Andrea Calì
Maria-Esther Vidal (Eds.)

AMW2015

Alberto Mendelzon International Workshop on Foundations of Data Management

Lima, Perú, May 6th-8th, 2015
Proceedings

Editors' addresses:
Birkbeck College, University of London, UK
andrea@dcs.bbk.ac.uk;

Universidad Simón Bolívar
Department of Computer Science
Valle de Sartenejas
Caracas 1086, Venezuela
mvidal@ldc.usb.ve

Preface

The Alberto Mendelzon Workshop (AMW) is a Latin American initiative started in 2006 to honor the memory of Alberto Mendelzon, who gave a significant contribution to the field of data management. The AMW is a research venue for top-quality research on foundations of data management; while focused especially on Latin American students and scholars, the workshop is open to submissions from anywhere and it has so far gathered some of the world's best researchers in the field.

This volume contains papers accepted at the 9th edition of the AMW, held in Lima, Perú, from the 6th to the 8th of May, 2015. The call for papers of this edition requested two types of submissions: regular and short papers, where the latter were intended to present ongoing research, results previously published, or data management applications. With around 40 submissions, we could put up an exciting program, which gave raise to discussions and new ideas. Despite the beauties of Lima, including its sophisticated cuisine and drinks, the attendees and the organizers worked hard and made the AMW 2015 a great success, both from the scientific and organizational point of view. The school, which took place before the workshop, provided local students with tutorials whose quality was on par with those at the best conferences in the area. The invited talks at the workshop were also certainly world-class and given by leading researchers: Guy van den Broeck (KU Leuven, Belgium), Wagner Meira Jr (Universidade Federal de Minas Gerais, Brazil), Dan Olteanu (University of Oxford, UK), and Victor Vianu (UC San Diego, USA).

For what we are proud to call a very exciting scientific event, we would like to warmly thank, in no particular order: the authors of the papers, the Program Committee and the external reviewers, the local organizers, the AMW School committee, the Steering Committee, the General Chair, and the local supporting institutions. Without the effort of all the above, the AMW 2015 could not have been successful.

We are also looking forward to the next editions of the Alberto Mendelzon Workshop, which has become an established and high-quality venue in the area of Data Management.

Lima, May 2015

Andrea Calì[1]
Maria-Esther Vidal

[1] Andrea Calì acknowledges support from the EPSRC grant "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1)

General Chair

Pablo Barceló (Universidad de Chile, Chile)

PC Chairs

Andrea Cali, Birkbeck College, University of London, UK)
Maria-Esther Vidal, Universidad Simón Bolívar, Venezuela

Program Committee

Cristina Dutra De Aguiar Ciferri (Universidade de Sao Paulo, Brazil)
Oscar Corcho (Universidad Politecnica de Madrid, Spain)
Isabel Cruz (University of Ilinnois at Chicago, USA)
Amelie Gheerbrant (Universite Paris VII Denis Diderot, France)
Parke Godfrey (York University, Canada)
Sergio Greco (Universita della Calabria, Italy)
Claudio Gutierrez (Universidad de Chile)
Mauricio A. Hernandez-Sherrington (IBM Research Almaden, USA)
Aidan Hogan (Universidad de Chile)
Elizabeth Leon (Universidad Nacional de Colombia)
Jorge Lobo (ICREA and Universidad Pompeu Fabra, Spain)
Maria Vanina Martinez (Universidad Nacional del Sur, Argentina)
Filip Murlak (University of Warsaw, Poland)
Mauricio Osorio (Universidad de las Americas Puebla, Mexico)
Reinhard Pichler (Vienna University of Technology, Austria)
Andreas Pieris (Vienna University of Technology, Austria)
Alessandro Provetti (University of Messina, Italy)
Jarek Szlichta (University of Ontario, Canada)
Regina Paola Ticona Herrera (Universite de Pau et des Pays de l'Adour, France)
Riccardo Torlone (Universita Roma Tre)
Peter Wood (Birkbeck, University of London, UK)

Steering Committee

Local Supporting Institutions

Universidad Mayor de San Marcos

- Facultad de Ingeniería de Sistemas e Informática
- Facultad de Ciencias Matemáticas

Universidad Ricardo Palma

- Facultad de Ingeniería Informática

Universidad Católica de San Pablo

- Escuela Profesional de Ciencia de la Computación

Sociedad Peruana de Computación

IEEE- Sección Peú

Colegio de Matemáticos de Peú

Contents

A Database Framework for Classifier Engineering

Benny Kimelfeld[1] and Christopher Ré[2]

[1] LogicBlox, Inc. and Technion, Israel
[2] Stanford University

1 Introduction

In the design of machine-learning solutions, a critical and often the most resourceful task is that of feature engineering [7, 4], for which recipes and tooling have been developed [3, 7]. In this vision paper we embark on the establishment of database foundations for feature engineering. We propose a formal framework for classification, in the context of a relational database, towards investigating the application of database and knowledge management to assist with the task of feature engineering. We demonstrate the usefulness of this framework by formally defining two key algorithmic challenges within: (1) *separability* refers to determining the existence of feature queries that agree with the given training examples, and (2) *identifiability* is the task of testing for the property of independence among features (given as queries). Moreover, we give preliminary results on these challenges, in the context of conjunctive queries. We focus here on boolean features that are represented as ordinary database queries, and view this work as the basis of various future extensions such as numerical features and more general regression tasks.

2 Formal Framework

We first present our formal framework for classification with binary features within a relational database.

2.1 Classifiers and Learning

In this work, a *classifier* is a function of the form

$$\gamma : \{-1, 1\}^n \to \{-1, 1\}$$

where n is a natural number that we call the *arity* of γ. A *classifier class* is a (possibly infinite) family Γ of classifiers. We denote by Γ_n the restriction of Γ to the n-ary classifiers in Γ. An n-ary *training collection* is a multiset T of pairs $\langle \mathbf{x}, y \rangle$ where $\mathbf{x} \in \{-1, 1\}^n$ and $y \in \{-1, 1\}$. We denote by $\mathbf{T}_n$ the set of all n-ary training collections. A *cost function* for a classifier class Γ is a function of the form

$$c : \big(\cup_n \left(\Gamma_n \times \mathbf{T}_n \right) \big) \to \mathbb{R}_{\geq 0}$$

where $\mathbb{R}_{\geq 0}$ is the set of nonnegative numbers. In the context of a classifier class Γ and a cost function c, *learning* a classifier is the task of finding a classifier $\gamma \in \Gamma_n$ that minimizes $c(\gamma, T)$, given a training collection $T \in \mathbf{T}_n$.

We illustrate the above definitions on the important class of *linear classifiers*. An n-ary linear classifier is parameterized by a vector $\mathbf{w} \in \mathbb{R}^n$, is denoted by $\Lambda_{\mathbf{w}}$, and is defined as follows for all $\mathbf{a} \in \{-1, 1\}^n$.

$$\lambda_{\mathbf{w}}(\mathbf{a}) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } \mathbf{a} \cdot \mathbf{w} \geq 0; \\ -1 & \text{otherwise.} \end{cases}$$

where "$\cdot$" denotes the operation of dot product. By Lin we denote the class of linear classifiers. An example of a cost function is the *least square* cost lsq that is given by

$$lsq(\Lambda_{\mathbf{w}}, T) \overset{\text{def}}{=} \sum_{\langle \mathbf{x}, y \rangle \in T} (\mathbf{x} \cdot \mathbf{w} - y)^2$$

for the arguments $\Lambda_{\mathbf{w}} \in \mathsf{Lin}_n$ and $T \in \mathbf{T}_n$.

2.2 Relational Formalism

Our relational terminology is as follows. A *schema* is a pair $(\mathcal{A}, \Sigma)$, where $\mathcal{A}$ is a *signature* that consists of *relation symbols*, and Σ is a set of logical integrity constraints over $\mathcal{A}$. Each relation symbol R has an associated arity. We assume an infinite set Const of *constants*. An *instance* I over a schema $\mathbf{S} = (\mathcal{A}, \Sigma)$ associates with every k-ary relation symbol $R \in \mathcal{A}$ a finite subset R^I of Const^k, such that all the constraints of Σ are satisfied. The *active domain* of an instance I, denoted $adom(I)$, is the set of all the constants in Const that are mentioned in I.

Let $\mathbf{S}$ be schema. A *query* (*over* $\mathbf{S}$) is a function Q that is associated with an arity k, and that maps every relation instance I over $\mathbf{S}$ into a finite subset $Q(I)$ of Const^k. A query Q' *contains* a query Q if $Q(I) \subseteq Q'(I)$ for all instances I over $\mathbf{S}$; if $Q \subseteq Q'$ and $Q' \subseteq Q$ then Q and Q' are said to be *equivalent*. A query Q is *additive* if for every two instances I_1 and I_2, if $adom(I_1)$ and $adom(I_2)$ are disjoint, then

$$Q(I_1 \cup I_2) = Q(I_1) \cup Q(I_2)\,.$$

A *query class* is a mapping that associates with every schema $\mathbf{S}$ a class of queries over $\mathbf{S}$. An example of a query class is that of the *conjunctive queries*. A conjunctive query (CQ) is represented by the logical formula $q(\mathbf{x})$ that has the form

$$\exists \mathbf{y}[\phi_1(\mathbf{x}, \mathbf{y}, \mathbf{d}) \wedge \cdots \wedge \phi_m(\mathbf{x}, \mathbf{y}, \mathbf{d})]$$

where $\mathbf{x}$ and $\mathbf{y}$ are disjoint sequences of variables, $\mathbf{d}$ is a sequence of constants, and each ϕ_i is an atomic query over $\mathbf{S}$ (i.e., a formula that consists of a single relation symbol and no logical operators). The result of applying the CQ $Q = q(\mathbf{x})$ to the instance I consists of all the tuples $\mathbf{a}$ (of the same length as $\mathbf{x}$) such that $q(\mathbf{a})$ is true in I; we denote this result is denoted by $Q(I)$.

2.3 Classification Framework

We now present our formal framework. An *entity schema* is a triple $(\mathcal{A}, \Sigma, E)$, where $(\mathcal{A}, \Sigma)$ is a schema and E is a relation symbol in $\mathcal{A}$ that represents the *entities* (as tuples). An *instance* I over a schema $(\mathcal{A}, \Sigma, E)$ is simply an instance over $(\mathcal{A}, \Sigma)$. Intuitively, an instance I over an entity schema $(\mathcal{A}, \Sigma, E)$ represents a set of entities, namely E^I (i.e., the set of tuples in E), along with information about the entities that is contained in the remaining relations R^I. For example, E may be the relation Persons and $\mathcal{A}$ may include, besides Persons, relations such as PersonAddress, PersonCompany, CompanyCity, and so on. If $\mathbf{S} = (\mathcal{A}, \Sigma, E)$ is an entity schema, then the elements $\mathcal{A}$, Σ and E are denoted by $\mathcal{A}_{\mathbf{S}}$, $\Sigma_{\mathbf{S}}$ and $E_{\mathbf{S}}$, respectively.

Let $\mathbf{S}$ be an entity schema, and let I be an instance over $\mathbf{S}$. A *feature query* (*over* $\mathbf{S}$) is a query π over the schema $\mathbf{S}$, such that $\pi \subseteq E_{\mathbf{S}}$, where $E_{\mathbf{S}}$ is viewed as the query that, given an instance I, copies the relation $E^I_{\mathbf{S}}$. In other words, a feature query is a query that selects entities. For example, if E is $\mathsf{Persons}(ssn, name)$ then a feature can be the following CQ $q(s, n)$ (selecting persons working in New York City).

$$\exists_c \big[\mathsf{Persons}(s,n) \land \mathsf{PersonCompany}(s,c) \land \mathsf{CompanyCity}(c, \text{`NYC'})\big]$$

If π is a feature query, then π^I denotes the function $f : E^I_{\mathbf{S}} \to \{-1, 1\}$ where

$$f(e) = \begin{cases} 1 & \text{if } e \in \pi(I); \\ -1 & \text{otherwise.} \end{cases}$$

A *statistic* (over $\mathbf{S}$) is a sequence $\Pi = (\pi_1, \dots, \pi_n)$ of feature queries. We denote by Π^I the function $(\pi^I_1, \dots, \pi^I_n)$ from $E^I_{\mathbf{S}}$ to $\{-1, 1\}^n$.

A *feature schema* is a pair $(\mathbf{S}, \Pi)$, where $\mathbf{S}$ is an entity schema, and Π is a statistic over $\mathbf{S}$ that produces a sequence of features for every entity of a given input instance. We say that Π and $(\mathbf{S}, \Pi)$ are in a query class $\mathbf{Q}$ if every query in Π belongs to $\mathbf{Q}$. A *training instance* over $\mathbf{S}$ is a pair (I, o), where I is an instance over $\mathbf{S}$ and $o : E^I_{\mathbf{S}} \to \{-1, 1\}$ is a function that partitions the entities into positive and negative examples. Given a feature schema $(\mathbf{S}, \Pi)$ and a classifier class Γ, the training instance (I, o) defines the training collection that consists of the tuple $\langle \Pi^I(e), o(e) \rangle$ for every $e \in E^I_{\mathbf{S}}$.

3 Feature Engineering

In *feature engineering*, one devises feature queries for a classification task. Next, we discuss two computational challenges that naturally arise in feature engineering.

3.1 Separability

Let $(\mathbf{S}, \Pi)$ be a feature schema, and let Γ be a classifier class. A training instance (I, o) is said to be *Γ-separable* with respect to (w.r.t.) Π if there exists a classifier $\gamma \in \Gamma$ that fully agrees with o; that is, γ and Π have the same arity, and $\gamma(e) = o(e)$ for every $e \in E^I_{\mathbf{S}}$. We define the following core computational problem.

Problem 1 (Separability). Let $\mathbf{S}$ be an entity schema, let $\mathbf{Q}$ be a query class over $\mathbf{S}$, and let Γ be a classifier class. The *separability problem* is the following. Given a training instance (I, o) over $\mathbf{S}$, determine whether there exists a statistic Π in $\mathbf{Q}$ such that (I, o) is Γ-separable w.r.t. Π.

The separability problem, as defined, can be extended in various practical directions. The input can include a bound N on the length n of the statistic Π (hence, limiting the *model complexity*, which results in classifiers that are more efficient and less overfitting). One can allow for an *approximate* agreement with o (e.g., the classifier should agree with o on at least $(1 - \epsilon)$ of the entities, or at most k examples should be misclassified). And one can impose various constraints on common query classes $\mathbf{Q}$ (e.g., limit the size of queries, number of constants, etc., again to limit the model complexity and potential overfitting). The following theorem considers the complexity of testing for separability in the case where the class of queries is that of CQs without constants,[3] which we denote by $\mathsf{CQ}^{\mathsf{nc}}$. It states that, in the absence of such extensions of the problem, it can very quickly get intractable.

Theorem 1. *Let* $\mathbf{Q}$ *be the class* $\mathsf{CQ}^{\mathsf{nc}}$*, and let* Γ *be the class* Lin*. For every entity schema* $\mathbf{S}$*, separability is in NP. Moreover, there exists an entity schema* $\mathbf{S}$ *such that separability is NP-complete.*

The proof of membership in NP is using the concept of a *canonical database* [1], and the proof of NP-hardness is by a reduction from the maximum-clique problem.

We note that a problem similar to separability has been studied in a recent paper by Cohen and Weiss [2], where data are labeled graphs and features are tree patterns.

3.2 Statistic Identifiability

We denote by $\mathbf{0}^m$ the vector of m zeroes. Let M be an $n \times k$ real matrix. A *linear column dependence* in M is a weight vector $\mathbf{w} \in \mathbb{R}^k$ such that $\mathbf{w} \neq \mathbf{0}^k$ and $M \cdot \mathbf{w} = \mathbf{0}^n$; if M does not have any linear column dependence, then we say that M is *linearly column independent*. Let $(\mathbf{S}, \Pi)$ be a feature schema, and let I be an instance of $\mathbf{S}$. We fix an arbitrary order over the entities in $E^I_{\mathbf{S}}$, and denote by $[\![\Pi^I]\!]$ the matrix that consists of the rows $\Pi^I(e)$ for every $e \in E^I_{\mathbf{S}}$ in order. The second computational problem we define is the following.

Problem 2 (Identifiability). Let $\mathbf{Q}$ be a query class. *Identifiability* is the problem of testing, given a feature schema $(\mathbf{S}, \Pi)$ in $\mathbf{Q}$, whether there exists an instance I over $\mathbf{S}$ such that the matrix $[\![\Pi^I]\!]$ is linearly column independent; in that case, we say that Π is *identifiable*.

Identifiability is an important property in the design of machine-learning solutions [5]. Particularly, in the case of the classifier class Lin and the cost function lsq, this property implies that there is a single optimal classifier, whereas its absence implies that the space of optimal solutions is unbounded.

[3] For CQs with constants, the problem is trivial and not interesting, since the positive examples can be hardcoded into the statistic.

Next, we show that in the case of CQs. identifiability amounts to query equivalence. A statistic Π is said to have *redundancy* if it contains two distinct feature queries that are equivalent.

Theorem 2. *Let* $\mathbf{Q}$ *be the query class of additive CQs, and let* $(\mathbf{S}, \Pi)$ *be a feature schema such that* Π *is in* $\mathbf{Q}$*. Then* Π *is identifiable if and only if* Π *has no redundancy.*

We conclude with comments on Theorem 2. First, this theorem is proved again by applying the concept of a canonical database of a CQ. Second, we can extend the theorem to the class of all CQs, but the condition that characterizes identifiability is significantly more complicated (and will be given in the extended version of this paper). Third, this theorem generalizes to *affine independence*, which is important in different cost functions such as *maximum entropy* [6]. Finally, by an immediate application of the NP-completeness of CQ containment [1] we get that identifiability is NP-complete in the case of additive CQs.

Acknowledgments

Benny Kimelfeld is a Taub Fellow, supported by the Taub Foundation. The research of Christopher Ré is supported by DARPA's projects XDATA (FA8750-12-2-0335), DEFT (FA8750-13-2-0039), MEMEX, and SIMPLEX. His research is also supported by NSF Career Award IIS-1353606, ONR Awards N000141210041 and N000141310129, NIH Grant U54EB020405 (awarded by NIBIB through funds provided by the trans-NIH BD2K initiative), the Sloan Research Fellowship, the Moore Foundation, American Family Insurance, Google, and Toshiba. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, AFRL, NSF, ONR, NIH, or the U.S. government.

References

1. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In J. E. Hopcroft, E. P. Friedman, and M. A. Harrison, editors, *STOC*, pages 77–90. ACM, 1977.
2. S. Cohen and Y. Y. Weiss. Learning Tree Patterns from Example Graphs. In M. Arenas and M. Ugarte, editors, *18th International Conference on Database Theory (ICDT 2015)*, volume 31 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 127–143, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
3. I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
4. S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2917–2926, 2012.
5. E. L. Lehmann and G. Casella. *Theory of point estimation*, volume 31. Springer, 1998.
6. M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.
7. C. Zhang, A. Kumar, and C. R. Materialization optimizations for feature selection workloads. In *SIGMOD Conference*, pages 265–276, 2014.

Extending Datalog with Analytics in LogicBlox

Molham Aref[1], Benny Kimelfeld[⋆1,2], Emir Pasalic[1], and Nikolaos Vasiloglou[1]

[1] LogicBlox, Inc.
[2] Technion, Israel

1 Introduction

LogicBlox is a database product designed for enterprise software development, combining transactions and analytics. The underying data model is a relational database, and the query language, LogiQL, is an extension of Datalog [13]. As such, LogiQL features a simple and unified syntax for traditional relational manipulation as well as deeper analytics. Moreover, its declarative nature allows for substantial static analysis for optimizing evaluation schemes, parallelization, and incremental maintenance, and it allows for sophisticated transactional management [11]. In this paper, we describe various extensions of Datalog for supporting prescriptive and predictive analytics. These extensions come in the form of mathematical optimization (mixed integer programming), machine-learning capabilities, statistical relational models, and probabilistic programming. Some of these extensions are currently implemented in LogicQL, while others are in either development or planning phases.

2 LogiQL Basics

In this section we briefly (and informally) describe LogiQL. (See [13] for a full description of the language.) The core components of LogiQL are its rules and constraints, which are stated over the *predicates* of the database schema. A *(basic) derivation rule* is a standard Datalog rule that defines how new facts are derived in the database. For example, the following rules define Anc as the transitive closure of the predicate Par.

$$\mathrm{Anc}(x,y) \leftarrow \mathrm{Par}(x,y)$$
$$\mathrm{Anc}(x,y) \leftarrow \mathrm{Anc}(x,z), \mathrm{Par}(z,y)$$

An *integrity constraint* does not derive new facts, but rather fires an error when violated. For example, the rule $\mathrm{Anc}(x,y) \rightarrow x \neq y$ states that ancestorship is antireflexive, and the rule

$$\mathrm{Par}(x,y), \mathrm{Par}(x,z) \rightarrow y = z$$

states that every object has at most one parent, that is, in Par the first element is a key. For interpretability sake, brackets are used to implicitly express key constraints, as in $\mathrm{Par}[x] = y$. Derivation rules and constraints syntactically differ in the direction of the implication arrow. The logical language for specifying the right-hand-side of

⋆ Taub Fellow – supported by the Taub Foundation rules, as well as both sides of constraints, allows arbitrary propositional formulas with numerical operations and comparisons (while safety conditions, such as stratification, may be imposed to bound complexity).

Predicate-to-Predicate (*P2P*) rules are used for deriving whole relations over tuples. An example of such a rule is the *aggregate* P2P rule, such as the following one that sums up the salaries for each employee.

$$\mathrm{Annual}[e] = a \leftarrow \mathsf{agg}{\ll} a = \mathsf{sum}(s){\gg}\ \mathrm{Salary}(e, 2014, s)$$

Such a rule always includes a component of the form $\mathsf{func}{\ll}\mathsf{arg}{\gg}$, where func is the type of the predicate rule, and arg is an additional argument that gives specific arguments for the rule. In the above rule, e is a *grouping variable* (as it occurs in the head). This rule derives a fact $\mathrm{Annual}(e, a)$ for every e in the first attribute of Salary.

3 Extensions for Analytics

We now describe several extensions of the LogicBlox system, for supporting prescriptive and predictive analytics.

3.1 Mixed Integer Programming (MIP)

MIP is often applied for effectively solving real-world optimization problems that naturally arise in prescriptive analytics, such as network flow optimization, resource allocation, and scheduling. Solutions for various classes of mathematical programming problems can be obtained by deploying specialized, highly optimized solvers (e.g., [12, 1]). As an example, a warehouse supplies stores s with products i from its available inventory a_i. For each store s and product i there is a demand d_i^s and available inventory a_i^s. For each product i, let w_i denote the quantity of i in the warehouse, and p_i denote its unit price. A set of N trucks delivers commodity, and for simplicity assume that each truck makes a single warehouse-to-store trip. We need to determine the quantity q_i^s to ship to each store in order to maximize revenue. In standard MIP notation, the problem can be phrased as follows. Here, ψ_s is a binary variable (with values in $\{0,1\}$) determining whether a truck should be sent to store s, and m_i^s is the quantity of product i missing in order to satisfy the demand at store s.

$$\text{Maximize} \quad (\sum_{i,s} d_i^s - m_i^s) * p_i \ \ \text{subject to:}$$

$$\forall i,s \quad w_i^s + q_i^s + m_i^s \geq d_i^s\ , \ \ q_i^s \leq B * \psi_s\ , \ \sum_{s'} q_i^{s'} \leq a_i\ , \ \sum_{s'} \psi_{s'} \leq N$$

$$\forall i,s \quad m_i^s \geq 0\ , \ q_i^s \geq 0\ , \ \psi_s \in \{0,1\}$$

In this program, we assume that a store can hold at most 1000 units of each product. Observe that the unknown variables are the q_i^s, m_i^s and ψ_s (while the others are fixed).

MIP declaration in LogiQL is done by means of predicates with *free second-order variables*, which are essentially unknown functions over predefined domains, except that they are eventually assigned actual values (by invoking a MIP solver). Moreover, the objective function (that one wishes to minimize or maximize) is simply an attribute of a relation. Linear constraints are phrased as LogiQL constraints. Hence, a LogiQL program with MIP is an ordinary program, with the addition that attributes are marked as second-order variables or objectives. As an example, the following program corresponds to the above MIP specification. Here, the second-order variables are $\mathrm{m}[i,s]$, $\mathrm{q}[i,s]$ and $\mathrm{psi}[s]$, and the objective is $\mathrm{Obj}(v)$.

$$\begin{aligned}
\mathrm{Obj}(v) &\leftarrow \mathsf{agg}{\ll} v = \mathsf{sum}(z){\gg}\ z = (\mathrm{d}[i,s] - \mathrm{m}[i,s]) * \mathrm{p}[i] \\
\mathrm{Store}(s), \mathrm{Prod}(i) &\rightarrow \mathrm{w}[i,s] + \mathrm{q}[i,s] + \mathrm{m}[i,s] \geq \mathrm{d}[i,s] \\
\mathrm{TotalTrans}[i] = v &\leftarrow \mathsf{agg}{\ll} v = \mathsf{sum}(z){\gg}\ \mathrm{q}[i,s] = z \\
\mathrm{TotalTrans}[i] = v &\rightarrow v \leq \mathrm{a}[i] \\
\mathrm{q}[i,s] = v_1, \mathrm{psi}[s] = v_2 &\rightarrow v_1 \leq v_2 * 1000 \\
\mathrm{TotalTrucks}(v) &\leftarrow \mathsf{agg}{\ll} v = \mathsf{sum}(z){\gg}\ \mathrm{psi}[s] = z \\
\mathrm{TotalTrucks}(v) &\rightarrow v \leq \mathrm{AvailableTrucks}[] \\
\mathrm{m}[i,s] = v &\rightarrow v \geq 0\,, (\mathrm{psi}[s] = 0\ \mathsf{or}\ \mathrm{psi}[s] = 1)
\end{aligned}$$

Observe that the constraints are used in a fashion similar to the *stable-model* (or *answer-set*) semantics [8], except that we also optimize an objective. Other similar approaches include [5, 10, 16, 14]. As far as we know, LogicBlox is the first commercial database system to provide native support for prescriptive analytics.

The engine automatically synthesizes the necessary mathematical programming instances from the program, and invokes a MIP solver (e.g., [12, 1]). Specifically, we ground (i.e., eliminates quantifiers in) the problem instance in a manner similar to [15], and translate the constraints over variable predicates into a representation that can be consumed by the solver. Then, the solver output is used for populating the value of marked predicates (turning unknown values into known ones). The evaluation engine listens to updates in the relevant relations (as part of standard view maintenance), and invokes the solver when necessary to populate unknown values. The underlying MIP instance is constructed incrementally. For example, if a store demand changes, then only the portion of the program that is relevant to that store is replaced in the current MIP instance (before being re-sent to the solver to obtain an updated solution). We found that this optimization often leads to considerable reduction in execution cost.

3.2 Machine Learning (ML) Aggregates

The next extension is predictive analytics by means of a built-in set of ML algorithms. We use the special predict P2P rule that comes in two modes: *learning* mode (where a model is being learned) and *evaluation* mode (where a model is being applied to make predictions). We do not give here the formal syntax and semantics for these rules; rather, we give an illustrative example.

Suppose that we wish to predict the monthly sales of products in branches. We have the following predicates:

- $\mathrm{Hstr}[\mathrm{sku}, \mathrm{branch}, \mathrm{month}]{=}\mathrm{amount}$ contains historical sales per sku ("stock keeping unit") and branch;
- $\mathrm{Ftr}[\mathrm{sku}, \mathrm{branch}, \mathrm{name}]{=}\mathrm{value}$ associates with every sku, branch and feature name a corresponding feature value.

The following learning rule learns a logistic-regression model for each sku and branch, and stores the resulting *model object* (which is handle to a representation of the model) in the predicate $\mathrm{SM}[\mathrm{sales}, \mathrm{branch}] = \mathrm{model}$.

$$\mathrm{SM}[s, b] = m \leftarrow \mathsf{predict}{\ll} m = \mathsf{logist}(v|f){\gg}\ \mathrm{Hstr}[s, b, t] = v, \mathrm{Ftr}[s, b, t, n] = f$$

And the following evaluation rule evaluates the model to get specific predictions.

$$\begin{gathered}\mathrm{Sales}[s, b, t] = v \leftarrow \mathsf{predict}{\ll} v = \mathsf{eval}(m|f){\gg}\\ \mathrm{Unknown}(s, b, t), \mathrm{SM}[s, b] = m, \mathrm{Ftr}[s, b, t, n] = f\end{gathered}$$

3.3 Statistical Relational Models

Such models are specified by various mechanisms, including Markov Logic Networks (MLN) [6] and Probabilistic Soft Logic (PSL) [4]. MLNs and PLSs are, intuitively, logical formalisms that allow for *soft rules*. The canonical example is

$$R(x) \leftarrow R(y)\ ,\ \mathrm{Friends}(x, y)$$

where R describes a person property (e.g., "smokes" or "votes"). This rule states that whenever x and y are friends, the property R propagates from y to x; this rule should be taken as a hint on the unknown, and not as rigid truth. While an ordinary Datalog program specifies a unique extension of the database, soft rules specify a probability space over such extensions. Intuitively, the probability of a possible extension is determined by the extent to which the rules are satisfied (where weights of rules are taken into account). A common practice is to find the *most likely* extension (i.e. Maximum A-Priori, or MAP, inference), such as the most likely votes given partial knowledge about votes, and use that world as an ordinary database. We make an ongoing effort to support MLN and PSL within LogicBlox. Our current implementation applies MAP inference by translation into MIP. In future work we plan to include specialized algorithms to reduce the execution cost.

3.4 Probabilistic Programming Datalog (PPDL)

Formalisms for specifying general statistical models, such as probabilistic-programming languages [9], typically consist of two components: a specification of a stochastic process (the prior), and a specification of observations that restrict the probability space to a conditional subspace (the posterior). We plan to enhance LogiQL with capabilities of probabilistic programming, in order to facilitate the design and engineering of ML solutions. Towards that, we have initiated a theoretical exploration of such an extension. In a recent paper [3], we have proposed *Probabilistic Programming Datalog* (*PPDL*), which is a framework that extends LogiQL with convenient mechanisms to include common numerical probability functions; in particular, conclusions of rules may contain values drawn from such functions. As a (simplistic) example, assume the relation $\mathrm{Client}(\mathrm{ssn}, \mathrm{branch}, \#\mathrm{visits})$ that associates clients with social security numbers, local branches, and an average number of visits (per month) in the branch. The rule

$$\mathrm{Visits}(c, b, \mathrm{Poisson}[\lambda]) \leftarrow \mathrm{Client}(c, b, \lambda)$$

associates with the client a random number of visits in the branch, where that number is drawn from the Poisson distribution with average (parameter) $\#\mathrm{visits}$.

