data. If the number of errors overwhelms the correct tuples in a dataset, causality as well as any scheme that does not use any other additional semantics should be expected to fare more poorly.

3.1.1 System Challenges

To support causal queries, we need to establish appropriate language constructs, possibly in the form of SQL extensions. We need to be able to handle three things: (a) specifying the output of interest (surprising or erroneous result), (b) the query that generated it, and (c) the exogenous and endogenous parts of the provenance.

The last one is probably the most challenging and interesting part of the system. The specification can be done at the schema level (entire relations are characterized as endogenous/exogenous), or at the tuple level, preferably in a declarative way. It is possible that some specifications can be done on an administrative level, for example if the validity of some relations or sources has been verified, the corresponding tuples can be labeled as exogenous. However, it is necessary that the user has the capability to personalize her specifications, based on the data/sources that she trusts, or provide further specifications on a per-query basis.

Extensions. Endogenous and exogenous data is currently defined in a binary way (a tuple is either exogenous or endogenous). However, some problems would benefit from an *endogenous score* which quantifies an initial scale of relevance for the base data. Instead of specifying for example that relation R is exogenous and S endogenous, meaning that the user is only interested in causes in S, the user may prefer to specify a priority score between the two tables, which would affect the final responsibility ranking.

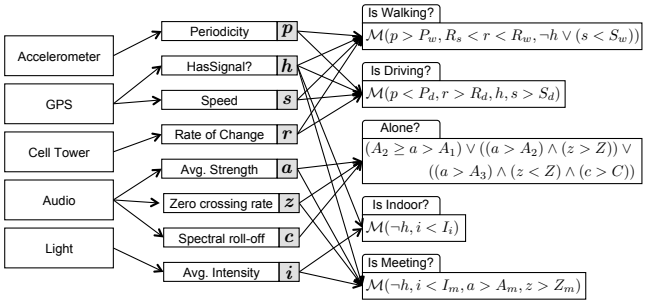

Figure 3: Structure of the classifier system used in [18].

3.2 Post-factum Cleaning

In some problem settings, dirty data cannot be detected with traditional cleaning techniques. [18] showcases a sensor-based context-aware recommendation engine (CARE) that uses *view-conditioned causality* to trace errors in sensory input. Errors in sensor data can go undetected, even by an expert user, as the data lacks context. For example, a series of values of the sensor features (p, h, s, t, r, c, z, a, i) of Fig. 3 can resemble the following:

```
(0.016,True,0.067,0,0.4,0.004,0.86,0.036,10)
(0.0009,False,0,0,0.2,0.0039,0.81,0.034,68)
(0.005,True,0.19,0,0.3,0.003,0.75,0.033,17)
(0.0008,True,0.003,0,0.1,0.003,0.8,0.038,18)
```

It is not clear what the context of this data is, and whether there are errors. Even assuming that the sensors are not faulty, it is not possible to know just by looking at the data whether a sensor is inhibited or recording something out of context. For example, the light sensor may be producing out-of-context readings if the user stores his cell phone in the car's glove compartment.

However, the presence of errors may become obvious after the data undergoes transformations. CARE [18] uses the extracted sensor features in a classification system to infer the user's current activities (e.g. walking, driving, in a business meeting etc.), and serves him with appropriate recommendations. Using the user's direct feedback or subsequent actions, the system can determine if some classifications were right or wrong. We can use causality to perform *post-factum data cleaning*: while in standard cleaning errors are corrected before the data is transformed and integrated, in this setting the errors are detected only *after* the transformation. In this case we have a much richer context based on multiple correct and incorrect results. *View-Conditioned Causality* conditions on the full known context of the output result, and is better suited for error tracing problems.

3.2.1 System Challenges

In addition to the system challenges of simple causality, view-conditioned causality further requires that the

user be able to specify as much of the output context as is available, labeling all known correct and incorrect results as such. This may be done on a per-tuple basis, or declaratively in some cases. It is possible that some errors can be automatically detected, e.g. the driving and walking classifications (Fig. 3) cannot be true together. The user should be able to specify assertion-like rules that identify such cases, e.g. if d and w denote the driving and walking classifications respectively, the rule could be $\neg d \vee \neg w =$`true`. The system can then alert the user of the existence of some error, or even attempt to automatically determine the output that is most likely wrong.

Extensions. In post-factum cleaning the data undergoes transformation before it becomes possible to detect problems. It is possible however that some errors can still go undetected, if they don't happen to produce obvious mistakes in the transformed data. In the example of [18] the error tracing is performed "on demand", whenever a mistake is detected in the classification. An interesting problem would be to perform post-factum cleaning proactively. Is it possible to construct transformations that can guarantee error detection?