The semantics of a program is a probability distribution over the possible outcomes of the input database with respect to the program; these possible outcomes are minimal solutions with respect to a related program that involves existentially quantified variables in conclusions. Observations are naturally incorporated by means of constraints. We focused on discrete numerical distributions (such as Poisson), but even then the space of possible outcomes may be uncountable (as a solution can be infinite). We defined a probability measure over possible outcomes by applying the known concept of *cylinder sets* [2] to a probabilistic chase procedure. This chase is similar to that of data exchange with *tuple-generating dependencies* [7], except that instead of introducing named nulls, we sample real values from the associated distribution (e.g., $\mathsf{Poisson}[5]$). We have shown that the resulting semantics is invariant under different chases.

4 Conclusions

We described four approaches taken by LogicBlox to extend LogiQL with built-in analytics. While MIP and ML aggregates are conceptually syntactic bridges between Datalog and external solvers, statistical relational models, and PPDL feature stronger ties to Datalog (and naturally require more in-house implementation effort). In future work we plan to investigate the applicability of our statistical specifications to real-life problems that arise in our business, as well as their theoretical and system aspects.

References

1. T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1), 2009. http://mpc.zib.de/index.php/MPC/article/view/4.
2. R. B. Ash and C. Doleans-Dade. *Probability & Measure Theory*. Harcourt Academic Press, 2000.
3. V. Barany, B. t. Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative statistical modeling with datalog. *arXiv preprint arXiv:1412.2221*, 2014.
4. M. Bröcheler, L. Mihalkova, and L. Getoor. Probabilistic similarity logic. In *UAI*, 2010.
5. M. Cadoli, G. Ianni, L. Palopoli, A. Schaerf, and D. Vasile. Np-spec: an executable specification language for solving all problems in np. *Computer Languages*, 26(2):165–195, 2000.
6. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on AI and Machine Learning. Morgan & Claypool Publishers, 2009.
7. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, volume 2572 of *LNCS*. Springer, 2003.

8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988 (2 Volumes)*, pages 1070–1080. MIT Press, 1988.
9. N. D. Goodman. The principles and practice of probabilistic programming. In *POPL*, 2013.
10. S. Greco, C. Molinaro, I. Trubitsyna, and E. Zumpano. NP datalog: A logic language for expressing search and optimization problems. *TPLP*, 10(2):125–166, 2010.
11. T. J. Green, M. Aref, and G. Karvounarakis. LogicBlox, platform and language: A tutorial. In *Int Conf on Datalog in Academia and Industry*, 2012.
12. I. Gurobi Optimization. Gurobi optimizer reference manual, 2015.
13. T. Halpin and S. Rugaber. *LogiQL: A Query Language for Smart Databases*. CRC Press, 2014.
14. A. Meliou and D. Suciu. Tiresias: The database oracle for how-to queries. In *SIGMOD*, pages 337–348, 2012.
15. F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in markov logic networks using an rdbms. *PVLDB*, 4(6):373–384, 2011.
16. T. E. Sheard. Painless programming combining reduction and search: Design principles for embedding decision procedures in high-level languages. In *ICFP*, pages 89–102, 2012.

Towards Reconciling SPARQL and Certain Answers (Extended Abstract)[*]

Shqiponja Ahmetaj, Wolfgang Fischl, Reinhard Pichler, Mantas Šimkus, and Sebastian Skritek

Institute of Information Systems, TU Vienna, Austria
{ahmetaj,wfischl,pichler,simkus,skritek}@dbai.tuwien.ac.at

Abstract. SPARQL entailment regimes are strongly influenced by the big body of works on ontology-based query answering, notably in the area of Description Logics (DLs). However, the semantics of query answering under SPARQL entailment regimes is defined in a more naive and much less expressive way than the certain answer semantics usually adopted in database and DL literature. In this paper we introduce an intuitive certain answer semantics also for SPARQL and show the feasibility of this approach. For OWL 2 QL entailment, we develop algorithms for the evaluation of an interesting fragment of SPARQL (the so-called well-designed SPARQL). Exploiting these algorithms, we can show that the complexity of neither query answering nor the most fundamental query analysis tasks (such as query containment and equivalence testing) is negatively affected by the presence of OWL 2 QL entailment under the proposed semantics.

1 Introduction

In the recently released recommendation [7], the W3C has defined various SPARQL entailment regimes to allow users to specify implicit knowledge about the vocabulary in an RDF graph. The theoretical underpinning to the systems for query answering under rich entailment regimes is provided by the big body of work on ontology-based query answering, notably in the area of Description Logics (DLs) [4]. However, the semantics of query answering under SPARQL entailment regimes is defined in a more naive and much less expressive way than the certain answer semantics usually adopted in the DL and database literature.

Example 1. Consider an RDF graph G containing a single triple $(b, \mathsf{a}, \mathsf{Prof})$ – stating that b is a professor – and an ontology $\mathcal{O}$ containing the triples

$$(\mathsf{Prof}, \mathtt{rdfs{:}sc}, \mathtt{\_{:}b}), (\mathtt{\_{:}b}, \mathsf{a}, \mathtt{owl{:}Restriction}),$$
$$(\mathtt{\_{:}b}, \mathtt{owl{:}onProperty}, \mathsf{teaches}), (\mathtt{\_{:}b}, \mathtt{owl{:}someValuesFrom}, \mathtt{owl{:}Thing}).$$

[*] A longer version of this paper was presented at WWW 2015 [1].

– stating that every professor teaches somebody. Now consider the following simple SPARQL query: SELECT $?x$ WHERE $(?x, \mathsf{teaches}, ?y)$.[1] Following the SPARQL entailment regimes standard [7], this query yields an empty result. □

This result is rather unintuitive: by the inclusion we know for certain that b teaches somebody. However, the SPARQL entailment standard requires that all values assigned to any variable must come from the RDF graph – thus treating distinguished variables (which are ultimately output) and non-distinguished variables (which are eventually projected out) in the same way. In contrast, the certain answer semantics retrieves all mappings on the distinguished variables that allow to satisfy the query in every possible model of the database and the ontology (yielding the certain answer $\mu = \{?x \to b\}$ in the above example).

The **goal of this work** is to introduce an intuitive certain answer semantics also for SPARQL under OWL 2 QL entailment with similarly favorable results as for CQ answering under DL-Lite$_{\mathcal{R}}$ (which provides the theoretical underpinning of the OWL 2 QL entailment regime).

The reason why for this purpose we cannot simply take over all the results from CQ answering under DL-Lite is that SPARQL provides some crucial extensions over CQs. One of them is the OPTIONAL operator (henceforth referred to as OPT operator, for short). It allows the user to retrieve *partial solutions* in cases where no match for the complete query can be found, instead of failing to provide any solution. Observe that these queries are no longer monotone. Thus, the usual certain answer semantics (i.e., something is a certain answer if it is present in every model) turns out to be unsatisfactory:

Example 2. Consider the SPARQL query: SELECT $?x, ?z$ WHERE $(?x, \mathsf{teaches}, ?y)$ OPT $(?y, \mathsf{knows}, ?z)$ over the graph $G = \{(b, \mathsf{teaches}, c)\}$ and empty ontology $\mathcal{O}$. The query yields a unique solution $\mu = \{?x \to b\}$. Clearly, also the extended graph $G' = G \cup \{(c, \mathsf{knows}, d)\}$ is a model of $(G, \mathcal{O})$. But in G', μ is no longer a solution since μ can be extended to solution $\mu' = \{?x \to b, ?z \to d\}$. Hence, there exists no mapping which is a solution in every possible model of $(G, \mathcal{O})$. □

In this paper, we discuss further problems with a literal adoption of a certain answer semantics in the presence of the OPT operator, and propose a suitable modified definition for the class of *well-designed* SPARQL queries [11]. This modified semantics also requires an adaptation and extension of the known query answering algorithms for DL-Lite. We present two such modified algorithms for query evaluation. Finally, we shall show that the additional expressive power due to the certain answers comes without an increase of the complexity.

Related Work. For our findings the following work is most relevant to us: the semantics of SPARQL was investigated in [3], which also introduces weakly-monotone queries, i.e. well-designed SPARQL. The semantics for SPARQL over

[1] Following [11], we use a more algebraic style notation, denoting triples in parentheses with comma-separated components, rather than the blank-separated turtle notation.

OWL ontologies is standardized by the World Wide Web consortium in [7]. Our two algorithms are based upon the standard rewriting algorithm for *DL-Lite* [5] and a more advanced algorithm for the DL Horn-$\mathcal{SHIQ}$ [6]. There is a huge body of results on CQ answering under different DLs (cf. [5, 6, 10, 12]). For SPARQL recent work [8] presents a *stronger* semantics, where entire mappings are discarded, whose possible extensions to optional subqueries would imply inconsistencies in the knowledge base. In [2], the authors describe a rewriting of SPARQL query answering under OWL 2 QL into Datalog$^{\pm}$. A slight modification allows them to remove the active domain semantics of variables, however this only applies to variables occuring in a single BGP. Libkin [9] also criticizes the standard notion of certain answers in case of non-monotone queries. Similar to his suggestion to use the greatest lower bounds in terms of informativeness, our approach chooses the most informative solutions as certain answers.

2 SPARQL and OWL 2 QL

OWL 2 QL is based on DL-Lite$_{\mathcal{R}}$, a lightweight description logic. Its fundamental building blocks are *constants* c, *atomic concepts* A and *atomic roles* R, which are countably infinite and mutually disjoint subsets of a set $\mathbf{U}$ of URIs. From these we can build *basic roles* R and R^-, and *basic concepts* B and $\exists Q$, where Q is a basic role. Using the above, DL-Lite$_{\mathcal{R}}$ allows one to express the following kind of statements: Membership assertions (c, a, B) or (c, Q, c'), concept inclusions $(B_1, \texttt{rdfs:sc}, B_2)$, role inclusions $(Q_1, \texttt{rdfs:sp}, Q_2)$ as well as concept and role disjointness (where c, c' are constants and B_i, Q_i are basic concepts resp. basic roles). In the following, an ontology $\mathcal{O}$ is any set of such expressions, excluding membership assertions, which we assume to be part of the RDF graph. A *knowledge base (KB)* $\mathcal{G} = (G, \mathcal{O})$ consists of an RDF graph G and an ontology $\mathcal{O}$.

The basic building block of SPARQL queries are *triple patterns* $(s, p, o) \in (\mathbf{U} \cup \mathbf{V})^3$, where $\mathbf{V}$ is a set of variables. In this work we only consider triple patterns of the form $(?x, \mathsf{a}, B)$ or $(?x, Q, ?y)$ where B (Q) is a basic concept (role). More complex *graph patterns* are built from triple patterns via operators like e.g. AND, OPT, or UNION. Here, we consider a SPARQL query to be a graph pattern, possibly extended by top-level projection. Given a graph pattern P, a set $\mathcal{X} \subseteq \mathbf{V}$ of variables occurring in P and an RDF graph G, the answer $[\![(P, \mathcal{X})]\!]_G$ to P, projected to $\mathcal{X}$, over G is a set of partial mappings from $\mathcal{X}$ to $\mathbf{U}$. We say a mapping μ_1 is subsumed by another mapping μ_2, denoted by $\mu_1 \sqsubseteq \mu_2$, if $\mathsf{dom}(\mu_1) \subseteq \mathsf{dom}(\mu_2)$ and $\mu_1(?x) = \mu_2(?x)$ for all $?x \in \mathsf{dom}(\mu_1)$, where $\mathsf{dom}(\mu_i)$ denotes the set of variables the mapping μ_i is defined on.

By imposing certain restrictions on the occurrence of variables, the fragment of *well-designed SPARQL (wdSPARQL)* was introduced in [11]. It possesses several desirable properties, like coNP-completeness of query evaluation. Of importance for our work is that these queries are *weakly-monotone* [3]: If $\mu \in [\![(P, \mathcal{X})]\!]_G$, then for every RDF graph G' with $G \subseteq G'$, there exists $\mu' \in [\![(P, \mathcal{X})]\!]_{G'}$ s.t. $\mu \sqsubseteq \mu'$ (i.e., while μ need not be a solution over G', it can be extended to one).

3 Certain Answers of well-designed SPARQL

Before providing our definition of certain answers, we need to introduce two additional notions. Let P be a well-designed graph pattern. Following [11], we say that P' is a reduction of P (denoted as $P' \trianglelefteq P$) if P' can be constructed from P by replacing in P sub-patterns of the form (P_1 OPT P_2) by P_1. Second, for a mapping μ and some property A, we shall say that μ is *$\sqsubseteq$-maximal w.r.t. A* if μ satisfies A, and there is no μ' such that $\mu \sqsubseteq \mu'$, $\mu' \not\sqsubseteq \mu$, and μ' satisfies A.

Definition 1. *Let $\mathcal{G} = (G, \mathcal{O})$ be a KB and $Q = (P, \mathcal{X})$ a well-designed query. A mapping μ is a* certain answer *to Q over $\mathcal{G}$ if it is a $\sqsubseteq$-maximal mapping s.t. (1) $\mu \sqsubseteq [\![Q]\!]_{G'}$ for every model G' of $\mathcal{G}$, and (2) $\mathsf{vars}(P') \cap \mathcal{X} = \mathsf{dom}(\mu)$ for some $P' \trianglelefteq P$. We denote by $\mathsf{cert}(P, \mathcal{X}, \mathcal{G})$ the set of all certain answers to Q over $\mathcal{G}$.*

The reason for restricting the set of certain answers to $\sqsubseteq$-maximal mappings is that queries with projection and/or UNION may have “subsumed” solutions, i.e. solutions s.t. also some proper extension is a solution. But then – with set semantics – we cannot recognize the reason why some subsumed solution is possibly not a solution in some possible world, as illustrated in Example 3. Since in our first step towards reconciling SPARQL and certain answers we decide to stick to set semantics, we allow only “maximal” solutions as certain answers.

Example 3. Consider the following query SELECT $?x, ?z$ WHERE $(?x, \mathsf{teaches}, ?y)$ OPT $(?y, \mathsf{knows}, ?z)$ over the graph $G = \{(a, \mathsf{teaches}, b), (b, \mathsf{knows}, c), (a, \mathsf{teaches}, d)\}$ and empty ontology $\mathcal{O}$. As possible models of $(G, \mathcal{O})$ we have all graphs containing G. Hence, $\mu = \{?x \to a, ?z \to c\}$ and $\mu' = \{?x \to a\}$ ($?y$ is bound to d) are both answers to G and can be extended to solutions in every possible model.

Next consider $G' = \{(a, \mathsf{teaches}, b), (b, \mathsf{knows}, c)\}$. If we take as certain answers all mappings that can be extended to some solution in every possible model, then μ' from above is still a certain answer. □

Property (2) in the definition of certain answers ensures that the domain of such an answer adheres to the structure of nested OPTs in the query. However, we can show that this property need not be considered during the computation of the certain answers, but can be enforced in a simple post-processing step. We call such answers that satisfy Definition 1 except property (2) *certain pre-answers*, and use $\mathsf{certp}(P, \mathcal{X}, \mathcal{G})$ to denote the set of all certain pre-answers. The same is also true for projection, which can also be performed in a simple post-processing step. Thus, it suffices to compute $\mathsf{certp}(P, \mathcal{G})$, which can be done via universal solutions (referred to as *canonical model* in the area of DLs) as follows.

Theorem 1. *Let $\mathcal{G} = (G, \mathcal{O})$ be a KB and P a well-designed graph pattern. Then, $\mathsf{certp}(P, \mathcal{G}) = \mathrm{MAX}([\![P]\!]_{\mathsf{univ}(G)}\downarrow)$, where $\mathrm{MAX}(M)$ is the set of $\sqsubseteq$-maximal mappings in M, $M\downarrow := \{\mu\downarrow \mid \mu \in M\}$ ($\mu\downarrow$ is the restriction of μ to those variables mapped to the active domain of G), and $\mathsf{univ}(G)$ is a universal solution of $\mathcal{G}$.*

However, computing the certain answers via a universal solution is not always practical, e.g. the universal solution can be infinite. As a result, query rewriting algorithms have been developed: These algorithms take the input query and the ontology, and rewrite them into a single query that can be evaluated over the input database without considering the ontology. By introducing several adaptations and extensions of the rewriting-based CQ evaluation for DL-Lite from [5], we develop two different approaches to answer well-designed SPARQL queries under OWL 2 QL entailment.

The first one proceeds in a modular way by rewriting basic building blocks of a SPARQL query (so-called BGPs) individually. It thus follows the general philosophy of SPARQL entailment regimes. One possible disadvantage of this modular approach is that it requires to maintain additional data structures to ensure consistency when combining the partial solutions for different BGPs. As a consequence, the complete algorithm has to be implemented from scratch because the standard tools cannot handle these additional data structures.

The goal of the second approach is thus to make use of the standard technology as much as possible. The idea is to transform the OWL 2 QL entailment under our new semantics into SPARQL query evaluation under RDFS entailment, for which strong tools are available. Unlike the first – modular – approach, this rewriting proceeds in a holistic way, i.e. it always operates on the whole query.

Based on these rewriting algorithms, we analyze the complexity of query answering and of several static query analyzing tasks such as query containment and equivalence. We are able to show that the additional power of our new semantics comes without additional costs in terms of complexity.

Acknowledgements

This work was supported by the Vienna Science and Technology Fund (WWTF), project ICT12-15 and by the Austrian Science Fund (FWF): P25207-N23.

References

1. S. Ahmetaj, W. Fischl, R. Pichler, M. Šimkus, and S. Skritek. Towards reconciling SPARQL and certain answers. In *Proc. of WWW 2015*, 2014.
2. M. Arenas, G. Gottlob, and A. Pieris. Expressive languages for querying the semantic web. In *Proc. of PODS 2014*, pages 14–26. ACM, 2014.
3. M. Arenas and J. Pérez. Querying semantic web data with SPARQL. In *Proc. of PODS 2011*, pages 305–316. ACM, 2011.
4. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.
5. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
6. T. Eiter, M. Ortiz, M. Šimkus, T. Tran, and G. Xiao. Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules. In *Proc. of AAAI 2012.* AAAI Press, 2012.
7. B. Glimm and C. Ogbuji. SPARQL 1.1 Entailment Regimes. W3C Recommendation, W3C, Mar. 2013. `http://www.w3.org/TR/sparql11-entailment`.

8. E. V. Kostylev and B. Cuenca Grau. On the semantics of SPARQL queries with optional matching under entailment regimes. In *Proc. of ISWC 2014*, 2014.
9. L. Libkin. Incomplete data: what went wrong, and how to fix it. In *Proc. of PODS 2014*, pages 1–13. ACM, 2014.
10. M. Ortiz, D. Calvanese, and T. Eiter. Data complexity of query answering in expressive description logics via tableaux. *Journal of Automated Reasoning*, 41(1):61–98, 2008.
11. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.
12. R. Rosati. On conjunctive query answering in EL. In *Proc. of DL 2007*, 2007.

Efficient Evaluation of Well-designed Pattern Trees (Extended Abstract)[⋆]

Pablo Barceló[1], Reinhard Pichler[2], and Sebastian Skritek[2]

[1] Center for Semantic Web Research & Department of Computer Science, University of Chile
[2] Faculty of Informatics, Vienna University of Technology

Abstract. Conjunctive queries (CQs) constitute the core of the query languages for relational databases and also the most intensively studied querying mechanism in the database theory community. But CQs suffer from a serious drawback when dealing with incomplete information: If it is not possible to match the complete query with the data, they return no answer at all. The semantic web therefore provides a formalism - known as well-designed pattern trees (WDPTs) - that tackles this problem. In particular, WDPTs allow us to match patterns over the data if available, but do not fail to give an answer otherwise. Here, we abstract away the specifics of semantic web applications and study WDPTs over arbitrary relational schemas. Since our language properly subsumes the class of CQs, the evaluation problem associated with it is intractable. In this paper we identify natural structural properties of WDPTs that lead to tractability of various variants of the evaluation problem.

1 Introduction

Conjunctive queries (CQs) constitute the core of the query languages for relational databases and also the most intensively studied querying mechanism in the database theory community. But CQs suffer from a serious drawback when dealing with incomplete information: they fail to provide an answer when the pattern described by the query cannot be matched completely into the data.

The semantic web therefore provides formalisms to overcome this problem. One simple such formalism corresponds to the $\{\mathsf{AND},\mathsf{OPT}\}$-fragment of SPARQL – the standard query language for RDF, the semantic web data model. The OPT-operator extends the AND-operator by the possibility to return partial answers. I.e., instead of returning no answer at all if not the complete query can be matched into the data, it allows to match parts of the query. Pérez et al. noticed that a non-constrained interaction of these two operators may lead to undesired behavior [11]. This motivated the definition of a better behaved syntactic restriction of the language, known as *well-designed* $\{\mathsf{AND},\mathsf{OPT}\}$-SPARQL. Queries in this fragment allow for a natural tree representation, called *well-designed pattern trees (WDPTs)* [10]. Here, we abstract away from the specifics of RDF and define WDPTs over arbitrary relational schemas.

Despite the importance of WDPTs, very little is known about some fundamental problems related to them. In particular, no in-depth study has been carried out regarding

[⋆] This is an extended abstract of [3]

efficient evaluation of these queries, a problem that permeates the literature on CQs and its extensions [13, 8, 9]. The main goal of this work is to initiate a systematic study of tractable fragments of WDPTs for the different variants of query evaluation that have been studied in the literature. We explain this in more detail below.

2 Well-designed Pattern Trees

We define the class of WDPTs below. Intuitively, the nodes of a WDPT represent CQs (called "basic graph patterns" in the semantic web context) while the nesting of optional matching is represented by the tree structure of a WDPT.

A *well-designed pattern tree* (WDPT) over schema σ is a pair $(T, \lambda, \bar{x})$, such that:

1. T is a rooted tree and λ maps each node t in T to a set of relational atoms over σ.
2. For every variable y mentioned in T, the set of nodes where y occurs is connected.
3. The tuple $\bar{x}$ of distinct variables from T denotes the *free variables* of the WDPT.

We say that $(T, \lambda, \bar{x})$ is *projection-free*, if $\bar{x}$ contains all variables mentioned in T.

Assume $p = (T, \lambda, \bar{x})$ is a WDPT over σ. We write r to denote the root of T. Given a subtree T' of T rooted in r, we define $q_{T'}$ to be the CQ $\text{Ans}(\bar{y}) \leftarrow R_1(\bar{v}_1), \ldots, R_m(\bar{v}_m)$, where $\{R_1(\bar{v}_1), \ldots, R_m(\bar{v}_m)\} = \bigcup_{t \in T'} \lambda(t)$, and $\bar{y}$ are all the variables that are mentioned in T'. That is, all variables in $q_{T'}$ appear free.

We define the semantics of WDPTs by naturally extending their interpretation under semantic web vocabularies [10]. The intuition behind the semantics of a WDPT $(T, \lambda, \bar{x})$ is as follows. A mapping h is an answer to (T, λ) over a database $\mathcal{D}$, if it is "maximal" among the mappings that satisfy the patterns $q_{T'}$ defined by the subtrees T' of T. This means, h is a solution to $q_{T'}$ and there is no way to "extend" h to a solution of some $q_{T''}$ for some bigger subtree T'' of T. The evaluation of a WDPT $(T, \lambda, \bar{x})$ over $\mathcal{D}$ corresponds then to the projection over the variables in $\bar{x}$ of the mappings h that satisfy (T, λ) over $\mathcal{D}$. We formalize this next: Let $\mathcal{D}$ be a database and $p = (T, \lambda, \bar{x})$ a WDPT over schema σ. Assume that $dom(\mathcal{D})$ is the set of elements in the active domain of $\mathcal{D}$ and $\mathbf{X}$ are the variables mentioned in p. Then:

- A *homomorphism* from p to $\mathcal{D}$ is a partial mapping $h : \mathbf{X} \to dom(\mathcal{D})$, for which it is the case that there is a subtree T' of T rooted in r such that h is a *homomorphism* from the CQ $q_{T'}$ to $\mathcal{D}$.
- The homomorphism h is *maximal* if there is no homomorphism h' from p to $\mathcal{D}$ such that h' *extends* h.

If h is a homomorphism from $p = (T, \lambda, \bar{x})$ to $\mathcal{D}$ we denote by $h_{\bar{x}}$ the restriction of h to the variables in $\bar{x}$. The *evaluation* of p over $\mathcal{D}$, denoted $p(\mathcal{D})$, corresponds to all mappings of the form $h_{\bar{x}}$, such that h is a maximal homomorphism from p to $\mathcal{D}$.

Notice that WDPTs properly extend CQs. In fact, assume $q(\bar{x})$ is a CQ of the form $\text{Ans}(\bar{x}) \leftarrow R_1(\bar{v}_1), \ldots, R_m(\bar{v}_m)$. Then $q(\bar{x})$ is equivalent to WDPT $p = (T, \lambda, \bar{x})$, where T consists of a single node r and $\lambda(r) = \{R_1(\bar{v}_1), \ldots, R_m(\bar{v}_m)\}$. In other words, $q(\mathcal{D}) = p(\mathcal{D})$, for each database $\mathcal{D}$.

3 Efficient evaluation of WDPTs

3.1 Evaluation of WDPTs:

We study the complexity of the evaluation problem $\textsc{Eval}(\mathcal{C})$ for different classes $\mathcal{C}$ of WDPTs. This problem is formally defined as follows: Given a database $\mathcal{D}$ and a WDPT p over σ, as well as a partial mapping $h : \mathbf{X} \to dom(\mathcal{D})$, where $\mathbf{X}$ is the set of variables mentioned in p, is it the case that h belongs to $p(\mathcal{D})$?

The complexity of $\textsc{Eval}(\mathcal{C})$ has been studied for the case when $\mathcal{C}$ is the class $\mathcal{C}_{\mathsf{all}}$ of all WDPTs or the class $\mathcal{C}_{\mathsf{pf}}$ of projection-free WDPTs. In particular, $\textsc{Eval}(\mathcal{C}_{\mathsf{all}})$ is Σ_2^P-complete [10] and $\textsc{Eval}(\mathcal{C}_{\mathsf{pf}})$ is coNP-complete [11]. This raises the need for understanding which classes of WDPTs can be evaluated in polynomial time.

Evaluation of WDPTs is defined in terms of CQ evaluation, which is an intractable problem in general. Therefore, our goal of identifying tractable classes of WDPTs naturally calls for a restriction of the classes of CQ patterns allowed in them. In particular, there has been a flurry of activity around the topic of determining which classes of CQs admit efficient evaluation that could be reused in our scenario [13, 8, 9]. These include classes of bounded *treewidth* [6], *hypertreewidth* [9], etc. We concentrate on the first two. From now on, we denote by $\mathsf{TW}(k)$ (resp., $\mathsf{HW}(k)$), for $k \geq 1$, the class of CQs of treewidth (resp., hypertreewidth) at most k.

A condition that has been shown to help identifying relevant tractable fragments of WDPTs is *local tractability* [10]. This refers to restricting the CQ defined by each node in a WDPT to belong to a tractable class. Formally, let $\mathcal{C}$ be either $\mathsf{TW}(k)$ or $\mathsf{HW}(k)$, for $k \geq 1$. A WDPT $(T, \lambda, \bar{x})$ is *locally in* $\mathcal{C}$, if for each node $t \in T$ such that $\lambda(t) = \{R_1(\bar{v}_1), \ldots, R_m(\bar{v}_m)\}$ the CQ $\text{Ans}() \leftarrow R_1(\bar{v}_1), \ldots, R_m(\bar{v}_m)$ is in $\mathcal{C}$. We write ℓ-$\mathcal{C}$ for the set of all WDPTs that are locally in $\mathcal{C}$.

It is known that local tractability leads to tractability of evaluation for projection-free WDPTs [10]. On the other hand, this result does not hold in the presence of projection, even when $\mathcal{C}$ is of bounded treewidth. Formally, $\textsc{Eval}$*(*ℓ-$\mathsf{TW}(k)$*)* and $\textsc{Eval}$*(*ℓ-$\mathsf{HW}(k)$*)* are NP-complete for every $k \geq 1$ [10].

This raises the question of which further restrictions on WDPTs are needed to achieve tractability. Here we identify a natural such restriction, called *bounded interface*. Intuitively, this restricts the number of variables shared between a node in a WDPT and its children. Formally, a WDPT $(T, \lambda, \bar{x})$ has *c-bounded interface*, for $c \geq 1$, if for each node $t \in T$ with children $t_1, \ldots, t_k$ it is the case that the number of variables that appear both in a relational atom in $\lambda(t)$ and in a relational atom in $\lambda(t_i)$, for some $1 \leq i \leq k$, is at most c. We denote by $\mathsf{BI}(c)$ the set of WDPTs of c-bounded interface. Interestingly, similar restrictions on the number of variables shared by different atoms of CQs have been recently applied for obtaining reasonable bounds for the problem of containment of Datalog into unions of CQs [5]. One of our main results states that local tractability and bounded interface yield tractability of WDPT evaluation:

Theorem 1. *Let $\mathcal{C}$ be $\mathsf{TW}(k)$ or $\mathsf{HW}(k)$ and $c \geq 1$. Then $\textsc{Eval}(\ell\text{-}\mathcal{C} \cap \mathsf{BI}(c))$ is in* PTIME.

Notice that CQs are a special case of WDPTs consisting of the root node only. Hence, $\mathsf{TW}(k) \subseteq \ell\text{-}\mathsf{TW}(k) \cap \mathsf{BI}(c)$ and $\mathsf{HW}(k) \subseteq \ell\text{-}\mathsf{HW}(k) \cap \mathsf{BI}(c)$ hold for each $c \geq 1$.

Therefore, Theorem 1 tells us that ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$ and ℓ-$\mathsf{HW}(k) \cap \mathsf{BI}(c)$ define relevant extensions of $\mathsf{TW}(k)$ and $\mathsf{HW}(k)$, respectively, that preserve tractability of evaluation.

3.2 Partial evaluation of WDPTs:

Given the nature of WDPTs, it is also interesting to check whether a mapping h is a *partial* answer to the WDPT p over $\mathcal{D}$ [11, 1], i.e., whether h can be extended to some answer h' to p over $\mathcal{D}$. This gives rise to the partial evaluation problem PARTIAL-EVAL($\mathcal{C}$) for a class $\mathcal{C}$ of WDPTs defined as follows: Given a database $\mathcal{D}$ and a WDPT $p \in \mathcal{C}$ over σ, as well as a partial mapping $h : \mathbf{X} \to \mathbf{U}$, where $\mathbf{X}$ is the set of variables mentioned in p, does there exists some $h' \in p(\mathcal{D})$ such that h' extends h?

If projection is allowed, then partial evaluation is NP-complete even under local tractability, i.e., even for the classes ℓ-$\mathsf{TW}(k)$ and ℓ-$\mathsf{HW}(k)$, for each $k \geq 1$ [10].

It is easy to modify the proof of Theorem 1 to show that adding bounded interface to local tractability yields efficient partial evaluation; that is, PARTIAL-EVAL(ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c)$) and PARTIAL-EVAL($\ell$-$\mathsf{HW}(k) \cap \mathsf{BI}(c)$) are in PTIME. However, partial evaluation is seemingly easier than exact evaluation. Hence, the question naturally arises if tractability of partial evaluation of WDPTs can be ensured by a weaker condition. Indeed, we give a positive answer to this question below. This condition will be referred to as *global tractability*. Intuitively, it states that there is a bound on the treewidth (resp., hypertreewidth) of the CQs defined by the different subtrees of a WDPT $(T, \lambda, \bar{x})$ rooted in r. Formally, let $\mathcal{C}$ be $\mathsf{TW}(k)$ or $\mathsf{HW}(k)$, for $k \geq 1$. A WDPT $(T, \lambda, \bar{x})$ is *globally in* $\mathcal{C}$, if for each subtree T' of T rooted in r it is the case that the CQ $q_{T'}$ is in $\mathcal{C}$. We denote by g-$\mathcal{C}$ the set of all WDPTs that are globally in $\mathcal{C}$.

The following proposition formally states that global tractability is a strictly weaker condition than the conjunction of local tractability and bounded interface.

Proposition 1. *1. Let $k, c \geq 1$. Then ℓ-$\mathsf{TW}(k) \cap \mathsf{BI}(c) \subseteq g$-$\mathsf{TW}(k + 2c)$ and ℓ-$\mathsf{HW}(k) \cap \mathsf{BI}(c) \subseteq g$-$\mathsf{HW}(k + 2c)$.*
2. For every $k \geq 1$ there is a family C_k of WDPTs in g-$\mathsf{TW}(k)$ (resp., in g-$\mathsf{HW}(k)$) such that $C_k \not\subseteq \mathsf{BI}(c)$, for each $c \geq 1$.

We can now formally state that global tractability leads to tractability of the partial evaluation problem for WDPTs:

Theorem 2. PARTIAL-EVAL(g-$\mathsf{TW}(k)$) *and* PARTIAL-EVAL(g-$\mathsf{HW}(k)$) *are in* PTIME *for every $k \geq 1$.*

It remains to answer the question if global tractability also suffices to ensure tractability of (exact) evaluation for WDPTs. It turns out that this is not the case.

Proposition 2. EVAL(g-$\mathsf{TW}(k)$) *and* EVAL(g-$\mathsf{HW}(k)$) *are* NP-*complete for all $k \geq 1$.*

3.3 Semantics based on maximal mappings:

The semantics of projection-free WDPTs is only based on *maximal* mappings, i.e., mappings that are not subsumed by any other mapping in the answer. This is no longer the case in the presence of projection [10]. Recent work on query answering for SPARQL under entailment regimes has established the need for a semantics for WDPTs that is uniquely based on maximal mappings [1]. This semantics is formalized as follows. Assume $\mathcal{D}$ is a database and p is a WDPT over σ. The *evaluation of p over $\mathcal{D}$ under maximal mappings*, denoted $p_m(\mathcal{D})$, corresponds to the restriction of $p(\mathcal{D})$ to those mappings $h \in p(\mathcal{D})$ that are not extended by any other mapping $h' \in p(\mathcal{D})$. This naturally leads to the decision problem MAX-EVAL($\mathcal{C}$) defined as follows: Given a database $\mathcal{D}$ and a WDPT $p \in \mathcal{C}$ over σ, as well as a partial mapping $h : \mathbf{X} \to \mathbf{U}$, where $\mathbf{X}$ is the set of variables mentioned in p, is $h \in p_m(\mathcal{D})$?

It follows from [1] that MAX-EVAL($\mathcal{C}$) is clearly intractable for the class $\mathcal{C}$ of all WDPTs. Analogously to PARTIAL-EVAL, local tractability is not sufficient to ensure tractability of MAX-EVAL:

Proposition 3. *For every $k \geq 1$ the problems* MAX-EVAL(ℓ-$\mathsf{TW}(k)$) *and* MAX-EVAL(ℓ-$\mathsf{HW}(k)$) *are* NP*-hard.*

To obtain tractability in this case it is however sufficient to impose global tractability, which is exactly the same condition that yields tractability of partial evaluation for WDPTs (as stated in Theorem 2):

Theorem 3. MAX-EVAL(g-$\mathsf{TW}(k)$) *and* MAX-EVAL(g-$\mathsf{HW}(k)$) *are in* PTIME *for every $k \geq 1$.*

4 Further Results

Taking these results as a starting point, we were also able to show that in several cases the complexity of static analysis tasks, like deciding containment and equivalence [12], decreases. Next, we also studied the problem of testing if some WDPT is equivalent to one from a tractable class (cf. e.g. [2, 4, 7]), and showed that evaluating such queries is fixed-parameter tractable w.r.t. the size of the query. Finally, we also studied the problem of approximating WDPTs by one from a tractable class (a problem that is now well-understood in the context of CQs [2]).

Acknowledgments. The work of Pablo Barceló is funded by the Millenium Nucleus Center for Semantic Web Research under grant NC120004. Part of this work was done while Reinhard Pichler and Sebastian Skritek were visiting Pablo Barceló on invitation by the Millenium Nucleus Center for Semantic Web Research. Reinhard Pichler and Sebastian Skritek were supported by the Vienna Science and Technology Fund (WWTF) through project ICT12-015 and by the Austrian Science Fund (FWF):P25207-N23.

References

1. S. Ahmetaj, W. Fischl, R. Pichler, M. Simkus, and S. Skritek. Towards reconciling sparql and certain answers. In *WWW'15*, 2015. Accepted for publication.
2. P. Barceló, L. Libkin, and M. Romero. Efficient approximations of conjunctive queries. *SIAM J. Comput.*, 43(3):1085–1130, 2014.

3. P. Barcelo, R. Pichler, and S. Skritek. Efficient evaluation and approximation of well-designed pattern trees. In *PODS'15*, 2015. Accepted for publication.
4. P. Barceló, M. Romero, and M. Y. Vardi. Semantic acyclicity on graph databases. In *PODS'13*, pages 237–248, 2013.
5. P. Barceló, M. Romero, and M. Y. Vardi. Does query evaluation tractability help query containment? In *PODS'14*, pages 188–199, 2014.
6. C. Chekuri and A. Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.
7. V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *CP'02*, pages 310–326, 2002.
8. G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.
9. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. Comput. Syst. Sci.*, 64(3):579–627, 2002.
10. A. Letelier, J. Pérez, R. Pichler, and S. Skritek. Static analysis and optimization of semantic web queries. *ACM Trans. Database Syst.*, 38(4):25, 2013.
11. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.
12. R. Pichler and S. Skritek. Containment and equivalence of well-designed SPARQL. In *PODS'14*, pages 39–50, 2014.
13. M. Yannakakis. Algorithms for acyclic database schemes. In *VLDB'81*, pages 82–94, 1981.

Approximation Algorithms for Schema-Mapping Discovery from Data Examples

Balder ten Cate[2,1], Phokion G. Kolaitis[1,3], Kun Qian[1], and Wang-Chiew Tan[1]

University of California Santa Cruz[1]
LogicBlox, Inc.[2]
IBM Research - Almaden[3]
{btencate,kolaitis,kunqian,tan}@soe.ucsc.edu

Abstract. In recent years, data examples have been at the core of several different approaches to schema-mapping design. In particular, Gottlob and Senellart introduced a framework for schema-mapping discovery from a single data example, in which the derivation of a schema mapping is cast as an optimization problem. Our goal is to refine and study this framework in more depth. Among other results, we design a polynomial-time $\log(n)$-approximation algorithm for computing optimal schema mappings from a given data example, for a restricted class of schema mappings; moreover, we show that this approximation ratio cannot be improved.

Keywords: Schema Mappings, Data Examples, Approximation

1 Introduction

Schema mappings between a source schema and a target schema constitute the essential building blocks for the specification of data exchange and data integration tasks [6,15,16]. However, deriving a schema mapping between two schemas can be an involved and time-consuming process. Most commercial systems (e.g., Altova Mapforce and the Stylus Studio) and research prototypes derive schema mappings automatically using correspondences between the elements of two schemas, as specified by a user through a graphical interface [7,13,18]. A shortcoming of this approach is that multiple non-equivalent schema mappings may be compatible with the same set of correspondences. More recently, data examples have been used as the basis of alternative approaches to schema-mapping design.

Data examples have been used in several different areas of data management (e.g., see [11,17,19,21]). In the context of schema-mapping design, a data example is a pair (I, J) consisting of a source instance and a target instance. In [2,22], the goal is, given a schema mapping, to *produce* meaningful data examples, that illustrate the schema mapping at hand. In [1], the goal is to investigate whether a given schema mapping can be *uniquely characterized* by a finite set of examples. In the reverse direction, there has been work on deriving a schema mapping that *fits* one or more given data examples [3,8]

Gottlob and Senellart [12] introduced and studied a cost model for deriving a schema mapping from a ground data example, i.e., a data example consisting of instances without nulls. Ideally, given a ground data example (I, J), one would like to find a GLAV schema mapping $\mathcal{M}$ that is *valid* (meaning that (I, J) satisfied the constraints of $\mathcal{M}$) and *fully explaining* for (I, J) (meaning that every fact in J belongs to every target instance K such that (I, K) satisfies the constraints of $\mathcal{M}$). However, such a schema mapping need not exist. For this reason, Gottlob and Senellart developed a framework that uses two schema-mapping languages: the standard language of GLAV constraints and an extended language GLAV$^{=,\neq}$ of *repairs* that augments, in precise way, GLAV constraints with equalities, inequalities, and ground facts. The *cost* of a GLAV schema mapping $\mathcal{M}$ w.r.t. a data example (I, J) is the minimum of the sizes of valid and fully explaining schema mappings $\mathcal{M}'$ obtained from $\mathcal{M}$ via a sequence of operations that transform $\mathcal{M}$ to a schema mapping in the extended language. Given a ground data example (I, J), the goal then is to find an *optimal* GLAV schema mapping for (I, J), that is to say, a GLAV schema mapping whose cost w.r.t. (I, J) is as small as possible. Gottlob and Senellart [12] investigated several different decision problems arising naturally in the context of the above framework, including the Existence-Cost problem: given a ground data example (I, J) and a positive integer k, is there a GLAV schema mapping $\mathcal{M}$ whose cost w.r.t. (I, J) is at most k? In [12], it is shown that the Existence-Cost problem is NP-hard, and that it belongs to the third level Σ_3^p of the polynomial hierarchy.

Here, we contribute to the study of schema-mapping discovery from data examples by refining and investigating the Gottlob-Senellart framework in more depth. We refine the framework by considering several different schema-mapping languages. At the base level, we consider sublanguages $\mathcal{L}$ of the standard language of GLAV constraints, such as the languages of GAV constraints, LAV constraints, and SH-LAV (single-head LAV) constraints. For each of these schema-mapping languages $\mathcal{L}$, we consider two corresponding repair languages, namely, $\mathcal{L}^{=,\neq}$ and $\mathcal{L}^{=}$; the former extends $\mathcal{L}$ with equalities, inequalities, and ground facts (as in [12]), while the latter extends $\mathcal{L}$ with equalities and ground facts only.

The main algorithmic problem in the Gottlob-Senellart framework is to compute an optimal schema mapping for a given ground data example. The tractability of this optimization problem was left open in [12] (the hardness of the Existence-Cost problem does not imply hardness of the optimization problem, because computing the cost of a given schema mapping for a given ground data example is itself a hard problem). We show that this optimization problem is indeed hard for GLAV mappings, w.r.t. both GLAV$^{=}$-repair and GLAV$^{=,\neq}$-repairs. More precisely, unless RP = NP, there is no polynomial-time algorithm that, given a ground data example, computes a GLAV mapping whose cost is bounded by some fixed polynomial in the cost of the optimal GLAV mapping. Moreover, an analogous result holds for GAV mappings.

After this, we design an approximation algorithm for computing near optimal schema mappings for the case of SH-LAV schema mappings. Specifically, we present a polynomial-time $O(\log n)$-approximation algorithm that, given a data example (I, J), produces a SH-LAV mapping $\mathcal{M}$ together with a SH-LAV$^{=}$-repair $\mathcal{M}'$ of $\mathcal{M}$ for (I, J), whose cost is within a logarithmic factor of the cost of the optimal SH-LAV mapping for (I, J). Finally, we establish that this is a best possible approximation result.

2 Preliminaries

Schemas and Instances An *instance* over a schema $\mathbf{R} = \{R_1, \ldots, R_k\}$ can be identified with the finite set of all *facts* $R_i(a_1, \ldots, a_m)$, such that R_i is a relation symbol of $\mathbf{R}$ and $(a_1, \ldots, a_m)$ is a tuple that belongs to the relation R_i^I of I interpreting R_i. Database instances contain values that are either *constants* or *nulls.* A *ground instance* is an instance such all of its facts are *ground.* i.e., they consist entirely of constants. In this paper, we will primarily consider ground instances, and we will, at times, drop the adjective "ground". We write $adom(I)$ to denote the *active domain* of an instance I.

Schema Mappings Let $\mathbf{S}, \mathbf{T}$ be two relational schemas, called the *source* schema and the *target* schema. A *schema mapping* is a triple $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ consisting of a source schema $\mathbf{S}$, a target schema $\mathbf{T}$, and a set Σ of constraints that are typically expressed in some fragment of first-order logic.

A *GLAV (Global-and-Local-As-View) constraint*, also known as a *tuple-generating dependency (tgd)*, is a FO-formula of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x},\mathbf{y}))$, where $\varphi(\mathbf{x})$ is a conjunction of atoms over $\mathbf{S}$ and $\psi(\mathbf{x},\mathbf{y})$ is a conjunction of atoms over $\mathbf{T}$. We will often drop the universal quantifiers when writing constraints.

The following are two important special cases of GLAV constraints: (1) A GAV (Global-As-View) constraint is a GLAV constraint whose right-hand side is a single atom without existential quantifiers, i.e., it is of the form $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow T(\mathbf{x}))$. (2) A LAV (Local-As-View) constraint is GLAV constraint whose left-hand side is a single atom, i.e. it is of the form $\forall \mathbf{x}(S(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x},\mathbf{y}))$. There is another special case of LAV constraints, called *SH-LAV (Single-Head LAV) constraints.* A SH-LAV constraint is a GLAV constraint in which both the left-hand side and right-hand side are a single atom, i.e., it is of the form $\forall \mathbf{x}(S(\mathbf{x}) \rightarrow \exists \mathbf{y} T(\mathbf{x},\mathbf{y}))$. In fact every DL-lite concept subsumption axiom is equivalent to a SH-LAV constraint [5].

Example 1. $\text{Geo}(x, y) \rightarrow \text{City}(y)$ is a LAV constraint that is also a GAV constraint, and hence, in particular, is a SH-LAV contraint; $\text{Geo}(x, y) \rightarrow \exists z \text{City}(z)$ is a SH-LAV constraint that is not a GAV constraint. Finally, the constraint $\text{Geo}(x, y) \wedge \text{Geo}(x, z) \rightarrow \text{City}(y)$ is a GAV constraint that is not a LAV constraint.

A *GLAV schema mapping* is a schema mapping $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where Σ is a finite set of GLAV constraints. The notions of GAV, LAV, and SH-LAV schema mappings are defined in an analogous way.

Data Examples Let $\mathbf{S}$ be a source schema and $\mathbf{T}$ be a target schema. A *data example* is a pair (I, J) such that I is a source instance and J is a target instance. A data example (I, J) is *ground* if both I and J are ground instances. The size $||(I, J)||$ of a data example (I, J) is the total number of facts in I and J.

Table 1. Data Example (I, J)

Source Instance		**Target Instance**
Geo(US, San Francisco)	Geo(Calif, San Jose)	City(San Francisco)
Geo(US, Los Angeles)	Geo(Calif, San Francisco)	City(San Jose)
Geo(US, Miami)	Geo(Calif, Los Angeles)	City(San Diego)
Geo(US, Boston)	Geo(Calif, San Diego)	City(Los Angeles)
Geo(US, New York)	Geo(Canada, Vancouver)	City(Boston)
Geo(NorthAm, Boston)	Geo(Germany, Berlin)	City(Toronto)
Geo(NorthAm, Toronto)	Geo(Japan, Tokyo)	City(New York)
Geo(NorthAm, New York)	Geo(China, Beijing)	City(Miami)
Geo(NorthAm, Miami)	Geo(France, Paris)	
	Geo(UK, London)	

Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$ be a schema mapping, and let (I, J) be a ground data example. We say that $\mathcal{M}$ is *valid* for (I, J) if $(I, J) \models \Sigma$, that is, $(I, J) \in E$ satisfies all constraints in Σ. We say that $\mathcal{M}$ *explains* a fact f of J with respect to I if, for all target instances K such that $(I, K) \models \Sigma$, we have that $f \in K$. Finally, we say that $\mathcal{M}$ is *fully explaining* for a data example (I, J) if $\mathcal{M}$ explains each fact of J with respect to I. We will use the expression *vfe* as a shorthand for "valid and fully explaining".

In [1, 3], a schema mapping $\mathcal{M}$ was said to *fit* a data example (I, J) if J is a universal solution of I w.r.t. $\mathcal{M}$, i.e., J is a solution for I w.r.t. $\mathcal{M}$ such that for every solution J' for I, there is a homomorphism from J to J' that maps constants to themselves (see [10] for more on universal solutions). It is not hard to verify that if (I, J) is a ground data example and $\mathcal{M}$ is a schema mapping, then $\mathcal{M}$ fits (I, J) if and only if $\mathcal{M}$ is vfe for (I, J).

Example 2. Let $\mathcal{M} = (\mathbf{S}, \mathbf{T}, \Sigma)$, where $\mathbf{S} = \{\text{Geo(area, city)}\}$, $\mathbf{T} = \{\text{City(cityName)}\}$, $\Sigma = \{\text{Geo}(x, y) \to \text{City}(y)\}$. Consider the data examples (I_i, J_i), $i = 1, 2, 3$, where

I_1 : Geo(CA, San Francisco), Geo(CA,San Jose), Geo(US,Boston), Geo(US, Los Angeles)
J_1 : City(San Francisco), City(San Jose)

I_2 : Geo(CA, San Francisco)
J_2 : City(San Francisco), City(New York)

I_3 : Geo(CA, San Francisco), Geo(US,New York)
J_3 : City(San Francisco), City(New York)

In these data examples, since I_1 contains Geo(US,Boston), but City(Boston) is not in J_1, we have that $\mathcal{M}$ is not valid for (I_1, J_1); moreover, $\mathcal{M}$ is valid for (I_2, J_2), but it fails to explain the target fact City(New York); however, $\mathcal{M}$ is valid and fully explaining for (I_3, J_3). Using a simple automorphism argument, it is easy to see there is no valid and fully explaining GLAV schema mapping for (I_1, J_1). Moreover, since J_2 contains a constant that does not appear in the I_2, there is no valid and fully explaining GLAV schema mapping for (I_2, J_2).

3 Repair Framework and Cost Model

In [12], the language of GLAV constraints was used as the "base language", and an extended "repair language" was introduced; the repair language includes equalities, inequalities, and ground facts, and is used for "repairing" GLAV schema mappings, so that they are valid and fully explaining for a given ground data example. We refine this framework by considering different sublanguages of the language of GLAV constraints, as well different repair languages.

Definition 1. *Fix a schema-mapping language $\mathcal{L}$ (e.g., GLAV). An* $\mathcal{L}^{=,\neq}$-constraint *is a formula θ of the form $\forall \boldsymbol{x}(\varphi(\boldsymbol{x}) \wedge \alpha(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y}(\psi(\boldsymbol{x},\boldsymbol{y}) \wedge \beta(\boldsymbol{x},\boldsymbol{y})))$, where*

1. *$\forall \boldsymbol{x}(\varphi(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \psi(\boldsymbol{x},\boldsymbol{y}))$ is an $\mathcal{L}$-constraint;*
2. *$\alpha(\boldsymbol{x})$ is a possibly empty conjunction of formulas of the form $x \sim c$, where $\sim \in \{=, \neq\}$, $x \in \boldsymbol{x}$, and c is a constant;*
3. *$\beta(\boldsymbol{x},\boldsymbol{y})$ is a possibly empty conjunction of formulas of the form $(x_1 = c_1 \wedge \ldots \wedge x_n = c_n) \rightarrow y = c$, where each $x_i \in \boldsymbol{x}$, $y \in \boldsymbol{y}$, and $c_1, \ldots, c_n, c$ are constants.*

A $\mathcal{L}^{=}$-constraint *is a $\mathcal{L}^{=,\neq}$-constraint with no conjuncts of the form $x \neq c$ in α.*

We will write $\mathcal{L}$ to denote a schema-mapping language (e.g., GLAV), and we will write $\mathcal{L}^{=,\neq}$ and $\mathcal{L}^{=}$ to denote the corresponding repair language (with and without inequalities). We will use $\mathcal{L}^*$ to refer to either $\mathcal{L}^{=,\neq}$ or $\mathcal{L}^{=}$. We say that the formula θ as above is a *$\mathcal{L}^*$-repair* of the constraint $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi(\mathbf{x},\mathbf{y}))$.

Definition 2. *An* $\mathcal{L}^*$-repair *of an $\mathcal{L}$-schema mapping $\mathcal{M} = (\boldsymbol{S}, \boldsymbol{T}, \Sigma)$ is an $\mathcal{L}^*$-schema mapping $\mathcal{M}' = (\boldsymbol{S}, \boldsymbol{T}, \Sigma')$ such that each $\psi \in \Sigma'$ is either a ground fact or is an $\mathcal{L}^*$-repair of some $\phi \in \Sigma$, and, for each $\phi \in \Sigma$, at least one $\mathcal{L}^*$-repair of ϕ belongs to Σ'. We write $repair_{\mathcal{L}^*}(\mathcal{M})$ to denote the set of all $\mathcal{L}^*$-repairs of $\mathcal{M}$.*

The above definition differs from that of repairs in [12] in that we allow $\mathcal{M}'$ to contain multiple repairs of the same $\mathcal{L}$-constraint from $\mathcal{M}$, whereas, in [12], the $\mathcal{L}^*$-constraints of $\mathcal{M}'$ stand in a one-to-one correspondence with the $\mathcal{L}$-constraints of $\mathcal{M}$. There are cases in which a $\mathcal{L}$-constraint may need to be repaired more than once with, say, different combinations of equalities and inequalities. In such cases, the optimal schema mapping in [12] may contain multiple copies of the same GLAV constraint or multiple GLAV constraints that are identical up to a renaming of variables (see Example 4). Our definition addresses this shortcoming.

Example 3. Recall that the data example (I_1, J_1) in Example 2 had no vfe GLAV schema mapping. Consider the following $GLAV^{=,\neq}$ schema mappings:

- $\mathcal{M}_1$=$\{\text{Geo}(x,y) \wedge (x = \text{CA}) \rightarrow \text{City}(y)\}$
- $\mathcal{M}_2$=$\{\text{Geo}(x,y) \rightarrow \exists z\ \text{City}(z) \wedge (y = \text{San Jose} \rightarrow z = \text{San Jose})$ $\wedge\ (y = \text{San Francisco} \rightarrow z = \text{San Francisco})\}$
- $\mathcal{M}_3$=$\{\text{City(San Francisco)}, \text{City(San Jose)}\}$
- $\mathcal{M}_4$=$\{\text{Geo}(x,y) \wedge (x \neq \text{US}) \rightarrow \text{City}(y)\}$
- $\mathcal{M}_5$=$\{\text{Geo}(x,y) \wedge (x = \text{moon}) \rightarrow \text{City}(x)\}$

$\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, and $\mathcal{M}_4$ are vfe for (I_1, J_1), but in different ways: $\mathcal{M}_1$ consists of a repair of $\text{Geo}(x,y) \rightarrow \text{City}(y)$ that uses an equality to restrict the constraint to source facts whose first attribute has value CA; $\mathcal{M}_2$ consists of a repair of $\text{Geo}(x,y) \rightarrow \exists z \text{City}(z)$ that uses two conditional equalities to explicitly specify a value of z depending on the value of y; $\mathcal{M}_3$ is a repair of the empty schema mapping, and it lists all the ground facts of J_1; finally, $\mathcal{M}_4$ consists of a repair of the same constraint as $\mathcal{M}_1$ that, instead, uses an inequality.

$\mathcal{M}_5$ is valid, but does not explain (I_1, J_1). It uses the equality $(x = \mathit{moon})$, where *moon* is outside the active domain of (I_1, J_1). We informally say that this equality *cancels* $\mathcal{M}_5$ (with respect to the data example (I_1, J_1)).

Example 4. We illustrate the need for multiple repairs of the same constraint. For the data example (I, J) in Table 1, consider the GLAV schema mappings

– $\mathcal{M}_1 = \{\mathrm{Geo}(x, y) \to \mathrm{City}(y)\}$,
– $\mathcal{M}_2 = \{\mathrm{Geo}(x_1, y_1) \to \mathrm{City}(y_1), \mathrm{Geo}(x_2, y_2) \to \mathrm{City}(y_2)\}$

and consider the GLAV$^{=,\neq}$-schema mapping $\mathcal{M}_r$ specified by the constraint

$$\mathrm{Geo}(x, y) \wedge (x = \mathrm{Calif}) \to \mathrm{City}(y),\ \mathrm{Geo}(x, y) \wedge (x = \mathrm{NorthAm}) \to \mathrm{City}(y).$$

Note that $\mathcal{M}_r$ is vfe for (I, J). Note also that $\mathcal{M}_2$ consists of two renamings of the GLAV constraints of $\mathcal{M}$. By our definition of repair, $\mathcal{M}_r$ is a repair of $\mathcal{M}_1$ and also of $\mathcal{M}_2$. However, according to the definition of repair in [12], $\mathcal{M}_r$ does not qualify as a repair of $\mathcal{M}_1$, and, indeed, one of the smallest repairs of $\mathcal{M}_1$ is obtained by adding an equality (x = US) to the left-hand side of the constraint in $\mathcal{M}_1$ together with adding three ground facts: City(San Jose), City(Toronto), and City(San Diego). Since the cost of a schema mapping w.r.t. a data example will be measured by the size of the smallest repair, we have that, under the cost model of [12], $\mathcal{M}_2$ has a lower cost than $\mathcal{M}_1$, which is counterintuitive.

As pointed out in [12], if (I, J) is a ground data example, then there is always a vfe GLAV$^{=,\neq}$ schema mapping for (I, J). In fact, trivially, the collection of all the ground facts in J is a vfe schema mapping for (I, J). This shows that, for all schema-mapping languages $\mathcal{L}$ considered her, every ground data example has a $\mathcal{L}^{=}$ repair. Indeed, every $\mathcal{L}$-schema mapping has a $\mathcal{L}^{=}$-repair that is vfe (obtained by cancelling all constraints and adding all ground target facts).

The cost model introduced in [12] focuses on finding a schema mapping in the base language, such that the cost of transforming it to a vfe schema mapping in the repair language is as small as possible.

Definition 3. *[12] The* size *of a first-order formula φ, denoted size(φ), is the number of occurrences of variables and constants in φ (each variable and constant is counted as many times as it occurs in φ); occurrences of variables as arguments of quantifiers do not count. A ground fact $R(a_1, \ldots, a_n)$, for present purposes, is viewed as shorthand for $\exists x_1, \ldots, x_n R(x_1, \ldots, x_n) \wedge x_1 = a_1 \wedge \cdots \wedge x_n = a_n$; therefore, we consider its* size *to be $3n$. The* size *of a repair of a schema mapping is the sum of the sizes of the constraints and the ground facts of the repair.*

Note that the motivation for the above treatment of ground facts is to discourage the use of ground facts in repairing schema mappings.

Definition 4. *The* cost *of an $\mathcal{L}$-schema mapping $\mathcal{M}$ for a data example (I, J) and a repair language $\mathcal{L}^*$, denoted by cost($\mathcal{M}, (I, J), \mathcal{L}^*$), is the smallest size of a vfe $\mathcal{L}^*$-repair of $\mathcal{M}$ for (I, J). An $\mathcal{L}$-schema mapping $\mathcal{M}$ is $\mathcal{L}^*$-*optimal for *(I, J) if cost($\mathcal{M}, (I, J), \mathcal{L}^*$) is the minimum of the quantities cost($\mathcal{M}', (I, J), \mathcal{L}^*$), where $\mathcal{M}'$ ranges over all $\mathcal{L}$-schema mappings.*

Example 5. We continue from Example 3 by considering the schema mappings $\mathcal{M}_a = \{\text{Geo}(x, y) \to \text{City}(y)\}$, $\mathcal{M}_b = \{\text{Geo}(x, y) \to \exists z\ \text{City}(z)\}$, and $\mathcal{M}_c = \emptyset$.

Both $\mathcal{M}_1$ and $\mathcal{M}_5$ are vfe repairs of $\mathcal{M}_a$ for (I_1, J_1) and both have size 5; it is easy to verify that no other vfe repairs of $\mathcal{M}_a$ has size less than 5, hence $cost(\mathcal{M}_a, \{(I_1, J_1)\}, GLAV^{=,\neq}) = 5$. The repair $\mathcal{M}_3$ is the only vfe repair of $\mathcal{M}_c$, and $cost(\mathcal{M}_c, \{(I_1, J_1)\}, GLAV^{=,\neq}) = 6$. Moreover, $\mathcal{M}_2$ is a vfe repair of $\mathcal{M}_b$ and $size(\mathcal{M}_2) = 11$, but the cost of $\mathcal{M}_b$ is not 11. To see this, consider the schema mapping $\mathcal{M}_2'$ specified by the constraint

$$\mathcal{M}_2' = \{\text{Geo}(x, y) \to \exists z\ \text{City}(z) \wedge (z = \text{San Jose}), \text{City}(\text{San Francisco})\}.$$

Clearly, $\mathcal{M}_2'$ is also a vfe repair of $\mathcal{M}_b$ for (I_1, J_1) and has size of 8.

The notion of an optimal schema mapping in Definition 4 differs from the corresponding definition in [12] in that we consider a slightly different notion of repair, as explained in the remarks following Definition 2. A consequence of this is that an optimal schema mapping in our sense has cost less than or equal than the cost of an optimal schema mapping in the sense of [12]. However, the complexity-theoretic results concerning the various algorithmic problems do not depend on this, and, indeed, the proofs are not affected by this change.

4 Hardness of Computing Optimal Schema Mappings

As we have seen, the main characteristic of the framework introduced in [12] and refined here is to cast schema-mapping discovery as an optimization problem: given a finite set of ground data examples, produce a schema mapping of minimum cost. Two different decision problems naturally arise in this setting.

Definition 5. *The decision problem* $\text{COST}_{\mathcal{L}^*}$ *asks: given a ground data example* (I, J)*, a* $\mathcal{L}$*-schema mapping* $\mathcal{M}$*, and an integer* $k \geq 0$*, is* $cost(\mathcal{M}, (I, J), \mathcal{L}^*) \leq k$*?*

Definition 6. *The decision problem* $\text{EXISTENCE-COST}_{\mathcal{L}^*}$ *asks: given a ground data example* (I, J) *and an integer* $k \geq 0$*, does there exists a* $\mathcal{L}$*-schema mapping* $\mathcal{M}$ *such that* $cost(\mathcal{M}, (I, J), \mathcal{L}^*) \leq k$*?*

In these two problems, the source schema and the target schema are part of the input. One can also consider the variants of these problems, such as $\text{EXISTENCE-COST}_{\mathcal{L}^*}(\mathbf{S}, \mathbf{T})$, obtained by fixing the source and target schemas.