3.3 Satisfying Aggregate Specifications

In some problem settings, requested changes may not be a matter of errors, but rather desired modifications. *What-if* or *hypothetical* queries [13, 3] attempt to address questions of the form "How would the output change for a proposed change in the input?". Their stronger motivation comes from the domain of business intelligence, where they are used for strategy evaluations and decisions. In an example from [3], an analyst from a brokerage company wants to investigate *what* would be the effect in the returns and volatility of the customer portfolios, *if* during the last 3 years the company had recommended buying Intel stock instead of Motorola.

As a "mirror" problem, a *reverse what-if query* or *how-to query* would ask "How could I have achieved 10% more return to all portfolios with the minimum number of trades". In essence, this problem relates to finding causes for aggregates. More specifically, we want to change a specific aggregate from value A to value B, and we need to find a *contingency set* in the provenance of the aggregate that achieves that change, under some optimality criterion. These contingency sets would then be counterfactual causes for changing the aggregate value from A to B. The problem statement may actually specify multiple aggregate requirements, and possibly constraints that the solution should satisfy (e.g. "the total investment amount should not exceed the current total"). Relevant to aggregate specifications is also recent work on the provenance of aggregates [2].

3.3.1 System Challenges

Aggregate specifications pose many interesting challenges, but due to space constraints we only discuss here the system's basic functionality. Our system should have language constructs to support the following specifications:

(a) Objective functions defining the optimization criterion. A sample language construct could be:
`CREATE OBJECTIVE <obj_name> AS <sql_expr>`

(b) Multiple constraints over the solutions. An example could be:
`CREATE CONSTRAINTS <constr_name> AS <sql_expr>`

(c) Aggregate specifications of the form:
`HOW TO <aggr1> op <value1>, ...`
`MAX/MIN <obj_name> SUBJECT TO <constr_name>`

If an objective function is not provided, the system can present the user with a set of possible solutions.

4 Conclusions

In this paper, we presented a variety of provenance related problems (ranking, error-tracing, aggregate modifications) that causal reasoning can handle cleanly and effectively, and proposed new directions and various challenges related to building a causality enabled system. In its various flavors, causality enriches provenance with a layer of analytical reasoning, that allows us to tackle many complex and interesting problems, making better use of provenance's underutilized potential.

Acknowledgements. This work was partially supported by NSF grants IIS-0915054 and IIS-0911036, and was performed in the context of the Causality in Databases project (`http://db.cs.washington.edu/causality/`).

References

[1] NELL: Never Ending Language Learning. http://rtw.ml.cmu.edu/rtw/.

[2] Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. *CoRR*, abs/1101.1110, 2011.

[3] A. Balmin, T. Papadimitriou, and Y. Papakonstantinou. Hypothetical queries in an olap environment. In *VLDB*, pages 220–231, 2000.

[4] P. Buneman, J. Cheney, W. C. Tan, and S. Vansummeren. Curated databases. In *PODS*, pages 1–12, 2008.

[5] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. H. Jr., and T. M. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[7] S. Chaudhuri and U. Dayal. Data warehousing and olap for decision support (tutorial). In *SIGMOD*, pages 507–508, 1997.

[8] J. Cheney. Causality and the semantics of provenance. In *DCM*, pages 63–74, 2010.

[9] J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[10] H. Chockler and J. Y. Halpern. Responsibility and blame: A structural-model approach. *J. Artif. Intell. Res. (JAIR)*, 22:93–115, 2004.

[11] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[12] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *Brit. J. Phil. Sci.*, 56:843–887, 2005. (Conference version in *UAI*, 2001).

[13] L. V. S. Lakshmanan, A. Russakovsky, and V. Sashikanth. What-if olap queries with changing dimensions. In *ICDE*, pages 1334–1336, 2008.

[14] D. Lewis. Causation. *The Journal of Philosophy*, 70(17):556–567, 1973.

[15] A. Meliou, W. Gatterbauer, J. Y. Halpern, C. Koch, K. F. Moore, and D. Suciu. Causality in databases. *IEEE Data Eng. Bull.*, 33(3):59–67, 2010.

[16] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. The complexity of causality and responsibility for query answers and non-answers. *PVLDB*, 4(1):34–45, 2010.

[17] A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. Why so? or Why no? Functional causality for explaining query answers. In *MUD*, 2010.

[18] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD*, 2011.

[19] S. Miles, P. T. Groth, S. Munroe, S. Jiang, T. Assandri, and L. Moreau. Extracting causal graphs from an open provenance data model. *Concurrency and Computation: Practice and Experience*, 20(5):577–586, 2008.

[20] K.-K. Muniswamy-Reddy and D. A. Holland. Causality-based versioning. *TOS*, 5(4), 2009.

[21] J. Pearl. *Causality: models, reasoning, and inference*. Cambridge University Press, Cambridge, U.K., 2000.