The preceding problems were introduced and studied in [12] when the base language $\mathcal{L}$ is GLAV or GAV, and the repair language is $\mathcal{L}^{=,\neq}$. It was shown there that $\text{COST}_{\text{GLAV}^{=,\neq}}$ belongs to Σ_3^p (the third level of the polynomial hierarchy) and is Π_2^p-hard, while $\text{COST}_{\text{GAV}^{=,\neq}}$ belongs to Σ_2^p and is DP-hard. It was also shown that $\text{EXISTENCE-COST}_{\text{GLAV}^{=,\neq}}$ belongs to Σ_3^p and that $\text{EXISTENCE-COST}_{\text{GAV}^{=,\neq}}$ belongs to Σ_2^p; moreover, both these problems were shown to be NP-hard.[1]

We show that the problems COST and EXISTENCE-COST are NP-complete for the languages of LAV schema mappings and SH-LAV schema mappings. Neither of these schema-mapping languages was considered in [12].

[1] The NP-hardness proof given in [12] is flawed. The authors have shared with us, in private communication, a correct NP-hardness proof.

Theorem 1. *Assume that* $\mathcal{L}^* \in \{\text{LAV}^{=,\neq}, \text{LAV}^{=}, \text{SH-LAV}^{=,\neq}, \text{SH-LAV}^{=}\}$. *Then the problems* $\textsc{Cost}_{\mathcal{L}^*}$ *and* $\textsc{Existence-Cost}_{\mathcal{L}^*}$ *are NP-complete. In fact, there are fixed schemas* $\boldsymbol{S}$ *and* $\boldsymbol{T}$ *for which these problems are NP-complete.*

Theorem 1 is established via a reduction from the $\textsc{Set-Cover}$ problem. Next, we show that the problem of computing optimal GLAV schema mappings is not solvable in polynomial time, unless RP=NP. Note that this does not follow directly from the Π_2^p-hardness of $\textsc{Existence-Cost}_{GLAV^{=,\neq}}$ established in [12], because $\textsc{Cost}_{GLAV^{=,\neq}}$ is DP-hard. The same holds true for GAV mappings. Actually, we show a stronger result: unless RP = NP, there is no polynomial-time algorithm that, given a ground data example (I, J), computes a GLAV schema mapping whose cost is bounded by a polynomial in the cost of the optimal GLAV schema mapping; moreover, the same holds true for GAV schema mappings.

Theorem 2. *Assume that* $\mathcal{L} \in \{GLAV, GAV\}$ *and let* $p(x)$ *be a polynomial. Unless* RP = NP*, there is no polynomial-time algorithm that, given a ground data example* (I, J)*, computes a* $\mathcal{L}$*-schema mapping* $\mathcal{M}$ *such that* $cost(\mathcal{M}, (I, J), \mathcal{L}^{=,\neq}) \leq p(cost(\mathcal{M}_{opt}, (I, J), \mathcal{L}^{=,\neq}))$*, where* $\mathcal{M}_{opt}$ *is an* $\mathcal{L}^{=,\neq}$*-optimal schema mapping for* (I, J)*. Moreover, the same holds true when* $\mathcal{L}^{=,\neq}$ *is replaced by* $\mathcal{L}^{=}$*. In fact, there are fixed source and target schemas for which these results hold.*

Theorem 2 also shows that the computational hardness is, to some extent, robust under changes to the precise cost function. The proof is based on procedure that transforms a ground data example (I, J) into another ground data example (I', J'), which, intuitively, contains many isomorphic copies of the original data example (I, J), such that every near-optimal GLAV (GAV) schema mapping for (I', J') corresponds to a fitting GLAV (GAV) schema mapping for (I, J) of near-minimal size. This is combined with a hardness result for computing fitting GAV schema mappings of near-minimal size, established in [8].

5 Approximation of Optimal Single-Head LAV Mappings

We now study the approximation properties of the following optimization problem:

Definition 7. *The* $\textsc{Optimal-Repair}_{\mathcal{L}^*}(\boldsymbol{S}, \boldsymbol{T})$ *problem asks: given a ground data example* (I, J) *with source schema* $\boldsymbol{S}$ *and target schema* $\boldsymbol{T}$*, compute a minimal-size valid and fully explaining* $\mathcal{L}^*$*-schema mapping for* (I, J) *and* $\mathcal{L}^*$*.*

This problem is equivalent to the problem that asks to compute an optimal $\mathcal{L}$-schema mapping $\mathcal{M}$ together with a minimal-size $\mathcal{L}^*$-repair of $\mathcal{M}$; in particular, it contains, as a special case, the problem of computing an optimal $\mathcal{L}$-schema mapping for a given ground data example. This is so because, from a minimal-size valid and fully explaining $\mathcal{L}^*$-schema mapping, we can immediately extract an optimal $\mathcal{L}$-schema mapping by, simply by dropping all equalities, inequalities, and ground facts. Therefore, it follows from Theorem 2 that (assuming RP $\neq$ NP) there is no polynomial-time algorithm that solves $\textsc{Optimal-Repair}_{\mathcal{L}^*}$, when $\mathcal{L}^* \in \{GLAV^{=}, GLAV^{=,\neq}, GAV^{=}, GAV^{=,\neq}\}$.

We establish a positive approximability result for SH-LAV$^{=}$ mappings.

Theorem 3. *Fix a pair of schemas $\boldsymbol{S}, \boldsymbol{T}$. There is a polynomial-time $\mathcal{H}(n)$-approximation algorithm for* OPTIMAL-REPAIR$_{SH\text{-}LAV^{=}}(\boldsymbol{S}, \boldsymbol{T})$*, where $\mathcal{H}(n) = \sum_{i=1}^{n} \frac{1}{i}$ is the n-th harmonic number.*

Our approximation algorithm is obtained by establishing a close connection between SET-COVER and OPTIMAL-REPAIR$_{\mathcal{L}^*}$. It is known that, although the SET-COVER problem is not constant-approximable, it is $\mathcal{H}(n)$-approximable, where n is the size of the universe. Moreover, this approximation ratio is known to be asymptotically optimal: unless P = NP, SET-COVER can only be approximated up to a factor of $c\ln(n)$, where c is a constant (specified in [4, 14]) and n is the size of universe (recall that $\mathcal{H}(n) = O(\ln n)$). Our approximation algorithm is largely based on the approximation algorithm of WEIGHTED SET-COVER [9]. Here, we can think of each constraint as describing a set of facts, namely, the set of target facts that it explains, and the size of the constraint is the weight of the corresponding set.

The above approximation algorithm can be extended in a straightforward manner to the case of SH-LAV$^{=,\neq}$-constraints with a bounded number of inequalities per variable, as well as to LAV$^{=,\neq}$-constraints with a bounded number of inequalities per variable and a bounded number of atoms in the right-hand side. We leave it as an open problem whether an analog of Theorem 3 holds for the general case of SH-LAV$^{=,\neq}$ and LAV$^{=,\neq}$.

We conclude with a matching lower bound for the approximability of OPTIMAL-REPAIR$_{\text{SH-LAV}^{=}}(\mathbf{S}, \mathbf{T})$. This result is established via an approximation-preserving reduction (more precisely, an L-reduction [20]) from MINIMUM SET COVER.

Theorem 4. *Let $\mathcal{L} \in \{LAV, SH\text{-}LAV\}$. There are fixed schemas $\boldsymbol{S}$ and $\boldsymbol{T}$, and a constant c such that there is no polynomial-time $c\ln(n)$-approximation algorithm for* OPTIMAL-REPAIR$_{\mathcal{L}^{=}}(\boldsymbol{S}, \boldsymbol{T})$*, where n is the total number of target facts in the input data example. The same holds true for* OPTIMAL-REPAIR$_{\mathcal{L}^{=,\neq}}(\boldsymbol{S}, \boldsymbol{T})$.

6 Concluding Remarks

The cost function in Definition 5 naturally extends to sets of data examples, where $cost(\mathcal{M}, E, \mathcal{L}^*)$ is defined as the smallest size of an $\mathcal{L}^*$-repair of $\mathcal{M}$ that is vfe for each $(I, J) \in E$ (provided that such a repair exists). Similarly, the decision problems and optimization problems we studied naturally extend to the case where the input is a finite set of data examples. All upper bounds presented in this paper hold true also in the more general setting for multiple data examples.

Two important questions that remain to be answered are the existences of a polynomial time $\mathcal{H}(n)$-approximation algorithms for computing near-optimal LAV$^{=,\neq}$ and SH-LAV$^{=,\neq}$ schema mappings.

We have focused on obtaining a complexity-theoretic understanding of the algorithmic aspects of schema-mapping discovery from data examples. Our results pave the way for leveraging further the rich area of approximation algorithms and applying it to schema-mapping discovery. In parallel, we have embarked on a prototype implementation of our approximation algorithm enhanced with heuristic rules. While much remains to be done, there is preliminary evidence that this is an approach that may lead to a reasonably efficient system for schema-mapping discovery from data examples.

7 Acknowledgments

Research partially supported by NSF Grant IIS-1217869; Tan is partially supported by NSF grant IIS-1450560.

References

1. B. Alexe, B. T. Cate, P. G. Kolaitis, and W.-C. Tan. Characterizing schema mappings via data examples. *ACM Trans. Database Syst.*, 36(4):23:1–23:48, 2011.
2. B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. Muse: Mapping Understanding and deSign by Example. In *ICDE*, pages 10–19, 2008.
3. B. Alexe, B. ten Cate, P. G. Kolaitis, and W. C. Tan. Designing and refining schema mappings via data examples. In *SIGMOD*, pages 133–144, 2011.
4. N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, Apr. 2006.
5. A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The dl-lite family and relations. *JAIR*, 36:1–69, 2009.
6. P. Barceló. Logical foundations of relational data exchange. *SIGMOD Record*, 38(1):49–58, 2009.
7. A. Bonifati, E. Chang, T. Ho, V. Lakshmanan, and R. Pottinger. HePToX: Marrying XML and Heterogeneity in Your P2P Databases. In *VLDB*, pages 1267–1270, 2005.
8. B. ten Cate, V. Dalmau, and P. G. Kolaitis. Learning schema mappings. *ACM Trans. Database Syst.*, 38(4):28, 2013.
9. V. Chvtal. A greedy heuristic for the set covering problem. *Math. Oper. Res.*, 4:233–235, 1979.
10. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *TCS*, 336(1):89–124, 2005.
11. G. H. Fletcher and C. M. Wyss. Towards a general framework for effective solutions to the data mapping problem. *Journal on Data Semantics*, XIV, 2009.
12. G. Gottlob and P. Senellart. Schema mapping discovery from data instances. *JACM*, 57(2), 2010.
13. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: From Research Prototype to Industrial Tool. In *ACM SIGMOD*, pages 805–810, 2005.
14. D. S. Johnson. Approximation algorithms for combinatorial problems. In *Proceedings of STOC*, pages 38–49, New York, NY, USA, 1973. ACM.
15. P. G. Kolaitis. Schema Mappings, Data Exchange, and Metadata Management. In *ACM PODS*, pages 61–75, 2005.
16. M. Lenzerini. Data Integration: A Theoretical Perspective. In *ACM PODS*, pages 233–246, 2002.
17. H. Mannila and K.-J. Räihä. Automatic generation of test data for relational queries. *JCSS*, 38(2):240–258, 1989.
18. R. J. Miller, L. M. Haas, and M. A. Hernández. Schema Mapping as Query Discovery. In *VLDB*, pages 77–88, 2000.

19. C. Olston, S. Chopra, and U. Srivastava. Generating example data for dataflow programs. In *ACM SIGMOD*, pages 245–256, 2009.
20. C. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proceedings of STOC*, pages 229–234, New York, NY, USA, 1988. ACM.
21. A. D. Sarma, A. G. Parameswaran, H. Garcia-Molina, and J. Widom. Synthesizing view definitions from data. In *ICDT*, pages 89–103, 2010.
22. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-Driven Understanding and Refinement of Schema Mappings. In *ACM SIGMOD*, pages 485–496, 2001.

IMGpedia: A Proposal to Enrich DBpedia with Image Meta-Data

Benjamin Bustos and Aidan Hogan

Center for Semantic Web Research
Department of Computer Science
University of Chile
{bebustos,ahogan}@dcc.uchile.cl

Abstract. We introduce IMGpedia: a research proposal aiming to bridge structured knowledge-bases and multimedia content. Our concrete plan is to enrich DBpedia data with further metadata about images from Wikipedia, including content-based visual descriptors. Our concrete goal is to create a unified querying and inference system that allows for interrogating the DBpedia knowledge-base and the visual content of Wikipedia's images together. Our broader ambition is to explore methods by which multimedia data can be made a first-class citizen of the Semantic Web.

1 Introduction

DBpedia [1] is an ongoing effort by the Linked Data community to extract structured content from Wikipedia and represent it in RDF. The main goal is to enable users to query the content of Wikipedia as a whole, getting direct answers automatically aggregated from multiple articles. The most recent version of DBpedia contains billions of facts extracted from 125 language versions of Wikipedia, with links to and from dozens of external datasets. Over the past seven years, it has become *the* central dataset of the Linked Open Data community [5].

However, DBpedia mainly focuses on extracting information from Wikipedia's info-boxes: attribute–value panes that appear on the right-hand side of articles. As such, aside from adding links, DBpedia ignores the images appearing in the body of the article for a given entity as well as the structured data available in image pages: no meta-data are extracted for images. Like many initiatives in the Semantic Web[1], DBpedia links to but otherwise disregards multimedia content.

Our proposal is thus to extract and associate meta-data from the images embedded in Wikipedia and link the resulting corpus with the DBpedia dataset. This dataset – which we call IMGpedia – would consider all images in an article, all meta-data associated with the image available from Wikimedia (author, date, size, etc.) and visual descriptors that capture the content of the image itself.

[1] But not all: see, e.g., http://www.w3.org/2005/Incubator/mmsem/

We are motivated by the idea of creating a corpus that allows for querying, in unison, both the structured/semantic meta-data of DBpedia and the visual content extracted from images; e.g., "*give me Europe cathedrals that have an image visually similar to one of the external images for Cusco Cathedral in Peru*". Likewise, we foresee the possibility of inferring new links from this dataset, e.g., inferring that Saddam Hussein and Donald Rumsfeld have met based on being associated with the same image (in which they are co-depicted). The resulting corpus may also serve as an interesting experimental dataset for the image-processing community, where the structured data associated with images may serve as a ground truth.

2 Images and Visual Descriptors

Before describing IMGpedia, we need to introduce some basic concepts about how images are encoded and what are visual descriptors. An image is a matrix of so-called *pixels* (picture elements). A pixel contains information about its color, which can be displayed for example on a computer monitor. There are several ways to encode the color information of a pixel, which depends on the selection of a color space. Common color spaces are RGB (red-green-blue, used by computer monitors) and CMYK (cyan-magenta-blue-black, used by printers), where colors are represented as tuples of numbers; for example, an RGB color is represented by a three tuple. There are several ways to compress the image encoding, mainly lossy compression methods (e.g., JPEG format) and lossless methods (e.g., PNG format).

A visual descriptor is a way of characterizing an image based on its content. This can be done considering the whole image (global descriptor) or regions of interest detected on the image (local descriptors). For this work, we will initially focus on global descriptors since they can be computed more efficiently than local descriptors, and likewise similarity between them is also more efficient to compute.

Visual descriptors can be defined in several ways; e.g., based on the colours, texture and/or shape of the image. They do not include any semantic information about what appears on the image—hence why they are also called "low-level features". For instance, a simple colour descriptor is the colour histogram [2], that captures the distribution of colour in the image. We note that visual descriptors are usually vectors of high dimensionality (tens to hundreds of real values).

Visual descriptors allow us to implement, e.g., content-based similarity search. A similarity query in an image data set returns the most similar images, according to its content, to a given one (the query image). This is also known as *query-by-example*. Formally, let $\mathbb{U}$ be the universe of all images, let $\mathbb{S} \in \mathbb{U}$ be the image data set, and let $\delta : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}^+$ be a function (the distance) that returns how dissimilar are two images. There are two basic types of similarity queries: (1) *Range query*: given the query image $q \in \mathbb{U}$ and a tolerance radius $r \in \mathbb{R}^+$, return all images from S that are within distance r to q; (2) *Top-k query*: return the k-closest objects to q. If $\mathbb{S}$ is formed by all visual descriptors (high-dimensional vectors) extracted from the images in the data set, and if q is the visual descriptor of the query image, and if δ is any metric function (e.g., the Euclidean distance), it is relatively straightforward to implement content-based range and top-k queries over $\mathbb{S}$.

3 IMGpedia Dataset

Our vision of IMGpedia is an enhanced version of DBpedia with image entities. An image entity contains metadata (e.g., title, subject, source, format, description, date, size, location, etc.) and content-based descriptors (e.g., colour descriptor) of the image. Image entities can be linked with other entities (not necessarily images).

For creating the IMGpedia dataset itself, we propose the following procedure:

– Locate and download images/image-pages from Wikimedia.
– Extract meta-data from the image page, including its size, author, licence, etc. Annotate images with tags computed from its (possibly many) captions [4].
– Compute the visual descriptors for the images. For this, we can use global visual descriptors like colour and edge [2], following the MPEG-7 standard [3].
– Create the image entities using the extracted metadata and content-based data.
– Represent and publish the IMGpedia dataset as Linked Data.

4 Querying IMGpedia

Our main research goal is to investigate methods by which semantic data (in this case DBpedia) and multimedia data (in this case describing Wikipedia images) can be combined such that they can be queried in a holistic manner. In the context of IMGpedia, our approach is divided into three main parts: materialising links between image resources, extending SPARQL to execute content-based analysis at runtime, and inferring new links between "primary entities" based on image data.

Materialising relations between images using content-based descriptors. Low-level descriptors do not contain any semantic information about the original image, making them hard for users to leverage in queries. This problem is known as the *semantic gap* [6]. However, high-level relations among image entities can be computed from visual descriptors and similarity queries. For example, the relation `near-copy` can be defined as two different images with distance δ less than some threshold τ. By using range queries, it would be easy to find all pairs of near-copies among the images. Other relevant relations that can be considered are `alt-size`, `contains` and `similar`. These could also be materialised as triples and added to the structured knowledge-base, with appropriate inference – e.g., for symmetry, reflexivity or subsumption of relations – allowing users to specify SPARQL queries such as:

```
SELECT ?usPolitician WHERE {
  db:Saddam_Hussein foaf:depiction ?img1 .
  ?usPolitician dbo:party db:Republican_Party_(US) ;
     foaf:depiction ?img2 .
  ?img1 i:nearCopy ?img2 .
}
```

Extend SPARQL with functions for content-based image search. Not all content-based user requirements can be anticipated in the form of discrete relations. Hence we propose to extend SPARQL to include content-based analysis features. More specifically, we propose to use extensible functions in SPARQL and custom datatypes to enable queries that combine querying of semantic content and image content. Taking the introductory example, let's say that the user wishes to find cathedrals in Europe with similar images to external images of Cusco Cathedral in Lima:

```
SELECT ?cathedral ?sim WHERE {
  db:Cusco_Cathedral foaf:depiction ?img1 .
  FILTER(i:colorRatio(?img1,i:rgb(40,100,150),i:rgb(170,200,255)) > 0.2)
  ?eurCathedral rdf:type dbo:ReligiousBulding ;
    dbo:location [ dcterms:subject dbc:Countries_in_Europe ] ;
    foaf:depiction ?img2 .
  BIND(i:sim(?img1,?img2) as ?sim) FILTER(?sim > 0.7)
} ORDER BY ?sim
```

The first `FILTER` uses extended functions to only consider images that have more than 20% of their pixels falling within the cuboid of colours bounded by the two RGB points (looking for blue sky). The subsequent `BIND` and `FILTER` allow the images from European buildings to be filtered and ordered by similarity.

A major challenge here is balancing expressivity and efficiency. In the above case, given a reasonable query plan, the first filter can be applied over the six images appearing in the Cusco Cathedral article, but then all images of all religious buildings in Europe need to be compared with the images that pass the first step. In order to improve the performance of queries, we propose to investigate the use of image indexing techniques that allow for such filters to be executed a lookup, rather than a post-filter, which should lead to more options for query planning. For example, in the query above, a more efficient query plan may try to bind values for `?img2` using a similarity range query (over values bound for `?img1`) allowing for a join to be computed with the knowledge-base rather than applying a brute-force similarity filter over bindings produced by the knowledge-base for `?img2`.

We see this as being one of the deepest technical challenges posed by the work: creating cost models and query plans that combine indexes over the knowledge-base and multimedia content appears to be a challenging but general problem.

Content-based-driven knowledge discovery. A more speculative idea is to infer new knowledge about the data using the images entities and their relations. For example, say that two DBpedia resources are associated with the same (near-copy of an) image. If both resources are of type `dbo:Person`, the relation `hasMet` could be inferred. If one resource was a `dbo:Person` and the other was a `dbo:Place`, the relation `hasVisited` could be inferred. Such inferences could be axiomatised as domain-specific rules. Of course, the resulting inferences may not always be crisp conclusions, but may be associated with a confidence value.

5 Conclusions

In this short paper, we have introduced and motivated IMGpedia: a proposal to enrich DBpedia with meta-data extracted from Wikipedia images. We view IMGpedia as a concrete use-case through which to investigate the challenges and opportunities of combining semantic knowledge-bases with multimedia content.

Acknowledgements This work was supported by the Millennium Nucleus Center for Semantic Web Research, Grant № NC120004, and Fondecyt, Grant № 11140900.

References

1. J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 2014.
2. B. S. Manjunath, J.-R. Ohm, V. V. Vasudevan, and A. Yamada. Color and texture descriptors. *IEEE Transactions on Circuits and Systems for Video Technology*, 11(6):703–715, 2001.
3. MPEG-7 Overview. URL: http://mpeg.chiariglione.org/standards/mpeg-7/mpeg-7.htm (accessed: 2015–01–29), 2015.
4. S. Noah, D. Ali, A. Alhadi, and J. Kassim. Going Beyond the Surrounding Text to Semantically Annotate and Search Digital Images. In *Intelligent Information and Database Systems*, pages 169–179. 2010.
5. M. Schmachtenberg, C. Bizer, A. Jentzsch, and R. Cyganiak. Linking Open Data Cloud Diagram 2014. `http://lod-cloud.net/`; l.a. 2015/01/30.
6. A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

From Classical to Consistent Query Answering under Existential Rules

Thomas Lukasiewicz[1], Maria Vanina Martinez[2], Andreas Pieris[3], and Gerardo I. Simari[2]

[1] Department of Computer Science, University of Oxford, UK
thomas.lukasiewicz@cs.ox.ac.uk
[2] Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur and CONICET, Argentina {mvm,gis}@cs.uns.edu.ar
[3] Institute of Information Systems, Vienna University of Technology, Austria
pieris@dbai.tuwien.ac.at

Abstract. Querying inconsistent ontologies is an intriguing new problem that gave rise to a flourishing research activity in the description logic (DL) community. The computational complexity of consistent query answering under the main DLs is rather well understood; however, little is known about existential rules. The goal of the current work is to perform an in-depth analysis of the complexity of consistent query answering under the main decidable classes of existential rules enriched with negative constraints. Our investigation focuses on the standard inconsistency-tolerant semantics, namely, the AR semantics. We establish generic complexity results, which demonstrate the tight connection between classical and consistent query answering. These results allow us to obtain in a uniform way a relatively complete picture of the complexity of our problem.

1 Introduction

An ontology is an explicit specification of a conceptualization of an area of interest. One of the main applications of ontologies is in ontology-based data access (OBDA), where they are used to enrich the extensional data with intensional knowledge. In this setting, description logics (DLs) and rule-based formalisms such as existential rules are popular ontology languages, while conjunctive queries (CQs) form the central querying tool. In real-life applications, involving large amounts of data, it is possible that the data are inconsistent with the ontology. Since standard ontology languages adhere to the classical FOL semantics, inconsistencies are nothing else than logical contradictions. Thus, the classical inference semantics fails terribly when faced with an inconsistency, since everything follows from a contradiction. This demonstrates the need for developing inconsistency-tolerant semantics.

There has been a recent and increasing focus on the development of such semantics for query answering purposes. Consistent query answering, first developed for relational databases [1] and then generalized as the AR semantics for several DLs [9], is the most widely accepted semantics for querying inconsistent ontologies. The AR semantics is based on the idea that an answer is considered to be valid if it can be inferred from each of the repairs of the extensional data set D, i.e., the $\subseteq$-maximal consistent subsets of D. The complexity of query answering under the AR semantics when the ontology is described using one of the central DLs is rather well understood. The data and combined complexity were studied in [11] for a wide spectrum of DLs, while the work [2] identifies cases for simple ontologies (within the *DL-Lite* family) for which tractable data complexity results can be obtained. On the other hand, little is known when the ontology is described using existential rules (a.k.a. tuple-generating dependencies (TGDs) and Datalog$^{\pm}$ rules), that is, formulas of the form $\forall \mathbf{X} \forall \mathbf{Y}(\varphi(\mathbf{X},\mathbf{Y}) \rightarrow \exists \mathbf{Z}(\psi(\mathbf{X},\mathbf{Z})))$, and negative constraints (NCs) of the form $\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow \bot)$, where $\bot$ denotes the truth constant *false*.

Our main goal in this work is to perform an in-depth analysis of the data and combined complexity of consistent query answering under the main decidable classes of existential rules, enriched with negative constraints. Let us recall that the main (syntactic) conditions on existential rules that guarantee the decidability of query answering are guardedness [3], stickiness [4] and acyclicity. Interestingly, our complexity analysis shows that a systematic and uniform way for transferring complexity results from classical to consistent query answering can be formally established.

To briefly summarize the main contributions:

- We present generic complexity results, which demonstrate the tight connection between classical and consistent query answering (Theorems 1 and 2).
- By exploiting our generic theorems, we obtain a (nearly) complete picture of the combined and data complexity of consistent query answering (Table 2).

For more details we refer the reader to [10].

2 Consistent Query Answering

In the classical setting of CQ answering, given a database D and a set Σ of TGDs and NCs, if the models of D and Σ, denoted $mods(D,\Sigma)$, is empty, then every query is entailed since everything is inferred from a contradiction.

Example 1. Consider the database $D = \{professor(\text{John}), fellow(\text{John})\}$, asserting that John is both a professor and a fellow, and the set Σ of TGDs and NCs consisting of

$$\forall X(professor(X) \rightarrow \exists Y(faculty(X) \wedge teaches(X,Y)))$$
$$\forall X(fellow(X) \rightarrow faculty(X))$$
$$\forall X(professor(X) \wedge fellow(X) \rightarrow \bot),$$

expressing that each professor is a faculty member who teaches a course, each fellow is a faculty member, and professors and fellows form disjoint sets. It is easy to see that $mods(D,\Sigma) = \varnothing$, since John violates the disjointness constraint; thus, for every (Boolean) CQ q, $(D \wedge \Sigma) \models q$. ■

As said above, the AR semantics is the standard semantics for querying inconsistent ontologies. A key notion, which is necessary for defining the AR semantics, is that of repair, which is a $\subseteq$-maximal consistent subset of the given database.

Definition 1. Consider a database D, and a set Σ of TGDs and NCs. A *repair* of D and Σ is some $D' \subseteq D$ such that (i) $mods(D', \Sigma) \neq \varnothing$; and (ii) there is no $\underline{a} \in (D \setminus D')$ for which $mods(D' \cup \{\underline{a}\}, \Sigma) \neq \varnothing$. Let $rep(D, \Sigma)$ be the set of repairs of D and Σ.

Example 2. Consider the database D and the set Σ of TGDs and NCs given in Example 1. The set of repairs of D and Σ consists of the following subsets of D:

$$D_1 = \{professor(\text{John})\} \qquad D_2 = \{fellow(\text{John})\}.$$

Clearly, we simply need to remove one of the database atoms in order to satisfy the single negative constraint occurring in Σ. ■

The AR semantics [9] is based on the idea that a query can be considered to hold if it can be inferred from each of the repairs.

Definition 2. Consider a database D, a set Σ of TGDs and NCs, and a Boolean CQ q. We say that q is entailed by D and Σ under the *AR semantics*, written $(D \wedge \Sigma) \models_{AR} q$, if $(D' \wedge \Sigma) \models q$, for every $D' \in rep(D, \Sigma)$.

Example 3. Consider the database D and the set Σ of TGDs and NCs given in Example 1, and also the Boolean CQs

$$q_1 = faculty(\text{John}) \qquad q_2 = \exists X(teaches(\text{John}, X)),$$

where q_1 asks whether John is a faculty member, while q_2 asks whether John teaches a course. Recall that $rep(D, \Sigma)$ consists of the databases D_1 and D_2 given in Example 2. Clearly, $(D_i \wedge \Sigma) \models q_1$, for each $i \in \{1, 2\}$, and thus $(D \wedge \Sigma) \models_{AR} q_1$. However, even if $(D_1 \wedge \Sigma) \models q_2$, $(D_2 \wedge \Sigma) \not\models q_2$, and therefore $(D \wedge \Sigma) \not\models_{AR} q_2$. ■

In the sequel, we refer to the problem of consistent (Boolean) CQ answering under the AR semantics as AR-CQ answering.

3 Generic Complexity Results

We present two generic complexity results that demonstrate the tight connection between classical and consistent CQ answering. These results will automatically provide us with a (nearly) complete picture of the combined and data complexity of AR-CQ answering under the main classes of TGDs, enriched with NCs. Given a class $\mathbb{C}$ of TGDs, let $\mathbb{C}_\perp$ be the formalism obtained by combining $\mathbb{C}$ with arbitrary negative constraints.

3.1 Combined Complexity

We first focus on the combined complexity. Since we would like to understand how the complexity of our problem is affected when some key parameters are fixed, we also consider the following two variants of the combined complexity: (1) the bounded-arity combined complexity (ba-combined complexity), which is calculated by assuming that the arity of the underlying schema is bounded; and (2) the fixed-program combined complexity (fp-combined complexity), which is calculated by considering the set of TGDs and negative constraints as fixed. We show the following:

Theorem 1. *Assume that CQ answering under a class $\mathbb{C}$ of TGDs is $\mathcal{C}$-complete in (x-)combined complexity, where $x \in \{\mathsf{ba}, \mathsf{fp}\}$. Then, the ($x$-)combined complexity of AR-CQ answering under $\mathbb{C}_\perp$ is (1) Π_2^p-complete, if $\mathcal{C} = \text{NP}$; and (2) $\mathcal{C}$-complete, if $\mathcal{C} \supseteq \text{PSPACE}$ is a deterministic class.*

Proof (sketch). Fix a database D, a set $\Sigma \in \mathbb{C}_\perp$ of TGDs and NCs, and a CQ q. The problem of deciding whether $(D \wedge \Sigma) \not\models_{AR} q$ can be easily solved via a guess-and-check algorithm. We simply need to apply the following steps:

1. Guess an instance $D' \subseteq D$;
2. Verify that $D' \in \mathit{rep}(D, \Sigma)$; and
3. Verify that $(D' \wedge \Sigma) \not\models q$.

We can show that steps 2 and 3 are not harder than classical query answering, which implies that AR-CQ answering under $\mathbb{C}_\perp$ is in $\text{coNP}^{\mathcal{C}}$. Therefore, (1) If $\mathcal{C} = \text{NP}$, then we get a Π_2^p upper bound since $\text{NP}^{\text{NP}} = \Sigma_2^p$ and $\text{co}\Sigma_2^p = \Pi_2^p$; and (2) If $\mathcal{C} \supseteq \text{PSPACE}$ is a deterministic class, then we get a $\mathcal{C}$ upper bound since $\text{NP}^{\mathcal{C}} = \mathcal{C}$ and $\text{co}\mathcal{C} = \mathcal{C}$.

Regarding the lower bounds, the $\mathcal{C}$-hardness result, when $\mathcal{C}$ is deterministic class above PSPACE, follows immediately since CQ answering is a special case of AR-CQ answering. For the Π_2^p-hardness, we show, by a reduction from the validity problem of 2QBF formulas, that AR-CQ answering under a single negative constraint $\forall \mathbf{X}(\varphi(\mathbf{X}) \rightarrow \perp)$, where φ consists of two atoms and it uses a single ternary predicate, while the database and the query use only binary and ternary predicates, is already Π_2^p-hard. $\square$

3.2 Data Complexity

By providing a similar analysis as above, we can establish the following generic data complexity result:

Theorem 2. *Assume that CQ answering under a class $\mathbb{C}$ of TGDs is $\mathcal{C}$-complete in data complexity. Then, the data complexity of AR-CQ answering under $\mathbb{C}_\perp$ is (1) coNP-complete, if $\mathcal{C} \subseteq \text{PTIME}$; and (2) $\mathcal{C}$-complete, if $\mathcal{C} \supseteq \text{PSPACE}$ is a deterministic class.*

Let us say that AR-CQ answering under a single negative constraint of the form $\forall X(p(X) \wedge s(X) \rightarrow \perp)$ and a fixed query is already coNP-hard, which in turn implies the coNP-hardness result in Theorem 2. Actually, the latter is implicit [2, Example 5], and it can be shown by a reduction from a variant of UNSAT, called 2+2UNSAT, where each clause has two positive and two negative literals, where the literals involve either regular variables or the truth constant true or false.

4 From Classical to AR-CQ Answering

We now focus on the main decidable classes of TGDs, enriched with NCs, and we show that the complexity of AR-CQ answering can be obtained in a uniform way by exploiting our generic complexity theorems. Recall that the main (syntactic) conditions on TGDs that guarantee the decidability of CQ answering are the following: (1) guardedness [3], which guarantees the treelikeness of the underlying canonical models; (2)

	Combined	ba**-combined**	fp**-combined**	**Data**
Guarded	2EXPTIME	EXPTIME	NP	PTIME
Weakly-Guarded	2EXPTIME	EXPTIME	EXPTIME	EXPTIME
Sticky	EXPTIME	NP	NP	in AC_0
Weakly-Sticky	2EXPTIME	2EXPTIME	NP	PTIME
Acyclic	NEXPTIME	NEXPTIME	NP	in AC_0
Weakly-Acyclic	2EXPTIME	2EXPTIME	NP	PTIME

Table 1. CQ answering. All results are completeness results, unless stated otherwise.

	Combined	ba**-combined**	fp**-combined**	**Data**
Guarded	2EXPTIME	EXPTIME	Π_2^p	coNP
Weakly-Guarded	2EXPTIME	EXPTIME	EXPTIME	EXPTIME
Sticky	EXPTIME	Π_2^p	Π_2^p	coNP
Weakly-Sticky	2EXPTIME	2EXPTIME	Π_2^p	coNP
Acyclic	NEXP - P^{NE}	NEXP - P^{NE}	Π_2^p	coNP
Weakly-Acyclic	2EXPTIME	2EXPTIME	Π_2^p	coNP

Table 2. AR-CQ answering. A single complexity class in a cell refers to a completeness result, while two classes $\mathcal{C}_1$-$\mathcal{C}_2$ refer to $\mathcal{C}_1$-hardness and $\mathcal{C}_2$-membership.

stickiness [4], which ensures the termination of backward resolution; and (3) acyclicity, which guarantees the finiteness of the underlying canonical models. Interestingly, each one of the above conditions has its "weakly" counterpart: weak-guardedness [3], weak-stickiness [4] and weak-acyclicity [6], respectively. The complexity of CQ answering under the above classes of TGDs is summarized in Table 1. Clearly, Table 1 and Theorems 1 and 2 imply Table 2, apart from the (ba-)combined complexity for acyclic TGDs and NCs; let us briefly comment on this.

The (ba-)combined complexity of CQ answering under acyclic TGDs has to our knowledge never been explicitly studied; we show that is NEXPTIME-complete: the upper bound is obtained by a reduction to nonrecursive logic programming [5], while the lower bound by a reduction from a TILING problem [7]. Notice that Theorem 1 does not cover the cases where classical CQ answering is in a nondeterministic class above PSPACE. Nevertheless, by exploiting the guess-and-check algorithm discussed in the proof of Theorem 1, we obtain $coNP^{NEXPTIME}$ upper bound. It is implicit in [8] that $NP^{NEXPTIME} = P^{NE}$, and since P^{NE} is a deterministic class, $coP^{NE} = P^{NE}$. Consequently, AR-CQ answering under acyclic TGDs and NCs is in P^{NE} in (ba-)combined complexity; the NEXPTIME-hardness is inherited from classical query answering.

5 Conclusions

In this work, which is a short version of [10], we performed an in-depth complexity analysis of the problem of consistent query answering under the main decidable classes of TGDs, focussing on the AR semantics. Notably, generic complexity results have been established, which allowed us to obtain a (nearly) complete picture of the complexity of our problem in a systematic and uniform way. Regarding future work, apart from bridging the complexity gap for acyclic TGDs, we intend to perform a similar complexity analysis for other important semantics such as the IAR semantics, that is, a sound approximation of the AR semantics [9].

Acknowledgements. This work has been funded by the EPSRC grant EP/J008346/1. M.V. Martinez and G.I. Simari are partially supported by Proyecto PIP-CONICET 112-201101-01000. A. Pieris is also supported by the Austrian Science Fund (FWF): P25207-N23 and Y698.

References

1. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: PODS. pp. 68–79 (1999)
2. Bienvenu, M.: On the complexity of consistent query answering in the presence of simple ontologies. In: AAAI (2012)
3. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. J. Artif. Intell. Res. 48, 115–174 (2013)
4. Calì, A., Gottlob, G., Pieris, A.: Towards more expressive ontology languages: The query answering problem. Artif. Intell. 193, 87–128 (2012)
5. Dantsin, E., Voronkov, A.: Complexity of query answering in logic databases with complex values. In: LFCS. pp. 56–66 (1997)
6. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
7. Fürer, M.: The computational complexity of the unconstrained limited domino problem (with implications for logical decision problems). In: Logic and Machines. pp. 312–319 (1983)
8. Hemachandra, L.A.: The strong exponential hierarchy collapses. J. Comput. Syst. Sci. 39(3), 299–322 (1989)
9. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-tolerant semantics for description logics. In: RR. pp. 103–117 (2010)
10. Lukasiewicz, T., Martinez, M.V., Pieris, A., Simari, G.I.: From classical to consistent query answering under existential rules. In: AAAI (2015)
11. Rosati, R.: On the complexity of dealing with inconsistency in description logic ontologies. In: IJCAI. pp. 1057–1062 (2011)

Finding Similar Products in E-commerce Sites Based on Attributes

Urique Hoffmann, Altigran da Silva, and Moisés Carvalho

Instituto de Computação
Universidade Federal do Amazonas
Manaus, Brazil
{uhsa,alti,moises}@icomp.ufam.edu.br

Abstract. We present a preliminary study on the problem of finding products similar to a product given as input, based solely on their attributes. We assume that we are given a set of products from a same category of a same on-line store, were each product is described in a catalog by a number of attributes (e.g., general characteristics, technical specifications, etc.). This problem, which at a first glance may be seen as straightforward or even mundane, is in fact challenging and intriguing. In fact, any automatic solution for it requires techniques for comparing tens of different atributes, whose semantics are often very technical and specific (e.g., the shutter speed of a camera) and also requires dealing with hundreds of products in the category. To be generic, such a solution must also deal with several distinct product categories. In here, we describe and evaluate a similarity function we have proposed for comparing products based on their attributes. This function uses a number of attribute-specific similarity functions, which are selected according to a class assigned to the attribute. The assignment of classes to attributes is carried out by a simple classification strategy, which we also describe and evaluate. Experiments we carried out to evaluate our proposed similarity function using data from real catalogs in five distinct popular product categories have shown promising results.

Keywords: Similarity Functions, E-Commerce, Recommender Systems

1 Introduction

Recommendation Systems are used by most e-commerce sites to suggest products to their users and provide additional information to help customers to decide which products to acquire [8]. Products can be recommended based on several different types of information such as top overall sellers on a site, customer's demographics, customer's past buying behaviour, or product attributes, e.g., technical specifications, general characteristics, brand, etc. [8]. Recommendations based on this last type of information are called *content-based* or *knowledge-based* recommendations [3].

A simple way of enabling content-based recommendation is, given a product, presenting to the user other products that are similar to it with respect to their attributes. This is useful, for instance, when costumers explicitly want to find products with certain characteristics or when the seller wants to present to a customer products similar to a product she is interest in, e.g., for the sake of comparison or to provide alternatives to out-of-stock itens.

However, in typical e-commerce sites, looking for similar products may require the user to browse manually through a large number of pages and products. For instance, suppose a user is interested in a specific camera, say, "Nikon S3500". Currently, if this user wants to find alternative cameras that are similar to this model (i.e., having similar features), for the sake of comparing their prices, it is likely that she would have to browse over hundreds of other cameras in the catalog to find them. On the other hand, if this camera is not in stock, it would be interesting to provide the user with similar alternative cameras in stock, without having her to look over the whole catalog.

Another interesting aspect of this kind of recommendation is that it enables suggesting products to the customers without relying on historical data. It means that the system can recommend products and provide buying options even if a costumer is new to the system or if the item is new to the catalog.

To find whether two products are similar it is necessary to compare them. Products on e-commerce sites are often described by their *attributes*. It means that, to make a comparison between two products, it is necessary to compare their attributes. This can be unfeasible to be carried out manually by casual users on the Web.

For instance, in a certain e-commerce site, to verify whether the "Nikon S3500" camera is similar to another camera, say the "Sony W830", a user has the option of comparing the 26 atributes provided for the first camera with the corresponding attributes of the second cameras. The lists of attributes available for each camera in this site are presented in Figure 1. Notice that the second camera has only 18 attributes. Also, notice that many attributes are difficult to be compared, unless the user is an expert in the field.

In general, the same situation occurs in many other categories, that is, comparing products requires comparing tens of attributes, some of them with very specific semantics.

In this paper we present a preliminary study on the problem of finding products similar to a given product. We assume that we are given a set of products from a same category of a same on-line store, along with their attributes. For instance, one of the datasets used in our experiments comprises a set of 489 camera models under the *Cameras* category of a real on-line store.

Specifically, we describe and evaluate a generic similarity function we have proposed for comparing products based on their attributes. This function uses a number of attribute-specific similarity functions, which are selected according to a class assigned to the attribute. The assignment of classes to attributes is carried out by a simple but effective classification strategy, which we also describe and evaluate here.

An experimental evaluation we carried out and reported here has shown promising results. Our proposed similarity function showed to be accurate in finding similar products, achieving average F-1 values above 0.75 in 5 representative product categories we have tested. Also, our strategy for attribute classification has correctly classified most of the attributes from these categories.

Attribute	*Nikon S3500*	*Sony W830*
Brand	Nikon	Sony
Type of Camera	Compact	Digital Camera
Monitor/Display	2,7" LCD / TFT 230.000	2.7"-LCD TFT-Clear Photo LCD
Resolution	20,1	20,1
Internal Memory	25MB	27MB
Memory Cards	Yes	Yes
Compatible Memory Cards	SD, SDHC and SDXC	Memory Stick Duo, Memory Stick PRO Duo (High Speed)
Sensor	CCD 1/2, 3 inch.	Super HAD CCD
Optical Zoom	7x	8x
Digital Zoom	4x	32x
Lenses	Crystal NIKKOR 26-182mm fixed	-
Shutter Speed	1/2000 - 1 s 4 s	-
Focus range	[W]: Aprox. 50 cm/[T]: Aprox. 1 m . . .	-
Opening	f/3.4-6.4	-
Flash Modes	Automatic TTL Flash with pre-flash monitor	Auto/On/Off/ Slow Syncro / Flash Extended
Flash range	[T]:1,0 to 2,1m (3 feet 4 inch. to 7 feet 1 inch.) . . .	ISO Auto: Aprox. 0.3m to 2.8m
Battery Type	Rechargeable Li-ion Battery EN-EL19	Battery Charger Adapter, Power Cable
Video Features	Full HD: 1920px1080p/30 / HD: 1280px720p/30 . . .	-
Scene modes	Backlight,. . .,Sports, Sunset	Sensitivity/Twilight/. . ./Pets
File Formats	.avi,.jpg,.wav	JPEG
Built-in microphone	Yes	-
Tripod mount	Yes	-
Menu Languages	Chinese,Danish,. . ., Arabic	-
Color	Purple	Violet
Dimensions (HxWxD)	5,7x9,6x2cm	9,31x5,25x2,25cm
Weight	129g	120g

Fig. 1. Attributes available for camera models "Nikon S3500" and "Sony W830" with their values. Some values were truncated to save space.

The remainder of this paper is organized as follows. In Section 2 we review related work. In Section 3 we present our strategy for attribute classification and in Section 4 we present our proposed similarity function. Section 5 reports our experiments and its results. Finally, Section 6 presents our conclusions and directions for future work.

2 Related Work

Although important and challenging, effective methods for finding similar products are scarce both in industry and in the academy.

Kagie et. al. [7] proposed a content-based graphical shopping interface based on product attributes to recommend similar products. To use this interface, the user must first define an *ideal* product by providing desired values to its attributes. The interface then shows products considered as similar to this ideal product in a 2D Map. By interacting with this map, the user chooses, from the products plotted, the most similar to the ideal. The interface then recalculates the similarity between the ideal product to all other products in the dataset. This process continues until the interface shows a product the user considers as the most similar. In this work the authors consider only two of attribute classes: categorical and numeric.

Our approach differs from this in many aspects. First, in our approach the user does not need to specify an ideal product. In fact, this is avoided, since we consider that casual users in e-commerce sites are not willing to specify desired values for several attributes. Instead, we only require the user to select one product to be used for comparison. Second, besides categorical and numerical attributes, we consider two additional classes of atributes: *multi-categorial* and *dimensional*. We adopted these two additional attributes classes because they are very common in e-commerce products. Third, in our case there is no need to ask the user to provide the class of each attribute involved in the comparison. Fourth, while Kagie's work seems to focused on a single category, our approach was conceived to deal with many categories typically found in e-commerce sites. Fifth, we instead of using a 2D map with several products, our approach can produce, as output, a ranking of products in order of similarity.

3 Attribute Classification

Prior to the application of our similarity function, it is necessary to take each attribute found in the products of a given category we are interested in and assign each one to a single class of a simple attribute taxonomy comprising four classes, namely: *Numerical*, *Categorical*, *Multicategorical* and *Dimensional*.

This taxonomy was created based on previous work by Kagie et. al. [6,7] and in our own experience in dealing with e-commerce catalogs. The original taxonomy by Kagie et. al. in [7] included only *Numerical* and *Categorical* attributes. It was extended in [6] to include the *Multicategorical* class. We further expanded it with the *Dimensional* class to handle the common case of atributes that describe the dimensions of products, displays, etc.

Although a number of different approaches could have been used for this task, we opted for using a simple strategy in which the values expected for the attributes in a given class are described by a regular expression we call *domain descriptors*. Domain descriptors are similar to the *Data Frames* used by Embley et. al. in several methods (e.g., in [1]) and provide a description on how values of attributes of the four classes above are written.

The classification of a attribute is carried out as follows. Let A_i be an attribute that occurs for products $p_1,\ldots,p_m$ in a given category. For instance, attribute *Scene Modes* occurs in the description of many products in the *Compact Cameras* category. First, for all products $p_j (1 \leq j \leq p)$, we take the value $v_{i,j}$ for A_i occurring in p_j.

Next, we perform several cleaning and standardization operations over set of values $v_{i,j}$ of A_i taken from products. These operations include duplicate values removal, white space and case normalization, among others. The result is a set of values $a_1,\ldots,a_m$ which we call the *occurrences* of A_i. Notice that by doing so we assume that all values of A_i have the same semantics in all p_j. For instance, we assume that the attribute *Scene Modes* has the same semantics in the description of all products in the *Compact Cameras* category.

Finally, we test each occurrence $a_1,\ldots,a_m$ against each domain descriptor ϵ_k (k=1,...,4) and associate atribute A_i with the atribute class C_k whose domain descriptor ϵ_k recognizes the majority of its occurrences.

Although simple, this classification procedure is very effective as we demonstrate in experiments we carried out and report later in this paper.

4 Similarity Function

Based on the general coefficient similarity proposed by Gower [5], we propose a similarity function for comparing products as the sum of all non-missing similarity scores s_{ijk} over the maximum number of attributes present in one of the products according to Equation 1.

$$\mathcal{S}_{ij} = \sum_{k=1}^{K} m_{ik} m_{jk} s_{ijk} / \max(\sum_{k=1}^{K} m_{ik}, \sum_{k=1}^{K} m_{jk}) \tag{1}$$

In this equation, similarity scores s_{ijk} are computed for every atribute A_k that has value for both products p_i and p_j being compared. Also, m_{ik} (m_{jk}) is 0 when the value for attribute A_k is missing for products p_i (p_j) and 1 when it is not missing.

The specific functions used for computing the similarity score s_{ijk} depend on the class of the attribute A_k. Recall from Section 3 that this class was already defined. For each one of the four atribute classes we defined an appropriate similarity function.

For the *Numerical* class, the similarity function is defined as the absolute difference between the values of the attribute in the two products, as shown in Equation 2.

$$s_{ijk}^{N} = 1 - \frac{|v_{ik} - v_{jk}|}{\max(v_{ik}, v_{jk}),} \tag{2}$$

where v_{ik} and v_{jk} are, respectively, the values of the attribute k for products p_i and p_j.

For the *Categorical* class, the similarity function is defined as

$$s_{ijk}^{C} = 1(v_{ik} = v_{jk}) \tag{3}$$

implying that objects having the same value get a similarity score of 1 and 0 otherwise.

For the *Multicategorical* class, the similarity function is computed using the Jaccard coefficient [4] between the sets of

$$s_{ijk}^{M} = \frac{|v_{ik} \cap v_{jk}|}{|v_{ik} \cup v_{jk}|} \tag{4}$$

In this case v_{ik} and v_{jk} denote the sets of individual categorical values composing the actual values. For instance, v_{ik} would be $\{$*Auto*, *On*, *Off*, *Slow Syncro*, $\ldots\}$ for the attribute *Flash Modes* in the camera *Sony W830* of Figure 1.

For the *Dimensional* class, the similarity function is the normalized euclidean distance over the dimension values, as described in Equation 5.

$$s_{ijk}^{D} = 1 - [\sum_{d=1}^{D}((v_{ik}^{d})' - (v_{jk}^{d})')^2]^{\frac{1}{2}} \tag{5}$$

where D is the number of dimensions found in the values of the attribute and v_{ik}^{d} is the value for dimension d in v_{ik} (the same applies to v_{jk}^{d}). For computing this function, each dimension is mean-centered and normalized using

$$(v_{xk}^{d})' = ((v_{xk}^{d}) - \mu^{d})/\sigma^{d} \tag{6}$$

μ^d and σ^d are, respectively, the mean and the standard deviation of the set of values of dimension p in all values of atribute k, for the products in the category.

As a final comment, it is worth noting that the general coefficient similarity proposed by Gower [5], and latter used by Kagie et. al in [6, 7], is unsuitable to deal with objects with few common attributes. For instance, if directly applied to the problem of comparing products, when two products have just one common attribute and this attribute have the same value in both products, the Gower similarity measure will assign the highest similarity score between these two products. Our function tries to overcome this problem by penalizing the score when the products have few common attributes, as defined in Equation 1.

5 Experimental Results

In this section we report the results of experiments we performed to evaluate the attribute classification strategy presented in Section 3, and the similarity function described in Section 4.

5.1 Experimental Setup

For the experiments, we have used five datasets provided by Neemu[1], a company that develops search and recommendation technology for major e-commerce sites in Brazil. These datasets comprise five different popular product categories, namely: *Cameras*, *Camcorders*, *Laptops*, *Smartphones* and *TVs*. The product

[1] `http://www.neemu.com`

descriptions available in these datasets often provide many attributes that are not related to the product characteristics themselves. For instance, attributes related to the packing of the products such as, packing dimension, package contents, etc., are very common. Thus, we disregarded these attributes in our experiments. In addition, we removed all atributes that are not found in at least 20% of the products in a given category. By doing so, we tried to increased the percentage of attributes that can be effectively compared to calculate the similarity between products.

Table 1 compares the number of attributes originally available in each dataset and the final number of attributes we considered in each category. Notice that, even though many attributes were removed, still the number of attributes considered is large to be handled manually by humans. This table also presents the number of distinct products available in each dataset.

Dataset	Products	Initial Attributes	Remaining Attributes
Cameras	489	178	28
Camcorders	41	69	26
Laptops	423	76	28
Smartphones	147	105	48
TVs	244	96	37

Table 1. Datasets used in the experiments with the number of products available and the number of attributes considered.

In Table 2, we present the number of attributes in each of the classes of our taxonomy. This classification was carried out manually to be used as a golden standard. Notice that the large majority of the attributes are categorical. This trend was observed in all categories. Also, a single dimensional attribute was available in each category,.

Dataset	Numeric	Categorical	Multicategorical	Dimensional
Cameras	5	16	6	1
Camcorders	4	14	7	1
Laptops	5	21	1	1
Smartphones	6	35	6	1
TVs	9	24	3	1

Table 2. Attributes from the datasets by class.

5.2 Attributes Classification

In Table 3, we summarize the results obtained with our attribute classification strategy. For this, we used the well known Precision, Recall and F-1 metrics. In this table, each line corresponds to the results obtained with attributes of a distinct classe, namely, "NUM" (*Numerical*), "CAT" (*Categorical*), "MCA" (*Multicategorical*) and "DIM" (*Dimensionall*).

Class	Cameras			Camcorders			Laptops			Smartphones			TVs		
	P	R	F	P	R	F	P	R	F	P	R	F	P	R	F
NUM	1.00	1.00	1.00	1.00	0.75	0.85	1.00	1.00	1.00	0.85	1.00	0.92	1.00	0.77	0.87
CAT	1.00	0.93	0.96	0.93	1.00	0.96	0.95	0.90	0.92	1.00	0.91	0.95	0.88	1.00	0.94
MCA	0.85	1.00	0.92	1.00	1.00	1.00	0.00	0.00	0.00	0.75	1.00	0.85	1.00	0.66	0.80
DIM	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Table 3. Experimental Results for Attribute Classification

As it can be notice, our strategy has correctly classified most of the attributes from all categories we tested. We obtained perfect classification in many cases and F-1 values equal or above 0.8 were obtained in all cases but one. This case is the *Multicategorical* class in the Laptops category, which has a single attribute (see Table 2), and our classification strategy missed it. In many cases, the value of some attributes eventually presented noise our cleaning operation was unable to identify and fix Nevertheless, we believe the small number of failures does not compromise the effectiveness of our strategy and, as will see next, did not harm the overall results of our method.

5.3 Similarity Measure Evaluation

Evaluating the effectiveness of the similarity measure we described in Section 4 proved to be a challenge by itself. Indeed, carrying out a thorough evaluation to obtain values of Precision, Recall and F-1 would require to compare hundreds of products, examining the values of tens of attributes, some of them very technical. Thus, we opted to evaluate our proposed similarity measure in a task close to its intended application. This task consists in taking a product given as input, using the similarity measure to compare this product with all others in the same category, and verifying if the k products deemed as the most similar are indeed similar to the input product, according to a human-based evaluation. The results are reported in terms of the precision considering these top-k answers, a metric often known as P@k. In our case we used $k = 5$, which is reasonable in terms of recommender systems.

For each of the five product categories, we randomly selected 10 products, which we refer to as *query products*, and, for each of them, we examine the 5 most similar products in the same category according to our similarity measure. Thus, a total of 250 pairs of products were manually evaluated. The results are presented in Figure 2.

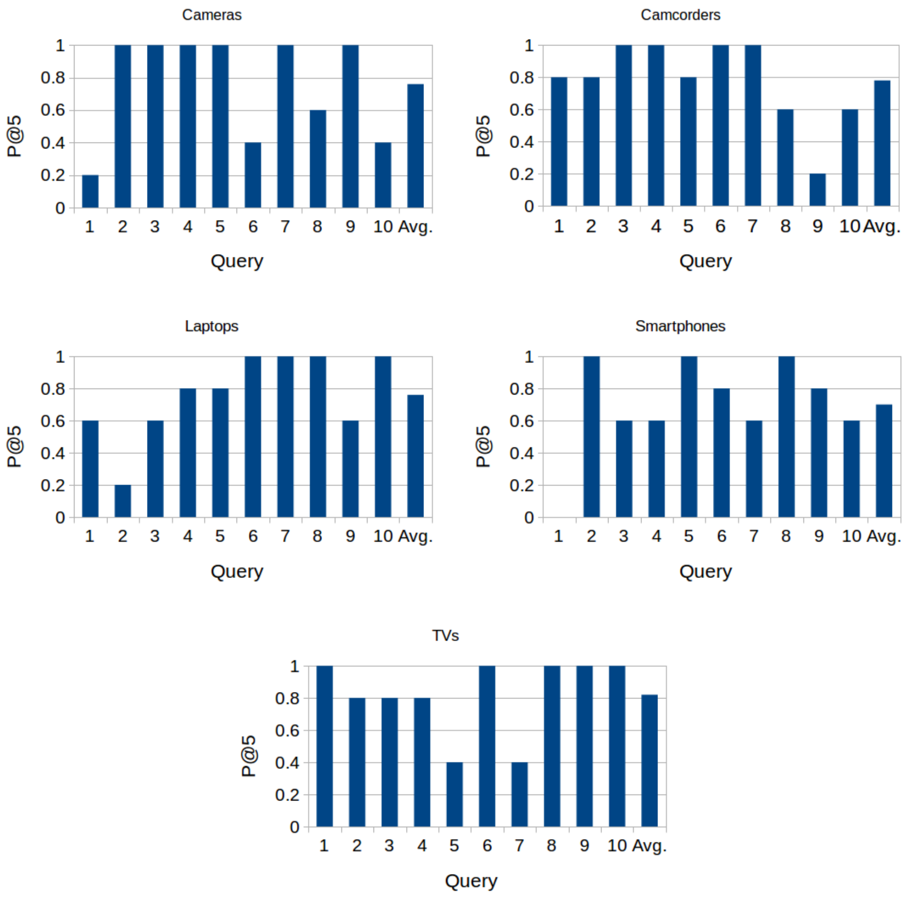


Fig. 2. Experimental Results for the Similarity Measure

In Figure 2, each graph corresponds to a product category and shows the P@5 values resulting from each of the 10 query products, along with the average of the ten values. Our similarity measure led to P@5=1 in 22 out of the 50 query products. Only in 8 cases, the P@5 values were below 0.5. In all categories, the average of P@5 values was around 0.75. An average above 0.8 was observed for the TVs category. Notice that the very low P@5 values obtained for some queries (e.g., 0 for query 1 in Smartphones or 1 for query 9 in Cancorders) does not necessarily implies that our similarity measure failed. For instance, it might happen that the query product has very few or none similar product in the catalog. In this case, our function just gave a low similarity score, but no similar products would appear among the top-5 answers. To solve this, a threshold on similarity score could be applied. However, there is no obvious way of imposing this threshold. Thus, we leave this study for future work.

6 Conclusions and Future Work

In this paper we presented a preliminary study on the problem of finding products similar to a product given as input. This problem, although important for e-commerce sites, has been ill addressed so far both in the industry and in the academy. We described and evaluated a similarity function we have proposed for comparing products based on their attributes. Our function is generic in the sense that it deals different types of attributes occurring in products from distinct categories. Prior to its application, the function requires that each attribute has been classified into to a class that determines an specific similarity function that handles this attribute. We demonstrate that this classification can be carry out by a simple but highly effective strategy we proposed, which relies of regular expressions. Experiments we have performed with our similarity function with datasests with real products, revealed that it is accurate in finding similar products, achieving average F-1 values above 0.75 in 5 representative product categories.

Our plans for future work address two main aspects. First, we are working on improving the effectiveness of our function by considering that different attributes may have different degrees of importante for users when comparing two products of a given category. Thus, we are investiganting ways for capturing this knowledge from the user and using it to improve our function. For this, we have been working on machine learning techniques, which require training from user data. Thus, the second aspect we are currently addressing is on how to obtain training data without requiring users to label instances specifically for this problem. Another interesting future work we plan to address is considering additional similarity functions for attributes. For instance, in the case of categorical data it is worth investigating the metrics studied in [2].

References

1. M. Al-Muhammed and D. Embley. Ontology-based constraint recognition for free-form service requests. In *IEEE 23rd International Conference on Data Engineering*, pages 366–375, April 2007.
2. S. Boriah, V. Chandola, and V. Kumar. Similarity measures for categorical data: A comparative evaluation. In *Proceedings of the SIAM International Conference on Data Mining*, pages 243–254, 2008.
3. R. Burke. Knowledge based recommender systems. In J. Daily, A.Kent, and H.Lancour, editors, *Encyclopedia of Library and Information Science*, volume 69. 2000.
4. S.-H. Cha. Comprehensive survey on distance/similarity measures between probability density functions. *International Journal of Mathematical Models and Methods in Applied Sciences*, 4(1):300–307, 2007.
5. J. Gower. A general coefficient of similarity and some of its properties. *Biometrics*, 27(4):857–874, 1971.
6. M. Kagie, M. van Wezel, and P. J. Groenen. Choosing attribute weights for item dissimilarity using clikstream data with an application to a product catalog map. In

Proceedings of the 2008 ACM Conference on Recommender Systems, pages 195–202, 2008.

7. M. Kagie, M. van Wezel, and P. J. Groenen. A graphical shopping interface based on product attributes. *Decision Support Systems*, 46(1):265 – 276, 2008.
8. J. Schafer, J. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1-2):115–153, 2001.

Entity Matching: A Case Study in the Medical Domain

Luiz F. M. Carvalho[1] and Alberto H. F. Laender[1] and Wagner Meira Jr.[1]

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Belo Horizonte, Brazil
{fernandocarvalho,laender,meira}@dcc.ufmg.br

Abstract. In this paper, we propose a simple and effective solution for the *entity matching* problem involving data records of healthcare professionals. Our method depends on three attributes that are available in most data sources in the medical domain: *name*, *specialty* and *address*. We apply a blocking technique to avoid comparisons, three matchers for conciliating the data records and a rule-based heuristic to combine the matchers. We performed experiments involving data from three Brazilian Web sources of healthcare professionals. Our results show that our solution is able to avoid unnecessary comparisons and provides good results.

Keywords: Entity Matching, Medical Records, Blocking, Matchers.

1 Introduction

With the increasing amount of data available on the Web, especially in social media networks and online catalogs of products and services, a recent challenge in Computer Science is the identification of data records related to the same real world entities. For example, a product offered on a website may appear in another one under a different name or description. Furthermore, multiple websites with similar functionality may offer different products. The conciliation of such products is a core task towards price monitoring [1, 14].

The task of disambiguation is also crucial in scenarios in which it is necessary to map all records related to a person considering her personal information such as name, occupation and address. In these cases, the task can be very complex due to existence of homonyms, incomplete data and multiple ways to represent the same information. Some examples of scenarios in which this task is performed are gathering all profiles of a same person in multiple social networks [19] and collecting all papers or co-authors of a same author [9, 21].

The task of integrating all records related to a given person is known as *entity matching* (also called *data matching*, *entity resolution* and *record linkage*) and can be defined as [15]:

Given two datasets A and B from two semantically related data sources S_A and S_B, the entity matching problem consists in finding all matches between data *records in $A \times B$ that refer to the same entity in the real world.*

A similar problem is *duplicate detection* in which the matching is performed over records from the same dataset [4, 5, 13].

Due to today's widespread use of medical management systems and the popularization of medical service websites, the medical domain is rich in applications that demand entity matching solutions for the integration of records of healthcare professionals. This task is important for many reasons that range from providing better search facilities to fraud detection.

In this paper, we propose a problem-oriented method for integrating data from healthcare professionals available on distinct Web sources, which is based on simple string matching strategies and adopts an overlapping blocking mechanism for reducing the number of comparisons between the records. For its assessment, we use data from three Brazilian Web sources: two general purpose healthcare professional directories, *Apontador*[1] and *Doctorália*[2], and the National Directory of Health Establishments maintained by the Brazilian Ministry of Health, *CNES*[3]. In summary, the main contributions of this paper are:

– A simple and effective solution for integrating data from healthcare professionals;
– A case study involving real data composed of 406,564 records collected from three distinct data sources;
– A detailed discussion over the main implementation issues and decisions involved in our method for entity matching in the medical domain.

Our method is generic for the medical domain and depends only on attributes that are usually available in most Web sources that provide information on healthcare professionals.

The rest of this paper is organized as follows. Section 2 provides some background on the core tasks involved in the entity matching problem. Section 3 presents our method and describes its implementation details. Section 4 discusses our experimental results. Finally, Section 5 presents our conclusions and some insights for future work.

2 Background

According to Christen [5], the process for solving the *entity matching* problem can be divided into five steps: (i) data pre-processing, (ii) indexing, (iii) record-pair comparison, (iv) classification and (v) evaluation. The first step involves the data preparation to ensure a standardized formatting for all involved data sources. The following three steps comprise the main tasks involved in the actual process of comparing the data records, whereas the last one consists of evaluating the quality of the results. Next, we briefly provide some background on

[1] *http://www.apontador.com.br*
[2] *http://www.doctoralia.com.br*
[3] *http://cnes.datasus.gov.br*

specific tasks, namely *blocking*, *matcher selection* and *matcher combination*, that support the three main steps of the entity matching process. For more details, we refer the reader to [5] and to some surveys on entity matching found in the literature [3, 8, 15].

Blocking. The purpose of blocking is to reduce the number of comparisons among the data records. It consists of defining a strategy to group the records in blocks so that only pairs of records in the same block are compared, avoiding the quadratic cost of comparing all pairs. A good blocking strategy minimizes the number of comparisons but does not separate related records into different blocks. A blocking strategy can either assign each record to just one block or set multiple blocks for each record (blocking with overlapping). In [11], the authors propose the *Sorted Neighborhood Method*, which is considered one of the first blocking methods described in the literature. The *Canopy Clustering* method [17] provides a clustering-based approach for blocking. For more details, Draisbach and Naumann [7] present a comparison of state-of-the-art blocking methods.

Matcher Selection. Matchers are algorithms that measure the similarity between a pair of records. There are two kinds of matchers: *context-based* and *value-based* matchers. Context-based matchers use structural information, such as graph distances, to define the similarity of a pair of records. A popular application of context-based matchers is the disambiguation of authors through co-authorship graph analysis, such as the solution proposed in [12]. On the other hand, value-based matchers establish a similarity degree between a pair of records by using only attribute values. The most popular value-based matchers are based on string comparison. For example, the *Fragment Comparison* algorithm [18] compares the components of author names as they appear in bibliographic citation records in order to decide whether they refer to the same author or not. This algorithm is robust to typing errors, abbreviations and variations in the sequence of name components. A comparison of string metrics for matching names and records is presented in [6].

Matcher Combination. This task consists in combining the values resulting from several distinct matchers to decide whether or not a pair of records is related to each other. The most popular solutions for combining matchers can be classified into three types: numerical, rule-based and flow-based. Numerical combinations apply a mathematical function to combine the involved matchers, as described in [10] and [20]. Rule-based combinations rely on logical operations and thresholds specifically defined for each case [2, 16]. Finally, flow-based combinations involve complex rules and many steps to perform the combination, like in the *MOMA* method [22].

We have implemented simple and effective solutions for each one of the above tasks, as described in the next section.

3 Proposed Method

In this section, we describe our solution for the entity matching problem in the medical domain. Our method relies on three specific attributes, *name*, *specialty* and *address*, of which *name* plays a major role in the matching process. We start by introducing the concept of *name relevance* that is central to our method. Then, we describe our solution for the three main tasks involved in the matching process discussed in the previous section: blocking, matcher selection and matcher combination. Finally, we discuss some strategies to reduce the execution cost of the whole process.

Name Relevance. To define the concept of name relevance, we consider that a person's name consists of several components. For example, the name "*luiz fernando magalhaes carvalho*" consists of four components: "*luiz*", "*fernando*", "*magalhaes*" and "*carvalho*". Informally, we can say that the relevance of a person's name depends on the discrimination power of each one of its components. Thus, let D be the set of datasets being matched and N the set of all names found in any dataset in D. For each name component c in N, its relevance is a value between 0 and 1 given by the ratio of its frequency and the frequency of the most frequent name component r in N, as expressed by:

$$relevance(c) = frequency(c)/frequency(r)$$

The relevance of a name C is given by the sum of the relevance of each of its components c_i. If the resulting value is greater than 1, it is set to 1, so that the relevance of each name is also a value between 0 and 1, as expressed by:

$$relevance(C) = \sum_{c_i \in C} relevance(c_i)$$

$$relevance(C) = \begin{cases} relevance(C), \; if \; relevance(C) < 1 \\ 1, \; otherwise \end{cases}$$

As we show next, the relevance of a name component is used to avoid unnecessary comparisons within a block whereas the relevance of a (full) name is used in the matcher combination.

Blocking. Blocking is a strategy to group records in blocks so that only records in the same block are compared. The blocking strategy of our method consists in creating one block for each name component found in any data source in D. As each block B is associated with a name component c, B contains all records that include a name with a name component c. Thus, this is a blocking strategy that implements an overlapping scheme, as each record is assigned to all blocks related to its name components, which means that each record is compared with all records that have at least one name component in common. This strategy is based on the assumption that if two records are related, they have at least one name component in common.

Matcher Selection. Having blocked all records, we need to use a matcher to measure the similarity of each record pair. For each one of the three attributes considered (*name*, *specialty* and *address*), we have implemented a matcher that returns a score between 0 and 1. Thus, for each record pair, three specific scores are produced: *name similarity score*, *specialty similarity score* and *address similarity score*. These scores are then combined in order to define whether the records are related or not. Since all three attributes are strings, their respective matchers are based on string comparison strategies empirically chosen for this specific purpose. Considering that the matching strategy for the attributes *specialty* and *address* are simpler, we describe them first.

Our matchers for the attributes *address* and *specialty* are based on the *Levenshtein* edit distance. Their returning scores have been defined as the complement of this metric, which measures the rate of changes that must be performed in two strings to make them identical. For example, given a pair of addresses $A1$ and $A2$, the corresponding *address similarity score* is computed as:

$$Levenshtein\_distance(A1, A2) = 1 - Levenshtein\_distance(A1, A2)$$

Unlike the address and specialty matchers, the name matcher is more complex and comprises four steps. Since names usually present typing errors, abbreviations, minor variations and missing components, we need a more sophisticated strategy to be able to properly match them. In addition, the first and last names are usually more reliable than the other names component and should be accordingly weighted. Thus, for our name matcher, we have developed a four step heuristic algorithm based on the *Fragment Comparison* algorithm [18] that satisfies all these conditions. The first three steps of the algorithm compare, respectively, the first name, the last name and the other names components of each pair of records, producing three similarity scores that range from 0 and 1. Then, the last step combines the three scores.

The first name and last name scores are similarly computed based on the *Levenshtein* edit distance. If the first (last) name in both records is not abbreviated, the first (last) name score is the *Levenshtein* similarity between them. On the other hand, if the first (last) name is abbreviated in one or both records, the first (last) name score is 1 if the first letter in both name components is the same or 0 otherwise. The similarity score for the other names components is computed as the rate of matched names among the remaining names. A match occurs when either two of the remaining names have a *Levenshtein* similarity above 0.8 in the case of non-abbreviated names or the first letters are the same in case of abbreviations. As the priority is the matching of non-abbreviated names, it is performed first, as shown in Algorithm 1.

Finally, after we have the individual scores for first name, last name and other names, the full name similarity score is computed as a linear combination of these three scores. We consider that the other names score is less relevant than the first and last name scores and set its weight to a lower value. We

Algorithm 1 Other names comparison step in the name comparison matcher.

```
$numberRemaining \leftarrow size(name1) + size(name2) - 4$
**for all** $x \in otherNames(name1)$ **do**
  **for all** $y \in otherNames(name2)$ **do**
    $threshold \leftarrow 0.8$
    **if** $(matched(x) == 0)$ $and$ $(matched(y) == 0)$ **then**
      **if** $(length(x) > 1)$ $and$ $(length(y) > 1)$ **then**
        **if** $Levenshtein(x, y) \geq threshold$ **then**
          $matched(x) \leftarrow 1$
          $matched(y) \leftarrow 1$
          $matches \leftarrow matches + 2$
**for all** $x \in otherNames(name1)$ **do**
  **for all** $y \in otherNames(name2)$ **do**
    **if** $(matched(x) == 0)$ $and$ $(matched(y) == 0)$ **then**
      **if** $(length(x) == 1)$ $or$ $(length(y) == 1)$ **then**
        **if** $firstLetter(x) == firstLetter(y)$ **then**
          $matched(x) \leftarrow 1$
          $matched(y) \leftarrow 1$
          $matches \leftarrow matches + 2$
$other\ names\ score \leftarrow matches\ /\ numberRemaining$
```

then use entropy to calibrate the weights of the first and last name scores, based on the intuition that a name component with larger uncertainty is more relevant.

Matcher Combination. After we have compared each pair of records in each block, we decide whether they are related or not based on the three similarity scores computed by the matchers and on the relevance of the corresponding full names. The solution proposed is a heuristic based on logical operations and thresholds that have been empirically defined. In the next section we show the details of the combination algorithm.

Pruning Comparisons. We also propose two heuristics for pruning unnecessary comparisons. The first heuristic consists in discarding blocks that are associated with low relevance name components. A threshold B defines the number of blocks to be discarded, i.e., the number of blocks for which no comparison is performed between their records. As stated before, each block is associated with a name component and for each such name we have computed its relevance. The B skipped blocks are those associated with names with small relevance. If a name is too popular, its respective block is large and the cost of comparing all its record pairs is huge. In addition, a popular name presents very low discrimination power, so it is likely that the matching rate within its block is very low. The second heuristic does not compare two records if the similarity score between two names is below a threshold S, that is, it does not compare the other two attributes, *specialty* and *address*. If the name similarity score produced by each matcher is smaller than S, it is very unlikely that the records are related, so it is not necessary to proceed with the remaining comparisons and the matcher combination. In the next section we describe how the values of B and S have been chosen.

4 Experimental Evaluation

4.1 Experimental Setup

Data Collected. We collected data about Brazilian healthcare professionals from three data sources: *Apontador*, *Doctorália* and *CNES*. After collecting the data, we extracted the attributes *name*, *specialty* and *address*. We selected only records of professionals from the state of Minas Gerais in order to reduce the data volume and make the calibration process easier. We believe that there is no significant difference between the data distribution among Brazilian states and our results should be closer to those from the whole country. Table 1 shows the total number of records collected from each data source.

Table 1. *Total number of records collected from each data source.*

Data Source	*Apontador*	*Doctorália*	*CNES*
Records Collected	14,060	10,324	382,180

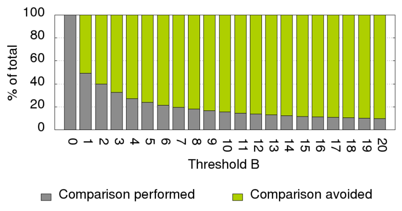
Blocking Parameters. The ideal value of B should minimize the number of comparisons but avoiding to separate related records that have in common only names associated with the largest B blocks. Therefore, if the value of B is too small, it does not avoid unnecessary comparisons. On the other hand, if the value of B is too large, some similar pairs may not be compared, although several comparisons are avoided. We then performed the following experiments.

First, we measured the amount of comparisons avoided by each value of B. Fig. 1(a) shows the percentage of comparisons avoided as a function of B considering that if B is equal to zero, no comparison is avoided, resulting in more than 11.6 billion comparisons. The results show that the rate of comparisons decreases fast initially, but the pace slows down in the range between $B = 5$ and $B = 14$. As it is not clear the stabilizing value, we chose the value $B = 5$ that avoids 76% of the original number of comparisons.

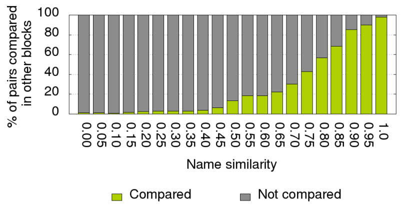
Next, we verified whether $B = 5$ was appropriate. Fig. 1(b) shows, for each similarity name range, the percentage of record pairs pruned for $B = 5$ that were also in another block and, therefore, are compared anyway. The results show that the larger the name similarity, the larger the number of comparisons. Thus, we can conclude that the threshold $B = 5$ prunes unnecessary comparisons and guarantees that pairs with related names are compared in non-pruned blocks.

According to Fig. 2(a), which shows the similarity distribution for the attribute *name*, the frequency of similar names decreases fast from 0.6, indicating that this is a good threshold to separate the actual similar names from random matches. Thus, S was set to 0.6.

Matchers and Combination Algorithm. As already described, for each pair of records compared, the matcher combination algorithm returns a binary value

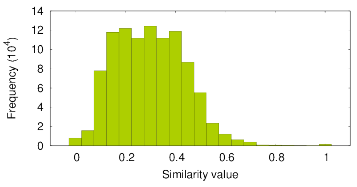


(a) Rate of comparison avoided

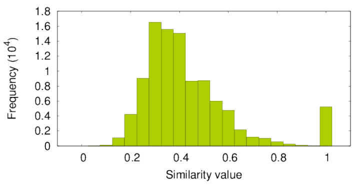


(b) Rate of skipped pairs compared in other blocks

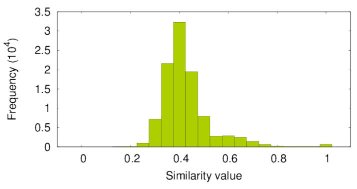
Fig. 1. Experiments results for choosing the value of B.



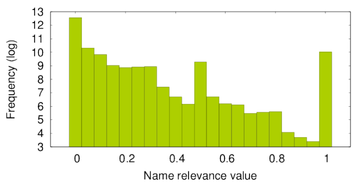
(a) Name similarity distribution



(b) Specialty similarity distribution



(c) Adress similarity distribution



(d) Full name relevance distribution

Fig. 2. Distribution of the scores considered in the algorithm for matchers combination.

indicating whether the two records match. Its implementation considers the similarity scores of the attributes *name*, *specialty* and *address*, and the relevance of the respective full names.

We start by defining the parameters for determining the matching score of a name. As already mentioned, we consider the first and last name components more relevant and thus set the weight of the other names component to 0.3. We then measure the entropy of the first and last name components, and set their weights based on the assumption that the higher their entropy, the higher their weight should be. We have found an entropy of 1.15 for the first name component and of 0.7 for the last name component. Thus, we proportionally set their weights to 0.43 and 0.27 respectively, resulting in the following expression for the name similarity score:

$$nameScore = ((0.43 \times scoreFirstName) + (0.27 \times scoreLastName) +$$
$$(0.3 \times scoreOtherNames))$$

Fig. 2(a)-(c) show the distribution of the similarity scores of the three attributes for a random sample of one million pairs. Note that for the attributes *specialty* and *address*, the sample also present a name similarity above 0.6. We also observe that the highest frequencies are associated with values smaller than the similarity threshold, showing the effectiveness of our heuristic. We have also measured the correlation between the matchers' returned values, but the results were smaller than 0.1, indicating that there was no correlation among them. As a consequence, the thresholds for each attribute type may be independently defined and set to a different value. We then present a histogram of the relevance scores for the 406,564 full names (Fig. 2(d)) extracted from our datasets. We can see that the threshold for name relevance should be lower to avoid missing relevant pairs, but we still have a large number of irrelevant pairs associated with lower scores.

Our heuristic strategy was empirically set by sampling the results. It is based on logical operators and thresholds, and considered the following assumptions:

- We divided all full names in the dataset into two groups: relevant names and non-relevant names. We consider a name as relevant if its relevance value is equal or greater than 0.2. This threshold sets 15% of the names as relevant.
- We set a threshold for each one of the three matchers in order to separate record pairs with similar attribute values.
- We set specific thresholds for the attribute *name* considering the cases in which the names are relevant and not relevant. For relevant names, the threshold is 0.8. Otherwise, it is set to 0.9.
- For the *specialty* and *address* attributes, the thresholds were set to 0.8 and 0.75, respectively.
- If the name similarity of a record pair is greater or equal than its threshold and the similarity of one of the other two attributes (*specialty* or *address*) also satisfies its respective threshold, the records are considered related, regardless of the similarity score of the third attribute.

Based on the above assumptions, Algorithm 2 below describes our implementation for the matcher combination.

Algorithm 2 Algorithm for matcher combination.

```
if $relevance(name1) \geq 0.2$ and $relevance(name2) \geq 0.2$ then
  if $nameSimilarity > 0.8$ and ($specialtySimilarity > 0.8$ or $addressSimilarity > 0.75$)
  then
    $MATCH$
else
  if $nameSimilarity > 0.9$ and ($specialtySimilarity > 0.8$ or $addressSimilarity > 0.75$)
  then
    $MATCH$
```

4.2 Experimental Results

In this subsection, we present a characterization of the records classified as matches and assess the accuracy of our method by sampling the results. In this execution of our method, with the thresholds $B = 5$ and $S = 0.6$, about 119.5 million comparisons have been performed, resulting in a total of 799,877 matched pairs, whereas approximately 118.6 million pairs have been classified as not matched.

Table 2 shows the percentage of pairs found by our method that present exact match for each attribute combination, i.e., they would be matched if an exact match approach were employed. For example, 21.5% of the pairs matched by our method present identical names and addresses, so they would be also matched by an exact match solution that considered only the attributes *name* and *address*. Moreover, only 2.1% of the related records have presented an exact match and more than 16,500 records (2.1%) do not have even the same name. Thus, the exact match approach would not be a good solution as many record pairs that have been matched (and are likely to be related) would not be matched.

Table 2. *Percentage of pairs of records matched by our algorithm that presents exact match for each set of attributes.*

Percentage of exact matches	97.9	71.3	21.6	69.7	21.5	2.2	2.1
Name	X			X	X		X
Specialty		X		X		X	X
Address			X		X	X	X

As we do not have a labeled dataset, we have sampled the dataset and labeled it manually. Since the number of not matched pairs is huge, it is not feasible to label a representative percentage of them. Thus, we have only labeled those records most likely to be incorrectly classified, i.e., we selected the 300,000 most similar pairs of unmatched records and manually labeled 1% of them. For the set of matched records, we have also selected the 300,000 least similar matched pairs and also manually labeled 1% of them. We have also assumed that the importance order of the attributes is *name*, followed by *specialty* and then by *address*. Thus, the similarity of a pair of records for the sample selection was computed as the combination value of the *name*, *specialty* and *address* similarities weighted by 5, 3 and 1, respectively. We notice that there are some pairs for which we have not been able to decide whether they were related or not. We refer to these pairs as unknown.

Table 3 shows the results evaluation considering the aforementioned sampling of 1% of the records most likely to be incorrectly classified. As we cane see, the results are quite good and, as the false negative rate is just 5%, our future efforts should aim those records that although related are not matched by our method.

Table 3. Results evaluation of the 6000 labeled pairs, 3000 from the matched set and 3000 from the not matched set.

		Real value			
Predicted value	Percentage of	Related	Not related	Unknown	Total
	Matched	95.31	1.43	3.26	100
	Not Matched	5.13	93.43	1.43	100

5 Conclusions

In this paper, we have proposed a simple and effective method for solving the entity matching problem involving data records of healthcare professionals. Our contributions are centered on the three main tasks involved in the matching process: blocking, matcher selection and matcher combination. Our blocking strategy adopts an overlapping mechanism for reducing the number of comparisons between related records. Our matchers are based on existing string matching functions especifically tuned for the problem at hand. Finally, our matcher combination strategy is based on an empirically designed heuristic algorithm. We have evaluated our method by conducting a case study involving data from three Brazilian Web sources of healthcare professionals. Our results show that our method has achieved a true positive rate of over than 95% and a true negative rate of over 93%.

As future work, we aim to improve our matchers by hierarchically labeling the medical specialties and splitting addresses into components such as street, number, city and state. We also aim to investigate a graph-based approach as an alternative to address the entity matching problem in the medical domain. Furthermore, we want to investigate the problem of data fusion, i.e., as in many situations the data related to a same entity might also present some conflict, it is very important to apply a method for deciding which information is correct and updated. Finally, we plan to compare our method with other existing entity matching approaches.

Acknowledgements. This work is partially funded by InWeb (grant MCT-CNPq 573871/2008-6), and by the authors' individual scholarships and grants from CAPES, CNPq and FAPEMIG.

References

1. Agrawal, R., Ieong, S.: Aggregating Web Offers to Determine Product Prices. In: Proc. of the 18th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. pp. 435–443 (2012)
2. Arasu, A., Ganti, V., Kaushik, R.: Efficient Exact Set-similarity Joins. In: Proc. of the 32nd Int'l Conf. on Very Large Data Bases. pp. 918–929 (2006)
3. Brizan, D.G., Tansel, A.U.: A Survey of Entity Resolution and Record Linkage Methodologies. Communications of the IIMA 6(3), 41–50 (2006)

4. de Carvalho, M.G., Laender, A.H.F., Gonçalves, M.A., da Silva, A.S.: A Genetic Programming Approach to Record Deduplication. IEEE Trans. Knowl. Data Eng. 24(3), 399 – 412 (2012)
5. Christen, P.: Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Springer-Verlag (2012)
6. Cohen, W., Ravikumar, P., Fienberg, S.: A Comparison of String Metrics for Matching Names and Records. In: Proc. of the KDD Workshop on Data Cleaning and Object Consolidation. vol. 3, pp. 73–78 (2003)
7. Draisbach, U., Naumann, F.: A Generalization of Blocking and Windowing Algorithms for Duplicate Detection. In: Proc. of the Int'l Conf. on Data and Knowledge Engineering. pp. 18–24 (2011)
8. Elmagarmid, A., Ipeirotis, P., Verykios, V.: Duplicate Record Detection: A Survey. IEEE Trans. on Knowl. and Data Engineering 19(1), 1–16 (2007)
9. Ferreira, A.A., Gonçalves, M.A., Laender, A.H.F.: A Brief Survey of Automatic Methods for Author Name Disambiguation. SIGMOD Record 41(2), 15 – 26 (2012)
10. Hassanzadeh, O., Chiang, F., Lee, H.C., Miller, R.J.: Framework for Evaluating Clustering Algorithms in Duplicate Detection. Proc. of VLDB Endow. 2(1), 1282–1293 (2009)
11. Hernández, M.A., Stolfo, S.J.: The Merge/Purge Problem for Large Databases. SIGMOD Record 24(2), 127–138 (May 1995)
12. Kang, I.S., Na, S.H., Lee, S., Jung, H., Kim, P., Sung, W.K., Lee, J.H.: On coauthorship for author disambiguation. Information Processing and Management 45(1), 84 – 97 (2009)
13. Kolb, L., Thor, A., Rahm, E.: Dedoop: Efficient Deduplication with Hadoop. Proc. of VLDB Endow. 5(12), 545 – 550 (2012)
14. Köpcke, H., Thor, A., Thomas, S., Rahm, E.: Tailoring Entity Resolution for Matching Product Offers. In: Proc. of the 15th Int'l Conf. on Extending Database Technology. pp. 545–550 (2012)
15. Köpcke, H., Rahm, E.: Frameworks for entity matching: A comparison. Data and Knowledge Engineering 69(2), 197 – 210 (2010)
16. Lee, M.L., Ling, T.W., Low, W.L.: IntelliClean: A Knowledge-based Intelligent Data Cleaner. In: Proc. of the Sixth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. pp. 290–294 (2000)
17. McCallum, A., Nigam, K., Ungar, L.H.: Efficient Clustering of High-dimensional Data Sets with Application to Reference Matching. In: Proc. of the Sixth ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining. pp. 169–178 (2000)
18. Oliveira, J.W., Laender, A.H., Gonçalves, M.A.: Remoção de Ambiguidades na Identificação de Autoria de Objetos Bibliográficos. In: Anais do XX Simpósio Brasileiro de Banco de Dados. pp. 205–219 (2005), (In Portuguese)
19. Raad, E., Chbeir, R., Dipanda, A.: User Profile Matching in Social Networks. In: Proc. of the 13th Int'l Conf. on Network-Based Information Systems. pp. 297–304 (2010)
20. Singla, P., Domingos, P.: Entity Resolution with Markov Logic. In: Proc. of the Sixth Int'l Conf. on Data Mining. pp. 572–582 (2006)
21. Tan, Y.F., Kan, M.Y., Lee, D.: Search Engine Driven Author Disambiguation. In: Proc. of the 6th ACM/IEEE-CS Joint Conf. on Digital Libraries. pp. 314–315 (2006)
22. Thor, A., Rahm, E.: MOMA - A Mapping-based Object Matching System. In: Proc. of the Third Biennial Conference on Innovative Data Systems Research (2007)

Using Statistics for Computing Joins with MapReduce

Theresa Csar[1], Reinhard Pichler[1], Emanuel Sallinger[1], and Vadim Savenkov[2]

[1] Vienna University of Technology
{csar, pichler, sallinger}@dbai.tuwien.ac.at
[2] Vienna University of Economy and Business (WU)
vadim.savenkov@wu.ac.at

1 Introduction

The MapReduce model has been designed to cope with ever-growing amounts of data [4]. It has been successfully applied to various computational problems. In recent years, multiple MapReduce algorithms have also been developed for computing joins – one of the fundamental problems in managing and querying data.

The main optimization goals of these algorithms for distributing the computation tasks to the available reducers are the replication rate and the maximum load of the reducers. The HyperCube algorithm of Afrati and Ullman [1] minimizes the former by considering only the size of the involved tables. This algorithm was later enhanced by Beame et al. [3] to minimize the latter by taking into account also so-called "heavy hitters" (i.e., attribute values that occur particularly often). However, in contrast to most state-of-the-art database management systems, more elaborate statistics on the distribution of data values have not been used for optimization purposes so far.

Recently, several approaches for handling skew in the computation of joins have been proposed, improving the partitioning of the data using histograms or varying a cost model [6, 7], but there is still ample room for enhancements and optimization. In [5] a survey of recent approaches for dealing with the weaknesses and limitations of the MapReduce model can be found.

The goal of this paper is to study the potential benefit of using more fine-grained statistics on the distribution of data values in MapReduce algorithms for join computation. To this end, we investigate the performance of known algorithms [1, 3] in the presence of skewed data, and extend them by utilizing data statistics. We compare the original algorithms with a modified one that makes use of additional statistical measures. Our initial study shows that our approach can indeed improve existing methods.

2 Preliminaries

A MapReduce join computation consists of three basic phases. First, in the *Map-Phase*, a key-value is assigned to every tuple. In the *Shuffle-Phase*, the tuples are distributed among the reduce tasks (also called *reducers*) according to their key-values. In the final *Reduce-Phase*, each reducer performs the join on all its tuples. The theoretical foundations of MapReduce query processing have been laid among others by [1–3], based on the HyperCube algorithm outlined below.

The HyperCube Algorithm. Key-values for tuples are formed by concatenating the hashes of the join attributes. Consider the triangle join $R(A, B) \bowtie S(B, C) \bowtie T(C, A)$ in which all attributes A, B and C are join attributes. Key-values are triples (a_i, b_i, c_i) obtained by the respective hash functions h_a, h_b and h_c. A tuple is sent to all reducers that may have join candidates for it. For instance, the tuple $R(a_1, b_1)$ is sent to the reducers identified by keys of the form $(h_a(a_1), h_b(b_1), *)$ where $*$ matches any value in the range of h_c. We take $[1, a], [1, b]$ and $[1, c]$ to be the ranges of the respective hash functions h_a, h_b and h_c. The size the range is called the *share* of the attribute. The respective shares are thus a, b and c, and the total number of reducers equals the product of the shares: $k = abc$.

An important measure for the performance of the HyperCube algorithm is the replication rate. For instance, each R-tuple $R(a_i, b_i)$ is replicated c times, since it is sent to the reducers responsible for the keys $(h(a_i), h(b_i), 1), \ldots, (h(a_i), h(b_i), c)$. The replication rate for the triangle join is $rc+sa+tb$, where r, s and t are the sizes of the tables. In [1], shares are chosen in order to minimize the replication rate. The solution for the shares for the triangle query in the model of [1] is $a = \sqrt[3]{\frac{krt}{s^2}}$, $b = \sqrt[3]{\frac{krs}{t^2}}$ and $c = \sqrt[3]{\frac{kst}{r^2}}$. For the four-atom chain query $R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie U(D, E)$, the solutions for the shares are $b = d\sqrt{\frac{rs}{tu}}$, $c = \sqrt{\frac{st}{ru}}$ and $d = \sqrt{\frac{ku}{s}}$.

In [3], the shares are chosen to minimize the maximum load per reducer, that is, the maximum number of tuples sent to a single reducer. The shares are calculated as the solution to a linear program. In contrast to [1], the method in [3] also addresses the problem of skew by treating heavy hitters separately. Also the expected load and maximum load per server is analyzed in [3], and a lower bound for the maximum load per server is given.

3 An Empirical Study and the Need for Statistics

The goal of our study is to compare the performance of HyperCube-based algorithms. To this end, we investigate how the shares chosen by such methods influence the workload distributions among the reduce tasks, in particular the maximum load. The analysis was performed on two well-studied types of queries, namely the triangle query $(R(A, B) \bowtie S(B, C) \bowtie T(C, A))$ and the chain query of length four $(R(A, B) \bowtie S(B, C) \bowtie T(C, D) \bowtie U(D, E))$. In both cases, there are three join attributes.

Methods. Apart from known methods for computing shares, namely [1] (which we shall call `AU`) and [3] (which we shall call `BKS`), we next introduce baseline methods as well as weighted variants of `AU` that take into account additional statistics. To facilitate a fair comparison, the shares produced by each method are normalized in the following way: they are rounded to integer values in such a way that the product of the shares is as close as possible to the fixed number of reduce tasks k. Shares have to be at least 1 and at most k. For the naive method, we define shares `naive` $= (\sqrt[3]{k}, \sqrt[3]{k}, \sqrt[3]{k})$. The worst-case we identified, `worst` $= (k, 1, 1)$, will be omitted from charts to keep the differences between other methods visible. The nearly-worst-case methods we consider are defined as `share1` $= (2\sqrt[3]{k}, \frac{1}{2}\sqrt[3]{k}, \sqrt[3]{k})$ and `share2` $= (\sqrt{k}, \sqrt{k}, 1)$, respectively.

Weigthed AU. The shares computed using AU (or BKS) depend only on the sizes of the tables, but not on other statistics indicating, e.g., the degree of skew. A simple way to detect a distribution with high variability is using the *standard deviation*. The more elaborate *gini-coefficient* is a measure for the variability of a distribution. For an observation X with possible values $x_1, \ldots, x_n$ and relative frequencies $p_1, \ldots, p_n$, the gini-coefficient is $\sum_{i=1}^{n} p_i^2$. A gini-coefficient close to 1 means that the values are very unevenly distributed.

We define our method SD as a variant of AU with the following modification: Assume that a table T has size t and attributes A and B. Instead of t, we give to AU the weighted value $t \cdot sd(T.A) \cdot sd(T.B)$, where sd denotes the standard deviation of the attribute values. The Gini method is defined analogously. Finally, the variant SD2 of SD is defined by normalizing the standard deviation relative to the maximum attribute value, i.e., $sd2(T.A) := sd(T.A) \,/\, max(T.A)$. Note that standard deviation is only defined for numeric values (and our test scenarios use only numeric values). We leave the study of similar variations of BKS for future work.

Test Methodology. The experimental study was implemented using the programming language R (`http://cran.r-project.org/`). The goal of our study is to compute the work loads of all reducers, and derive in particular the maximum load and various other statistics based on the loads. Thus, we only implement the *Map-Phase* of the MapReduce process. To this end we compute the loads of the reducers and our presented statistics.

As databases for our test scenarios, we use randomly generated data sets where attributes are generated according to a variety of different distributions. For each such database, all methods (AU, BKS, ...) are applied 1000 times to the input tables to compute the shares. In each round, other (randomly generated) hash functions are used. Performing 1000 repetitions is done to be able to isolate the effect of the method (in particular, the chosen shares) from the effect of the exact hash function that is used.

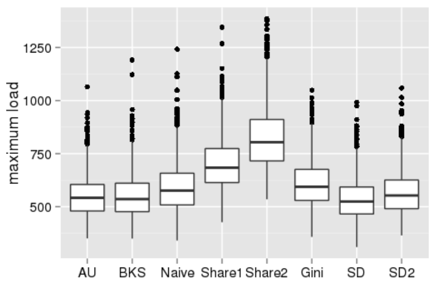


Fig. 1: Triangle query – maximum loads

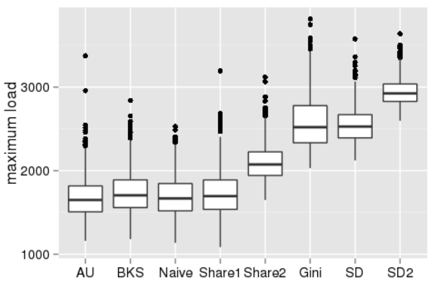


Fig. 2: Chain query – maximum loads

Triangle Query. For the triangle query, we first look at a sample database generated using the methodology described above. The number of reduce tasks used is 150. The resulting maximum load at the reduce tasks can be seen in Fig. 1, where it can be observed that all reasonable methods (i.e., all methods besides the nearly-worst-case ones) do not show any significant difference in the performance based on the the maximum loads. When observing the variance and the gini-coefficient of the loads, a similar picture arises. This is surprising, since the assigned shares differ a lot (see Table 1a). As expected, AU yields the lowest replication rate.

method	a	b	c	replication rate
AU	3	6	8	4.72
BKS	3	6	8	4.72
Naive	5	6	5	4.91
Gini	6	5	5	5.55
SD	3	5	10	4.82
SD2	2	6	11	4.73
share1	10	3	5	7.18
share2	8	9	2	7.18
worst	150	1	1	82.3

(a) First triangle query

method	b	c	d	replication rate
AU	8	3	6	13.3
BKS	11	2	7	13.5
Naive	5	6	5	14.5
Gini	48	1	3	25.9
SD	1	50	3	33.0
SD2	7	21	1	41.6
share1	10	3	5	14.4
share2	8	9	2	23.3
worst	150	1	1	75.6

(b) Chain query

Table 1: Shares and replication rates for the triangle and chain queries.

Triangle Query for Highly Skewed Data. For highly skewed data, we show that the AU and BKS methods do not always yield optimal maximum load. Indeed, the maximum load produced by AU and BKS exceeds the value obtained with the SD by more than 30%. We illustrate this by an example.

We consider the database instance D given in Fig. 3 and let the maximum number of reducers be 64. Table R contains 1040 distinct tuples (a_i, b_i) and the tables S and T contain groups of 16 $c_{i,j}$ values associated to the same a_i or b_j value, which sums up to 1040 values per table as well.

In total, few R tuples take part in triangles, but those that take part have 16 $c_{i,j}$ values that form triangles with them. Such a situation would be typical in a company where, say, few employees are department heads, but those who are department heads have a number of employees they are responsible for.

We calculated the shares, the resulting replication rate and maximum load for the discussed methods. Both AU and BKS yield shares $(4, 4, 4)$. Applying the pigeonhole principle, one can show that this leads to the load of $65 + 16 + 16 = 97$ tuples for *most* reducers as a lower bound. A much better maximum load (66 tuples for one reducer and 33 for the rest) could have been obtained using shares $(8, 8, 1)$. The suboptimal result of AU and BKS is due to taking only table sizes into account, whereas the SD method yields the solution $(8, 8, 1)$. An important observation here is that the actual performance of each method depends heavily on the concrete hash function and that "usual" hash functions based on integer division may by far miss the optimum.

R	
A	**B**
a_1	b_1
⋮	⋮
a_{1040}	b_{1040}

S	
A	**C**
a_1	$c_{1,1}$
a_1	$c_{1,2}$
⋮	⋮
a_1	$c_{1,16}$
⋮	⋮
a_{65}	$c_{65,1}$
a_{65}	$c_{65,2}$
⋮	⋮
a_{65}	$c_{65,16}$

T	
C	**B**
$c_{1,1}$	b_1
$c_{1,2}$	b_1
⋮	⋮
$c_{1,16}$	b_1
⋮	⋮
$c_{65,1}$	b_{65}
$c_{65,2}$	b_{65}
⋮	⋮
$c_{65,16}$	b_{65}

Fig. 3: Database instance D.

Chain Query. For the chain query, a random database is constructed according to the methodology outlined earlier. Again, our tests are performed for 150 reduce tasks. Interestingly, the resulting maximum loads are much higher for `share2`, `Gini`, `SD`, and `SD2` than for the other methods (see Fig. 2). The high maximum load in case of `Gini`, `SD`, and `SD2` suggests that some fine-tuning of the weights caused by the data statistics is needed. On the positive side, it turns out that the loads resulting from the `Gini`, `SD`, and `SD2` methods are distributed more evenly among the reducers than with `AU` and `BKS`, as can be seen in Fig. 5. The high variability in the median (Fig. 4), especially for the `Gini` method, again underlines that the choice of the hash function is crucial.

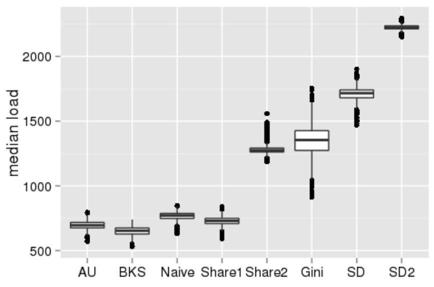


Fig. 4: Chain query – median of loads

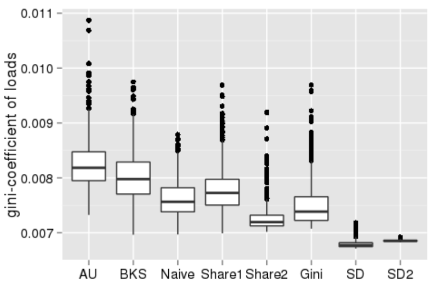


Fig. 5: Chain query – gini of loads

4 Conclusion

We have initiated the comparative study of methods for computing joins using MapReduce. We have seen that current methods perform relatively well compared to baseline and adapted methods. However, we have also seen that data-dependent statistics provide much potential for further improvement of these algorithms, which needs to be further explored. In particular, if we aim at a uniform distribution of computation tasks among the available reducers, taking into account additional statistical measures such as standard deviation or gini coefficient seems inevitable. Another important lesson learned from our investigation is the importance and difficulty of choosing an optimal hash function: even if the shares are – in theory – “optimal” for a certain criterion (such as maximum load), it is highly non-trivial to actually attain this optimum by choosing the “right” hash function. Current MapReduce research thus also has to be extended towards optimizing the hash function. Beyond that, we want to investigate the tradeoff between the cost of computing statistics and the gain provided by these statistics.

Acknowledgements. This work was supported by the Austrian Science Fund projects (FWF):P25207-N23 and (FWF):Y698.

References

1. Afrati, F.N., Ullman, J.D.: Optimizing multiway joins in a map-reduce environment. IEEE Trans. Knowl. Data Eng. 23(9), 1282–1298 (2011)
2. Beame, P., Koutris, P., Suciu, D.: Communication steps for parallel query processing. In: Proc. PODS 2013. pp. 273–284. ACM (2013)
3. Beame, P., Koutris, P., Suciu, D.: Skew in parallel query processing. In: Proc. PODS 2014. pp. 212–223. ACM (2014)
4. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM 51(1), 107–113 (2008)
5. Doulkeridis, C., Nørvåg, K.: A survey of large-scale analytical query processing in mapreduce. The VLDB Journal 23(3), 355–380 (2014)
6. Gufler, B., Augsten, N., Reiser, A., Kemper, A.: Load balancing in mapreduce based on scalable cardinality estimates. In: Proc. ICDE 2012. pp. 522–533 (2012)
7. Okcan, A., Riedewald, M.: Processing theta-joins using mapreduce. In: Proc. SIGMOD 2011. pp. 949–960. ACM (2011)

TriAL-QL: Distributed Processing of Navigational Queries*

Martin Przyjaciel-Zablocki[1], Alexander Schätzle[1], and Adrian Lange[2]

[1] Department of Computer Science, University of Freiburg, Germany
zablocki|schaetzle@informatik.uni-freiburg.de
[2] IIG Telematics, University of Freiburg, Germany
lange@iig.uni-freiburg.de

Abstract. Navigational queries are natural query patterns for RDF data, but yet most existing RDF query languages fail to cover all the varieties inherent to its triple-based model, including SPARQL 1.1 and its derivatives. TRIAL* is one of the most expressive approaches but not supported by any existing RDF triplestore. In this paper, we propose TRIAL-QL, an easy to write and grasp language for TRIAL*, preserving its procedural structure. To demonstrate its feasibility, we provide a proof of concept implementation for Hadoop using Hive as execution layer and give some preliminary experimental results.

1 Introduction

Graph databases and their respective graph query languages are commonly used for RDF data querying. However, in contrast to the standard graph model, an edge label in RDF (*predicate*) does not come from a finite alphabet and may also appear as a source or destination (*subject* and *object*, respectively) of another edge. Consequently, RDF query languages based on typical graph query languages like *nested regular expressions* (NRE) are not capable of such constructs and lose important features, e.g. reasoning over predicates within a query [2]. To the best of our knowledge, there are only two RDF query languages that enable expressive navigational capabilities with reasoning and can be evaluated in polynomial time, namely *Triple Query Language Lite* (TRIQ-LITE) [3] and *Triple Algebra with Recursion* (TRIAL*) [4]. TRIQ-LITE is defined as a Datalog extension that captures SPARQL queries enriched with the OWL 2 QL profile, whereas TRIAL* is a closed language that works directly with triples including recursion over *triple joins*. Although, the steady growth of Semantic Web data with its high degree of diversity in both, structure and vocabulary, justifies such expressive RDF query languages, it also raises the need for solutions that scale with the data size. In recent years, Hadoop has become the de-facto standard for processing Big Data, with a recent trend on scalable, interactive SQL-on-Hadoop solutions. The descent of TRIAL* from relational algebra and its inherent compositionality led us to the decision to build an implementation of

* A substantially extended version will appear in [5]

TriAL* based on Hadoop. While TriAL* is a neat approach for querying RDF, its algebraic notation is inappropriate for practical usage. Thus, we propose the TriAL* Query Language (TriAL-QL) that keeps the procedural structure of TriAL* by representing each algebra operation with a SQL-like statement. This way, even complex navigational queries are easy to grasp and write.

Our major contributions can be summarized as follows: (1) We introduce TriAL-QL, a query language for TriAL* with an intuitive mapping between both. (2) Next, we provide an Hadoop-based implementation of TriAL-QL using Hive. Optimizations include a precomputed 1-hop neighborhood, different evaluation strategies and a carefully chosen storage layout. (3) Finally, we show some preliminary experiments that demonstrate the scalability and feasibility of our approach.

2 TriAL-QL: A Procedural Query Language for RDF

The *Triple Algebra with Recursion* (TriAL*) [4] is one of the most expressive RDF query languages with polynomial complexity. In contrast to many other approaches, TriAL* is a closed and hence compositional language, i.e. the output is a set of triples rather than graphs or bindings. It works directly with triples, which allows us to write queries that are not expressible using query languages based on the standard graph model (e.g. *regular path queries* and *nested regular expressions*). TriAL* takes the relational algebra as its basis with some restrictions to guarantee closure. The most important operator is a *triple join* between two ternary relations E_1 and E_2 representing sets of triples, defined as: $E_1 \bowtie^{i,j,k}_{\theta,\eta} E_2$, where $i, j, k \in \{s_1, p_1, o_1, s_2, p_2, o_2\}$ indicate the implicit projection on three fields to keep the operation closed with s_1 referring to the subject of E_1, etc. θ represents the join conditions whereas η is a set of conditions between objects and data values. To express paths of arbitrary length, recursion is added with the *right* $(e \bowtie^{i,j,k}_{\theta,\eta})^*$ and *left* $(\bowtie^{i,j,k}_{\theta,\eta} e)^*$ *Kleene closure*, where e is a TriAL* expression. Thus, a reachability query that asks for pairs (x, z) which follow the connection pattern $\begin{tikzcd} \overset{x}{\bullet} \arrow[r, bend left, "\bullet"] & \bullet \arrow[r, bend left, "\bullet"] & \bullet & \cdots & \bullet \arrow[r, bend left, "\bullet"] & \overset{z}{\bullet} \end{tikzcd}$ corresponds to the TriAL* expression $(E \bowtie^{s_1,\ p_1,\ o_2}_{o_1=s_2})^*$ with E being a relation name in a triplestore (cf. [4] for more details).

TriAL-QL. Next, we introduce the notation of TriAL-QL. The basic idea is to flatten the algebra expressions of TriAL* to a sequence of interrelated statements. A complete grammar can be found on our project website [1]. Table 1 shows the algebra of TriAL* with the corresponding syntax in TriAL-QL. Each algebra operation is represented by exactly one SQL-like statement. Accordingly, we can express the previously discussed reachability query by the following TriAL-QL statement:

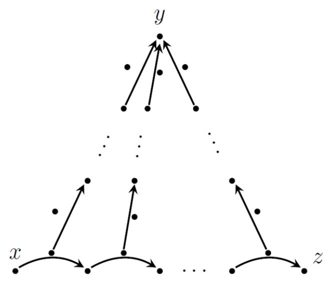
$$\text{SELECT } s_1,\ p_1,\ o_2 \text{ FROM } E \text{ ON } o_1 = s_2 \text{ USING } \textit{right}$$

[1] http://dbis.informatik.uni-freiburg.de/forschung/projekte/DiPoS/

Table 1. TriAL-QL Algebra & Syntax, where e, e_1 and e_2 correspond to a TriAL* expression. (i, j, k, θ, η as previously defined, cf. [4].)

Algebra (TriAL*)	**Syntax (TriAL-QL)**
$\sigma_{\theta,\eta}(e)$	SELECT i, j, k FROM e FILTER θ, η
$e_1 \bowtie^{i,j,k}_{\theta,\eta} e_2$	SELECT i, j, k FROM e_1 JOIN e_2 ON θ FILTER η
$e_1 \cup e_2$	e_1 UNION e_2
$e_1 - e_2$	e_1 MINUS e_2
$(e \bowtie^{i,j,k}_{\theta,\eta})^*$	SELECT i, j, k FROM e ON θ FILTER η USING *right*
$(\bowtie^{i,j,k}_{\theta,\eta} e)^*$	SELECT i, j, k FROM e ON θ FILTER η USING *left*

Next, we consider a more complex reachability problem introduced in [4] asking for pairs (x, z) which follow a connection pattern that requires reasoning capabilities:



TriAL*:
$e_1 = (E \bowtie^{s_1,\ o_2,\ o_1}_{p_1=s_2})^*$
$e_2 = (e_1 \bowtie^{s_1,\ p_1,\ o_2}_{o_1=s_2,\ p_1=p_2})^*$
$\Downarrow$
TriAL-QL:
$e_1 =$ SELECT $s_1,\ o_2,\ o_1$ FROM E
ON $p_1 = s_2$ USING *left*
$e_2 =$ SELECT $s_1,\ p_1,\ o_2$ FROM e_1
ON $o_1 = s_2,\ p_1 = p_2$ USING *left*

The inner expression e_1 computes the transitive closure of the predicates from E, while e_2 computes the transitive closure of this resulting relation. Again, we can see that TriAL-QL stays close to its TriAL* expression illustrating the strength of a compositional language where the result of the first statement can be used as input for the second. This makes TriAL-QL queries easy to write, understand and modify.

Further, new operators can be added smoothly, meeting our requirements. First, we extend the syntax of TriAL-QL with a STORE operation that enables us to materialize the result of a TriAL* expression e as a new relation in a triplestore. This way, not only the output but also intermediate results can be stored for later processing, if desired. Next, as we focus on processing web-scale RDF data, we have seen the need to introduce more control over the recursion depth of the *right* and *left Kleene* operator. Therefore, we introduce a scalar that replaces the *Kleene Star* and limits the number of join compositions. In TriAL-QL this can be formulated within the USING clause by writing, e.g., *left*(4) for applying *left Kleene* four times.

3 A Distributed Execution Engine for TriAL-QL

We implemented our execution engine on top of Hadoop using Hive as intermediate layer rather than working directly with MapReduce. This makes us independent of any Hadoop changes (e.g. Yarn) while taking advantage of continuously optimized Hive versions or newer execution engines like Tez that come along with the recent SQL-on-Hadoop trend. Due to very limited space restrictions, we give only a brief introduction to our implementation. In short, a TriAL-QL query is first parsed to generate an abstract syntax tree and next mapped to TriAL* which is in turn translated into HiveQL queries. We investigated different evaluation strategies based on (1) *linear* and (2) *nonlinear* recursion as introduced in [1] and performed exhaustive experiments with different storage formats (e.g. RCFile, ORC, Parquet) and strategies (e.g. indices, partitions) to identify best practices for storing RDF data in Hive. Further, a precomputed 1-hop neighborhood reduces the amount of required joins.

Experiments. Some preliminary results are illustrated in Figure 1. We used the *Social Intelligence Benchmark* (SIB) data generator[2] to create social networks of different sizes. The left hand side (a) shows execution times and number of resulting triples for an exemplary query with three joins and one set operation using linear evaluation. Both series exhibit an almost linear scaling behavior. The right hand side (b) compares the linear to the nonlinear evaluation on a more complex query involving the *Kleene Star*. In this example, the nonlinear evaluation is superior to the linear one with increasing data size as it reduces the amount of required join iterations from 12 (linear) to 8 (nonlinear). Exhaustive experiments that include more advanced evaluation strategies are needed for more comprehensive conclusions and are part of ongoing work [5]. However, the first preliminary results already demonstrate the scalability and feasibility of our approach.

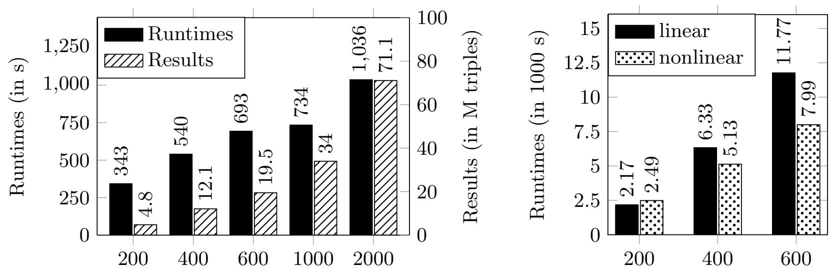


Fig. 1. (a) execution times vs. results (linear) (b) linear vs. nonlinear execution

References

1. Afrati, F.N., Borkar, V.R., Carey, M.J., Polyzotis, N., Ullman, J.D.: Map-reduce extensions and recursive queries. In: EDBT 2011, Sweden, March 21-24 (2011)
2. Angles, R.: A comparison of current graph database models. In: 28th ICDE Workshops, 2012, Arlington, VA, USA, April 1-5 (2012)
3. Arenas, M., Gottlob, G., Pieris, A.: Expressive languages for querying the semantic web. In: PODS'14, Snowbird, UT, USA, June 22-27, 2014 (2014)
4. Libkin, L., Reutter, J.L., Vrgoc, D.: Trial for RDF: adapting graph query languages for RDF data. In: PODS 2013, New York, NY, USA - June 22 - 27, 2013 (2013)
5. Przyjaciel-Zablocki, M., Schätzle, A., Lausen, G.: TriAL-QL: Distributed Processing of Navigational Queries. In: 18th WebDB 2015, Melbourne, Australia (2015)

[2] http://www.w3.org/wiki/Social_Network_Intelligence_BenchMark

On Axiomatization and Inference Complexity over a Hierarchy of Functional Dependencies

Jaroslaw Szlichta[1], Lukasz Golab[2], and Divesh Srivastava[3]

[1] University of Ontario Institute of Technology, Oshawa, Canada
jaroslaw.szlichta@uoit.ca
[2] University of Waterloo, Waterloo, Canada
lgolab@uwaterloo.ca
[3] AT&T Labs-Research, New Jersey, USA
divesh@research.att.com

Abstract. Functional dependencies (FDs) have recently been extended for data quality purposes with various notions of similarity instead of strict equality. We study these extensions in this paper. We begin by constructing a hierarchy of dependencies, showing which dependencies generalize others. We then focus on an extension of FDs that we call Antecedent Metric Functional Dependencies (AMFDs). An AMFD asserts that if two tuples have *similar* but not necessarily equal values of the antecedent attributes, then their consequent values must be equal. We present a sound and complete axiomatization as well as an inference algorithm for AMFDs. We compare the axiomatization of AMFDs to those of the other dependencies, and we show that while the complexity of inference for some FD extensions is quadratic or even co-NP complete, the inference problem for AMFDs remains linear, as in traditional FDs.

1 Introduction

Poor data quality is a bottleneck to effective business decision-making. Big data initiatives are likely to take longer, cost more, and deliver fewer benefits without clean data. The ability to store data is no longer a problem: according to a survey of 586 senior executives conducted in June 2011 by the Economist Intelligence Unit (EIU) [1], less than 20% indicated data storage as a problem; however more than 50% rated data management tasks such as cleaning as problematic.

With the interest in data analytics at an all-time high, data quality has become a critical issue in research and practice. Integrity constraints, which specify the intended semantics and attribute relationships, are commonly used to characterize and ensure data quality [6, 20]. In particular, Functional Dependencies (FDs), which have traditionally been used in schema design, have recently been extended for data consistency purposes. An FD asserts that if two tuples agree on the left-hand-side attributes, then they must also agree on the right-hand-side attributes. The idea behind various extensions of FDs is to replace strict equality with some notion of *similarity*, either on the left-hand-side (see, e.g,

Table 1: Movie Relation

source	title	length	year	director
A	A Beautiful Mind	135	2001	Ridley Scott
B	A Beaut. Mind	135	2001	Ridley Scott
C	Beautiful Mind	135	2001	Ridley Scott

Matching Dependencies [7, 5, 9]), on the right-hand-side (see, e.g, Metric Functional Dependencies [13] and Sequential Dependencies [11]), or on both sides of the dependency (see, e.g., Differential Dependencies [16]).

In this paper, we study these generalizations of FDs. Our first objective is to construct a hierarchy of dependencies, revealing which ones (strictly) generalize others, and comparing their axiomatization and complexity of inference.

We then introduce a particular generalization of FDs that we call Antecedent Metric FDs (AMFDs). An AMFD asserts that if two tuples have *similar* but not necessarily equal values of the antecedent attributes, then their consequent values must be equal; we will compare AMFDs with related dependencies in Section 2.

To illustrate the utility of AMFDs, consider the movie data set shown in Table 1, which was put together from multiple data sources. In the process of merging data from various sources, it is often the case that small variations occur. For example, one source might report the movie *A Beautiful Mind* to have a running time of *135* minutes, as shown in Table 1, while another source may refer to the same movie as *A Beaut. Mind* and the third one as *Beautiful Mind.* An AMFD $\{$*title, year, director*$\} \mapsto \{$*length*$\}$ indicates that movies with similar titles, years, and directors (up to some distance threshold, as we will discuss in Section 2) must have equal lengths. Of course, we assume that the semantics are such that two similar movie titles, made in similar years, by similar director names do in fact refer to the same movie.

An FD $\{$*title, year, director*$\} \rightarrow \{$*length*$\}$ would not require the three *length* values in Table 1 to be equal, even though they refer to the same movie. Thus, AMFDs generalize FDs and can express the additional semantics of similarity.

The inference problem is to determine whether a dependency is logically entailed by a set of dependencies. For FDs, the inference problem has been well studied in previous work [3, 4]. We prove that while AMFDs are more expressive than FDs and have a more complex axiomatization, their complexity of inference remains linear.

The contributions of this paper are as follows.

1. *Hierarchy:* we construct a hierarchy of dependencies, showing which ones generalize others and comparing their complexity of reasoning. Our hierarchy shows which dependencies are practical and which are hard to reason about, and suggests further research on identifying tractable extensions of FDs.
2. *FD extension:* we introduce AMFDs, which describe integrity constraints on tuples with similar attribute values and are useful in data cleaning.
3. *Axiomatization:* we present a sound and complete axiomatization for AMFDs. Axiomatization is a first necessary step to designing an efficient inference procedure. Our axiomatization reveals interesting insights about inference

Table 2: Notational Conventions

Relations

– A bold capital letter represents a relation schema: **R**. Italic capital letters near the beginning of the alphabet represent single attributes: A and B.
– A small bold capital letter in italic represents a relation (a table): **t**.
– Small italic letters near the end of the alphabet denote tuples: s and t.
– Small italic letters near the beginning of the alphabet denote attribute values: a, b and c. A small italic letter m denotes a similarity metric.

Sets

– Italic capital letters near the end of alphabet stand for sets of attributes: X
– XY is shorthand for $X \cup Y$. Likewise, AX or XA stand for $X \cup \{A\}$.

rules over AMFDs. For instance, the Reflexivity and Augmentation axioms, which hold for traditional FDs, are not necessary true for AMFDs.

4. *Inference Procedure:* we develop an inference procedure for AMFDs that runs in linear time in the complexity of the schema[4]. We implemented the inference algorithm and experimentally verified its efficiency.

The remainder of this paper is organized as follows. In Section 2, we review previous work, formally define AMFDs, and present a hierarchy of dependencies. In Sections 3 and 4, we present a sound and complete axiomatization and an inference procedure for AMFDs, respectively, and we compare the axiomatization to those of other related dependencies. We conclude the paper in Section 5.

2 Fundamentals

2.1 AMFDs

We provide notational conventions in Table 2. To accommodate small variations in the attribute values on the left-hand-side of the dependency, we define AMFDs (Definition 2). This is a departure from traditional FDs which enforce equality on both sides. Before we define AMFDs, we first define a *similarity* operator with a distance threshold.

Definition 1. *(similarity) For every attribute A in a relational schema $\boldsymbol{R}$, we assume a binary similarity relation ($\approx_{m,\theta}$) w.r.t. some similarity metric m and a threshold parameter $\theta \geq 0$. Specifically, for two tuples s and t, $s[A] \approx_{m,\theta} t[A]$ iff $m(s[A],\ t[A]) \leq \theta$. Metric m satisfies standard properties; it is symmetric, satisfies the triangle inequality and identity of indiscernibles, i.e., $m(a,\ b) = 0$ iff $a = b$. For two tuples s, t in relation **t** over $\boldsymbol{R}$, we write $s[X] \approx_{\boldsymbol{m},\ \Theta} t[X]$ to mean $s[A_1] \approx_{m_1,\theta_1} t[A_1]$, ..., $s[A_n] \approx_{m_n,\theta n} t[A_n]$, where $X = \{A_1,\ ...,\ A_n\}$, $\boldsymbol{m} = [m_1,\ ...,\ m_n]$ and $\Theta = [\theta_1,\ ...,\ \theta_n]$.*

[4] Our inference procedure is efficient because it is done at the schema level, which is much smaller than the size of the data.

Next, we define AMFDs. By definition, AMFDs generalize FDs.

Definition 2. *(AMFD) Let X and Y be two sets of attributes, and let $\boldsymbol{m}_X$ and Θ_X be metrics and thresholds for attributes X. Then, $X \mapsto Y$ denotes an antecedent metric FD (AMFD), read as X metrically functionally determines Y. Let $\boldsymbol{R}$ be a relation schema that contains the attributes that appear in X and Y, and let $\mathbf{t}$ be a relation instance of $\boldsymbol{R}$. Relation $\mathbf{t}$ satisfies $X \mapsto Y$ ($\mathbf{t} \models X \mapsto Y$), iff for all tuples s, $t \in \mathbf{t}$, $s[X] \approx_{\mathbf{m}_X,\Theta_X} t[X]$ implies $s[Y] = t[Y]$. An AMFD $X \mapsto Y$ is said to hold for $\boldsymbol{R}$, written as $\boldsymbol{R} \models X \mapsto Y$, iff for each admissible relational instance $\mathbf{t}$ of $\boldsymbol{R}$, relation $\mathbf{t}$ satisfies $X \mapsto Y$. An AMFD $X \mapsto Y$ is trivial iff for all $\mathbf{t}$, $\mathbf{t} \models X \mapsto Y$.*

Example 1. (AMFD) Assume metrics m_{title} and $m_{director}$ are edit distances with thresholds $\theta_{title} = 6$ and $\theta_{director} = 0$, respectively, in Table 1 (movie relation). Also assume that metric m_{year} is an integer distance with a threshold $\theta_{year} = 0$. Therefore, Table 1 satisfies the AMFD $\{$*title, year, director*$\} \mapsto \{$*length*$\}$.

2.2 Related Work

AMFDs and traditional FDs are specified over a *single* relation. However, AMFDs replace strict equality on the *left-hand-side* of the dependency with similarity. Dependencies defined over a single relation with similarity on the *right-hand-side*, called Metric FDs, were proposed by Koudas et al. [13]. We call them Consequent Metric FDs (CMFDs) to distinguish them from AMFDs. The verification problem over CMFDs was studied in [13], which is to decide whether the instance satisfies a prescribed set of dependencies. However, axiomatization and inference were not considered.[5]

Bertossi et al. [5], Fan [7] and Fan et al. [9] studied Matching Dependencies (MDs), which are object-identification constraints across *multiple* relations. MDs also enforce similarity rather than equality on the left-hand-side, but allow arbitrary Boolean similarity functions. (These similarity functions only need to satisfy reflexivity, symmetry and subsumption of equality.) On the other hand, AMFDs are defined over a single relation and only allow a restricted notion of similarity, namely thresholds over similarity metrics (recall Definition 1). Fan et al. presented a sound and complete axiomatization[6] and a quadratic-time inference procedure for MDs.

Pointwise Order Dependencies (PODs) [10] consider order relationship rather than equality of attribute values. A relation satisfies a POD $X \hookrightarrow Y$ if, for all tuples s and t, for every attribute A in X, $s_A \; op \; t_A$ implies that for every attribute B in Y $s_B \; op \; t_B$, where $op \; \epsilon \{<, >, \leq, \geq, =\}$. For example, in relation

[5] Some of the authors of this paper solved the axiomatization and inference problems for CMFDs in a paper currently under submission.

[6] It is stated in Fan et al. [9] (without a proof of completeness) that a complete axiomatization for MDs consists of 11 axioms, but only 9 sound axioms are presented.

Table 3: TimePolls Relation

sequential_id	timestamp	date	year	month	day
1	20140201142320	20140201	2014	02	01
2	20140201142325	20140202	2014	02	02

TimePolls (Table 3), the POD $\{date^{>}\} \hookrightarrow \{year^{=}, month^{=}, day^{>}\}$ holds; however, the POD $\{date^{>}\} \hookrightarrow \{year^{=}, month^{=}, day^{\leq}\}$ does not hold. Ginsburg and Hull [10] present a sound and complete axiomatization for PODs and show that the inference problem for them is co-NP-complete.

PODs are defined over sets of attributes. On the other hand, Lexicographical Order Dependencies (LODs) are defined over lists of attributes [17, 19]. LODs describe the relationship among lexicographical orderings of sets of tuples. This is the notion of order used in SQL and in query optimization, as per the *order by* operator (nested sort). A relation satisfies a LOD $\mathbf{X} \hookrightarrow \mathbf{Y}$ if any list of its tuples that satisfy *order by* $\mathbf{X}$ also satisfies *order by* $\mathbf{Y}$; however, not necessarily vice versa. ($\mathbf{X}$ and $\mathbf{Y}$ denote lists of attributes.) For instance, in relation TimePolls, the LODs *timestamp* $\hookrightarrow$ *date* and [*date*] $\hookrightarrow$ [*year, month, day*] are true. The default direction of the SQL order by is ascending. This can be generalized to order-by's that mix *asc* and *desc* directions, e.g., *order by name asc, age desc.* For example, in relation TimePolls, the LOD [$-$*sequential_id desc*] $\hookrightarrow$ [*timestamp asc*] holds. Szlichta et al. present a sound and complete axiomatization for lexicographical order dependencies and show that the inference problem for LODs is co-NP-complete [17, 19].

Another constraint for ordered data, sequential dependencies (SDs), was introduced in Golab et al. [11]. For example, the SD *sequential_id* $\hookrightarrow_{[4,5]}$ *timestamp* means that after sorting the data by the attribute *sequential_id*, the *gaps* between consecutive timestamps are between 4 and 5. This particular SD holds in the *TimePolls* relation; however, the SD *sequential_id* $\hookrightarrow_{[6,7]}$ *timestamp* does not hold. Golab et al. present a framework for discovering which subsets of the data obey a given SD, but axiomatization and inference were not considered.

SDs were generalized in Song and Chen [16] by introducing gaps (differential functions) on both sides of the dependency and named Differential Dependencies (DDs). For instance, in the relation TimePolls, the DDs $sequential\_id^{[1,1]} \hookrightarrow timestamp^{[4,5]}$ and $\{date^{[0,1]}\} \hookrightarrow \{year^{[0,0]}, month^{[0,0]}, day^{[0,1]}\}$ hold. However, the DD $sequential\_id^{[1,1]} \hookrightarrow timestamp^{(5,6]}$ does not hold. Song and Chen present an axiomatization and show that inference problem for DDs is co-NP-complete.

2.3 Hierarchy of Dependencies

Figure 1 illustrates a hierarchy of the dependencies we discussed above as well as a new class: MDDs. MDDs strictly generalize MDs and DDs by allowing differential functions (on the left hand side and the right hand side) with arbitrary similarity functions and allowing multiple tables. Below each dependency name, we point out the complexity of inference. Observe that for the "not studied" dependencies, their complexity of inference is bookended by their immediate ancestors and descendants in the hierarchy.

We say that a dependency class $\mathcal{A}$ *generalizes* dependency class $\mathcal{B}$ *iff* there is a semantically preserving mapping of any dependency of class $\mathcal{B}$ into a set of dependencies of class $\mathcal{A}$. Class $\mathcal{A}$ *strictly generalizes* class $\mathcal{B}$ *iff* $\mathcal{A}$ generalizes $\mathcal{B}$, however, $\mathcal{B}$ does not generalize $\mathcal{A}$. The hierarchy in Figure 1 shows which dependencies strictly generalize others; due to space constraints, proofs will appear in extended version of this paper.

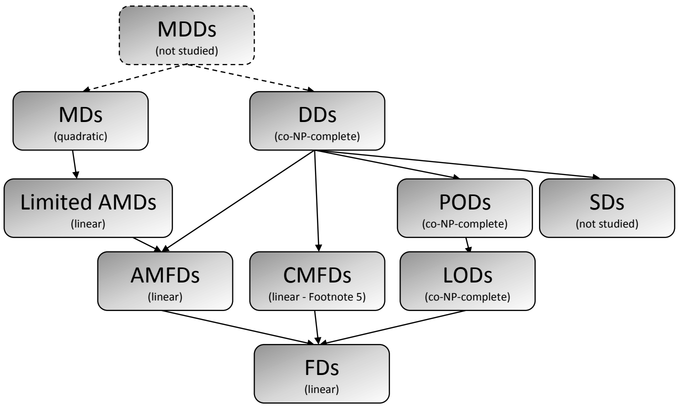


Fig. 1: Hierarchy of dependencies and their complexity of inference.

For example, DDs strictly generalize SDs. In our example involving Table 3, with the SD *sequential_id* $\hookrightarrow_{[4,5]}$ *time*, consecutive sequence numbers can be simulated by using on the left-hand-side of the DD a similarity metric which returns distance *one* if two numbers are consecutive and *zero* otherwise. Axiomatization and complexity of inference for SDs are open problems. However, since our hierarchy indicates that DDs strictly generalize SDs, the upper bound on the complexity of inference for SDs is co-NP complete.

Similarly, DDs strictly generalize PODs (which strictly generalize LODs [19]). For instance, a POD $A^{\geq} \hookrightarrow B^{\leq}$ is equivalent to a DD $A^{[0;+\infty]} \hookrightarrow B^{[-\infty;0]}$. This suggests that the complexity results for PODs can be adapted to DDs and did not have to be re-developed from scratch in [16]. AMFDs and CMFDs are also subsumed by DDs, since DDs allow similarity both on the left-hand-side and right-hand side. (Limited AMDs are introduced in Section 3.) Both AMFDs and CMFDs strictly generalize FDs by replacing equality with similarity.

3 Axiomatization

3.1 Soundness and Completeness

We now present an axiomatization for AMFDs, analogous to Armstrong's axiomatization for FDs [3, 4]. This provides a formal framework for reasoning about

1. *Void*
   $X \mapsto \{\}$
2. *Transitivity*
   If $X \mapsto Y$
   and $Y \mapsto Z$
   then $X \mapsto Z$

3 *Composition*
   If $X \mapsto Y$ and $Z \mapsto W$
   then $XZ \mapsto YW$

4 *Decomposition*
   If $X \mapsto Y$ and $Z \subseteq Y$
   then $X \mapsto Z$

5 *Reduce*
   If $XZ \mapsto Y$ and $X \mapsto Z$
   then $X \mapsto Y$

6 *Limited Reflexivity*
   If $Y \subseteq X$ and $\Theta_Y = 0$
   then $X \mapsto Y$

Fig. 2: Axiomatization for AMFDs.

AMFDs. The axioms give insights into how AMFDs behave and reveal how dependencies logically follow from others, which is not easily evident when reasoning from first principles. Also, a sound and complete axiomatization is necessary for an efficient inference procedure (see Section 4).

The axioms for AMFDs are presented in Figure 2. Recall that $\{\}$ denotes an empty set. Two of the axioms generate trivial dependencies that are always true: Void and Limited Reflexivity. Below we introduce additional inference rules that follow from the axioms in Figure 2. These will be used throughout the paper, particularly to prove that our AMFD axioms are complete.

Lemma 1. *(Left Augmentation) If $X \mapsto Y$, then $XZ \mapsto Y$.*

Proof. By Void and Composition it follows that $XZ \mapsto Y$. □

Lemma 2. *(Union) If $X \mapsto Y$ and $X \mapsto Z$, then $X \mapsto YZ$.*

Proof. By Composition it follows that $X \mapsto YZ$. □

Next, we define *closure* over AMFDs. The closure of a set of attributes X is the set of attributes that X logically determines given a set of AMFDs F.

Definition 3. *(Closure X^+) The AMFD-closure of set of attributes X, denoted X^+, w.r.t. a set of AMFDs F using axioms $I = \{1\text{–}6\}$ in Figure 2, is defined as, $X^+ = \{A \mid F \vdash X \mapsto A\}$.*

Lemma 3 tells us whether a dependency follows from F using our axioms.

Lemma 3. *(Closure for AMFDs) $F \vdash X \mapsto Y$, if and only if $Y \subseteq X^+$.*

Proof. Let $Y = \{A_1, ..., A_n\}$. Assume $Y \subseteq X^+$. By definition of X^+, $X \mapsto A_i$ for all $i \in \{1, ..., n\}$. Therefore, by the Union axiom, $X \mapsto Y$. To prove the other direction, suppose $X \mapsto Y$ follows from the axioms. For each $i \in \{1, ..., n\}$, $X \mapsto A_i$ by Decomposition, so $Y \subseteq X^+$. □

Theorem 1. *(Completeness) AMFD axioms are sound and complete.*

Proof. The soundness proof (if $F \vdash X \mapsto Y$, then $F \models X \mapsto Y$) is trivial. We just have to show that each axiom is true. We present the completeness proof (if $F \models X \mapsto Y$, then $F \vdash X \mapsto Y$). We consider a table $\mathbf{t}$ with two rows, whose template is shown in Table 4. We divide the attributes of $\mathbf{t}$ into three subsets: X_+, the set N, consisting of attributes in X that are not in the closure of X[7],

[7] For a traditional FD $X \mapsto Y$, by Reflexivity all the attributes in X are also in X^+. However, this is not true for AMFDs, as we will show in Example 2.

Table 4: Table template for AMFDs.

X^+	$N = \{A \mid A \in X \text{ and } A \notin X^+\}$	other attributes
$a...a$	$a...a$	$a...a$
$a...a$	$b...b$	$c...c$

and all the remaining attributes. All the attributes of the first row have the value a, while for the second row, the attributes in X^+ are a's, the attributes in N are b's and the other attributes are c's. Without loss of generality, assume that for all the attributes A in N and *other attributes* over $\mathbf{t}$ we use the same metric m, and that a and b are similar ($a \approx_{m,\theta_A} b$) but not equal. Also, assume that the values a and c are not similar ($a \not\approx_{m,\theta_A} c$), and hence not equal.

We first show that all dependencies in the set of AMFDs F are satisfied by table $\mathbf{t}$ ($\mathbf{t} \models F$). Since the AMFD axioms are sound, AMFDs inferred from F are true. Note that by Void and Limited Reflexivity, all trivial AMFDs are satisfied in table $\mathbf{t}$. Assume $V \mapsto Z$ is in F but is not satisfied by table $\mathbf{t}$. Therefore, $V \subseteq \{X^+ \cup N\}$ because otherwise two rows of $\mathbf{t}$ are not similar on some attribute of V since $a \not\approx_{m,\theta_A} c$, and consequently an AMFD $V \mapsto Z$ would not be violated. Moreover, Z cannot be a subset of X^+ ($Z \not\subseteq X^+$), or else $V \mapsto Z$ would be satisfied by $\mathbf{t}$. Let A be an attribute of Z not in X^+. Since the dependency $V \mapsto Z$ is in F, by Decomposition, $V \mapsto A$. Let V_1 be a maximal set of attributes such that $V_1 \subseteq V$ and $V_1 \subseteq X^+$. Let V_2 be a maximal set of attributes such that $V_2 \subseteq V$ and $V_2 \subseteq N$. By Union and Definition 3 of closure, $X \mapsto X^+$. Therefore, by Left Augmentation and Reduce, $XV_2 \mapsto A$. Since $N = \{A \mid A \in X$ and $A \notin X^+\}$, $V_2 \subseteq X$. Hence, $X \mapsto A$, which is a contradiction.

Our remaining proof obligation is to show that any AMFD not inferable from the set of AMFDs F with our axioms ($F \not\vdash X \mapsto Y$) is not true ($F \not\models X \mapsto Y$). Suppose it is satisfied ($F \models X \mapsto Y$). It follows by the construction of table $\mathbf{t}$ that $Y \subseteq X^+$; otherwise, two rows of table $\mathbf{t}$ agree or are similar on X but disagree on some attribute A from Y. Since $Y \subseteq X^+$, by Lemma 3 it can be inferred that $X \mapsto Y$, which is a contradiction. Thus, whenever $X \mapsto Y$ does not follow from F by the AMFD axioms, F does not logically imply $X \mapsto Y$. That is, the axiom system is complete over AMFDs, which ends the proof . □

3.2 Discussion

The axiomatization for AMFDs is more involved than its FDs counterpart. A sound and complete axiomatization for traditional FDs consists of only three axioms: Reflexivity, Augmentation and Transitivity. Interestingly, *Reflexivity* (if $Y \subseteq X$, then $X \mapsto Y$) is not necessary true for AMFDs.

Example 2. (lack of Reflexivity) Consider table $\mathbf{t}$ (Table 4). Assume again that the values a and b are not equal ($a \neq b$) but they are similar ($a \approx_{m,\theta_A} b$) for each attribute A in N. Let attributes $\{BCD\} \subseteq N$. Therefore, the AMFDs $BCD \mapsto BCD$ and $BCD \mapsto BC$ are not satisfied in $\mathbf{t}$ because the values are similar on the left hand side of the dependencies, but not equal on their right hand side.

Table 5: Comparison of Axiomatizations

Dependency Class	Axioms
DDs [16]	Extended Reflexivity, Extended Augmentation, Extended Transitivity, Impropriety
SDs [11]	N\A
PODs [10]	Reflexivity, Augmentation, Transitivity, Reversal, Disjunction, Total Order, Impropriety
LODs [17, 19]	Reflexivity, Transitivity, Augmentation, Suffix, Normalization, Chain
MDs [7, 5, 9]	9 sound axioms (out of 11) appear in [9]
limited AMDs [this paper]	Void, Transitivity, Composition, Decomposition, Reduce
AMFDs [this paper]	Void, Transitivity, Composition, Decomposition, Reduce, Limited Reflexivity
CMFDs [13]	Footnote 5
FDs [3, 4, 12]	Reflexivity, Augmentation, Transitivity

Similarly, Augmentation, which is another axiom for FDs, does not necessary hold for AMFDs. (Augmentation states that if $X \mapsto Y$ then $XZ \mapsto YZ$.)

We replaced Reflexivity with Void and Limited Reflexivity in the axiomatization for AMFDs. Lack of Augmentation forced us to add Composition and Decomposition to the axiomatization. Note that Left Augmentation (Theorem 1) holds for AMFDs. Since Reflexivity does not hold for AMFDs, we had to add Reduce. Only Transitivity (base axiom for FDs) was preserved in axiomatization.

We also studied an axiomatization for a simplified version of MDs over a single table, rather than multiple tables as originally defined in [5, 7, 9]. We call these limited AMDs. The main difference between limited AMDs and AMFDs is that the former allow arbitrary similarity functions while the latter employ thresholds on similarity metrics. A sound and complete axiomatization for limited AMDs consists of the following five axioms: Void, Transitivity, Composition, Decomposition and Reduce. The proof will appear in the extended version of this paper; it is a simplified version of the proof of Theorem 1. In comparison to AMFDs, Limited Reflexivity does not hold for limited AMDs. A sound and complete axiomatization for a full class of MDs is more complex (Footnote 6), as it incorporates axioms that allow us to reason over multiple relations.

Table 5 compares the axiomatization of AMFDs and AMDs with other dependency classes. We point out several interesting observations below.

In contrast to MDs and AMFDs, the distance (gap) functions for DDs are defined at the dependency level for each attribute instead of the schema level. Therefore, Transitivity for DDs additionally requires an order relation over differential functions. For instance, if we have the dependency "if the date difference for two tuples is $\leq$ 30 days, then price $\geq$ \$50", then the dependency "if the date difference for two tuples is $\leq$ 30 days, then price $\geq$ \$40" must also hold.

There is an extra axiom for DDs (Impropriety) that accommodates inconsistencies between dependencies (this problem does not arise in AMFDs and MDs). For example, the following two dependencies are inconsistent since it is not pos-

Algorithm 1 Inference procedure for AMFDs

Input: A set of AMFDs F, and a set of attributes X.
Output: The closure of X with respect to F.

```
1: $F_{unused} \leftarrow F$; $n \leftarrow 0$
2: $X^n \leftarrow W$ where $W = \{A \mid A \in X$ and $\theta_A = 0\}$
3: loop
4:   if $\exists\ V \mapsto Z \in F_{unused}$ and $V \subseteq \{X^n \cup X\}$ then
5:     $X^{n+1} \leftarrow X^n \cup Z$
6:     $F_{unused} \leftarrow F_{unused} - \{V \mapsto Z\}$
7:     $n \leftarrow n + 1$
8:   else
9:     return $X^n$
10:  end if
11: end loop
```

sible to instantiate a relation that satisfies both of them: a) if the date difference for two tuples is $\leq$ 30 days, then price = \$50; and b) if the date difference for two tuples is $\leq$ 30 days, then price $>$ \$50. Similarly, Augmentation and Reflexivity have to be modified for DDs to accommodate different distance functions used by different dependencies on the same attribute. For instance, different distance functions for the same attribute may result in the same actual distance.

Interestingly, as we traverse the hierarchy of dependencies, the number of axioms does not necessary decrease. There are 6 axioms for AMFDs, 7 for PODs and 6 for LODs versus 4 for DDs; however, there are 3 axioms for FDs at the bottom of hierarchy. There are two reasons for this. First, the axioms for DDs are quite complex. Second, as we go down the hierarchy, the dependencies become more specialized and therefore we may need more axioms to express their restricted semantics, e.g., lack of Reflexivity. As the dependencies become more generalized, some axioms must be weakened, e.g., Limited Reflexivity.

4 Inference Procedure

A goal of a dependency theory is to develop algorithms for the inference problem. Inference for DDs is co-NP-complete [16] and for MDs it is quadratic [7]. Since DDs and MDs generalize AMFDs, this sets an upper bound for the complexity of inference for AMFDs. However, computing closure, X^+, for AMFDs can be done more efficiently. It takes time proportional to the length of the dependencies in F, written out (linear time), which is as efficient as for FDs. (The complexity of inference for limited AMDs is also linear; the proof will appear in the extended version of this paper.) Algorithm 1 presents an inference procedure for AMFDs. Our experiments have shown that it is efficient. For 10 AMFDs prescribed over a dataset generated by the UIS Database [2], the algorithm runs in time $\leq$ 1ms.

Example 3. (inference) Let $F = \{AB \mapsto C, ABC \mapsto EG, EG \mapsto H\}$ denote the set of AMFDs. Also, let $\theta_C = 0$ and $\theta_D > 0$ for all attributes D in $ABEGH$ Let us calculate the closure of set of attributes AB with Algorithm 1:

1) $X^0 = \{\}$; 2) $X^1 = C$; 3)$X^2 = CEG$; 4) $X^3 = CEGH$.

The closure of AB is $CEGH$. For traditional FDs, the closure of AB is $ABCEGH$.

Theorem 2. *(inference) Alg. 1 correctly computes the closure X^+ over AMFDs.*

Proof. First we show by induction on k that if Z is placed in X^k in Algorithm 1, then Z is in X^+.
Base case: $k = 0$. By Limited Reflexivity, $X \mapsto W$, where $W = \{A \mid A \in X$ and $\theta_A = 0\}$.
Induction step: $k > 0$. Assume that X^{k-1} only consists of the attributes in X^+. Suppose Z is placed in X^k because $VW \mapsto Z \in F_{unused}$, such that $V \subseteq X^{k-1}$ and $W \subseteq X$. Since $V \subseteq X^{k-1}$, we know by the induction hypothesis that $V \subseteq X^+$. Hence, by Lemma 3, $X \mapsto V$. Therefore, since $XV \mapsto VZ$ by Composition, then by Reduce and Decomposition $X \mapsto Z$. Thus, Z is in X^+.

Now we prove the opposite: if Z is in X^+, then Z is in the set returned by Algorithm 1. Suppose Z is in X^+ but Z is not in the set returned by Algorithm 1. Consider table **t** similar to that in Table 4. Table **t** has two tuples that agree on attributes in X^n, are similar but not equal on attributes X that are not subset of X^n, and disagree on all other attributes. We claim that **t** satisfies F. If not, let $P \mapsto Q$ be a dependency in F that is violated by **t**. Then $P \subseteq X^n \cup X$ and Q cannot be a subset of $X^n \cup X$, if the violation happens. We used a similar argument in the proof of Theorem 1. Thus, by Algorithm 1, Lines 4–7, there exists X^{n+1}, which is a contradiction. □

5 Conclusions and Future Work

In this paper, we developed a hierarchy of dependency classes and laid out the theoretical foundations for AMFDs, which generalize traditional FDs. In future work, we plan to investigate the following problems.

- Determining whether a given AMFD holds on a given relation, and using AMFDs for data cleaning, similarly to how FDs were employed in previous data cleaning work [6, 7].
- Algorithms for automatic discovery of dependencies have been proposed for some dependencies, such as FDs and CFDs [8]. Similarly, we plan to study algorithms for discovering AMFDs.
- We plan to explore an inference framework for multiple dependencies. For example, the following inference rules hold: a) if an AMFD $X \mapsto Y$, then a CMFD $X \mapsto Y$; b) if an AMFD $X \mapsto Y$, then an FD $X \rightarrow Y$; c) if an FD $X \rightarrow Y$, then a CMFD $X \mapsto Y$. These rules along with the axioms for AMFDs (Figure 2) and CMFDs (Footnote 5), are *sound* for the integrated inference problem with FDs, AMFDs and CMFDs. However, an open question is if this rule set is *complete* and what is the complexity of the inference problem.
- Integrity constraints have been widely used in query optimization. For instance, FDs and LODs have been shown to be useful in simplifying queries

with *group by* and *order by* [14, 18, 17, 19] We believe that AMFDs can be used in similar ways to simplify SQL queries with similarity operators [15].

References

1. Economist Intelligence Unit analysis, http://www.economistinsights.com/technology-innovation/analysis/big-data.
2. UIS Data Generator, http://www.cs.utexas.edu/users/ml/riddle/data.html.
3. W. Armstrong. Dependency Structures of Database relationships. In *Proceedings of the IFIP Congress*, pages 580–583, 1974.
4. C. Beeri and P. Bernstein. Computional Problems Related to the Design of Normal Form Relational Schemas. *TODS 4(1):*, 4(1):30–59, 1979.
5. L. Bertossi, S. Kolahi, and V. Lakshmanan. Data cleaning and query answering with matching dependencies and matching functions. In *ICDT*, pages 268–279, 2011.
6. G. Beskales, I. F. Ilyas, and L. Golab. Sampling the repairs of functional dependency violations under hard constraints. *PVLDB*, 3(1):197–207, 2010.
7. W. Fan. Dependencies Revisited for Improving Data Quality. In *PODS*, pages 159–170, 2008.
8. W. Fan, F. Geerts, J. Li, and M. Xiong. Discovering Conditional Functional Dependencies. *TKDE*, 23(5):683–698, 2011.
9. W. Fan, X. Jia, J. Li, and S. Ma. Reasoning about Record Matching Rules. *PVLDB*, 2(1):407–418, 2009.
10. S. Ginsburg and R. Hull. Order dependency in the relational model. *Theoretical Computer Science*, 26(1–2):149–195, 1983.
11. L. Golab, H. Karloff, F.Korn, A. Saha, and D. Srivastava. Sequential dependencies. *PVLDB*, 2(1):574–585, 2009.
12. Y. Huhtala, J. Karkk, P. Porkka, and H. Toivonen. TANE: An Efficient Algorithm for Discovering Functional and Approximate Dependencies. *Computer Journal*, 42(2):100–111, 1999.
13. N. Koudas, A. Saha, A. Srivastava, and S. Venkatasubramanian. Metric Functional Dependencies. In *ICDE, 1291-1294*, 2009.
14. M. Malkemus, P. S., B. Bhattacharjee, L. Cranston, T. Lai, and F. Koo. Predicate Derivation and Monotonicity Detection in DB2 UDB. In *ICDE, 939-947*, 2005.
15. Y. N. Silva and S. Pearson. Exploiting database similarity joins for metric spaces. *PVLDB*, 5(12):1922–1925, 2012.
16. S. Song and L. Chen. Differential dependencies: Reasoning and discovery. *TODS*, 36(3):16, 2011.
17. J. Szlichta, P. Godfrey, and J. Gryz. Fundamentals of Order Dependencies. *PVLDB*, 5(11):1220–1231, 2012.
18. J. Szlichta, P. Godfrey, J. Gryz, W. Ma, P. Pawluk, and C. Zuzarte. Queries on dates: fast yet not blind. In *EDBT 497-502*, 2011.
19. J. Szlichta, P. Godfrey, J. Gryz, and C. Zuzarte. Expressiveness and Complexity of Order Dependencies. *PVLDB 6(14): 1858-1869*, 2013.
20. M. Volkovs, F. Chiang, J. Szlichta, and R. J. Miller. Continuous data cleaning. In *ICDE*, pages 244–255, 2014.

PPDL: Probabilistic Programming with Datalog

Balder ten Cate[1], Benny Kimelfeld[⋆2,1], and Dan Olteanu[3,1]

[1] LogicBlox, Inc., USA [2] Technion, Israel [3] University of Oxford, UK

1 Introduction

There has been a substantial recent focus on the concept of *probabilistic programming* [6] towards its positioning as a prominent paradigm for advancing and facilitating the development of machine-learning applications.[4] A probabilistic-programming language typically consists of two components: a specification of a stochastic process (the prior), and a specification of observations that restrict the probability space to a conditional subspace (the posterior). This paper gives a brief overview of *Probabilistic Programming DataLog* (PPDL), a recently proposed declarative framework for specifying statistical models on top of a database, through an appropriate extension of Datalog [1]. By virtue of extending Datalog, PPDL offers a natural integration with the database, and has a robust declarative semantics, that is, semantic independence from the algorithmic evaluation of rules, and semantic invariance under logical program transformations. It provides convenient mechanisms to allow common numerical probability functions as first-class citizens in the language; in particular, conclusions of rules may contain values drawn from such functions.

2 PPDL

The semantics of a PPDL program is a probability distribution over the possible outcomes of the input database with respect to the program. These outcomes are minimal solutions with respect to a related program that involves existentially quantified variables in conclusions. Observations are incorporated by means of logical integrity constraints. As argued in [1], the ability to express probabilistic models *concisely* and *declaratively* in a Datalog extension, with probability distributions as first-class citizens, is what sets PPDL apart from the wealth of literature on probabilistic Datalog [3], probabilistic databases [8,11], and Markov Logic Networks [4, 7, 10]. In the remaining of this section we introduce PPDL using an example program, and show how to interpret it probabilistically.

Our example is inspired by the burglar example of Pearl that has been frequently used for illustrating probabilistic programming. This example models a statistical process of alarming due to burglaries and earthquakes, and the goal is

[⋆] Taub Fellow – supported by the Taub Foundation

[4] An effort in this direction is led by DARPA's *Probabilistic Programming for Advancing Machine Learning* (PPAML) program.

House	
id	*city*
NP1	Napa
NP2	Napa
YC1	Yucaipa

Business	
id	*city*
NP3	Napa
YC1	Yucaipa

City	
name	*burglaryrate*
Napa	0.03
Yucaipa	0.01

ObservedAlarm
unit
NP1
YC1
YC2

Fig. 1. Database instance in the running example.

to estimate the likelihood that a given collection of alarms indicates these alarming events. Consider a database consisting of the following relations: House(h, c) represents houses h and their location cities c, Business(b, c) represents businesses b and their location cities c, City(c, r) represents cities c and their associated burglary rates r, and ObservedAlarm(x) represents units (houses or businesses) x where the alarm went off. These are the *EDB* relations that are not changed by the program. Figure 1 shows an instance over the schema. Now consider the PPDL program $\mathcal{P}$ in Figure 2, where some rules use the Flip distribution in their heads. The first rule states, intuitively, that for every fact of the form City(c, r), there must be a fact Earthquake(c, y) where y is drawn from the Flip (Bernoulli) distribution with the parameter 0.01. The fourth rule states that a burglary happens in a unit (house or business) with probability r, where r is a number that represents the rate of burglaries in the city of the unit (note that we represent by Burglary$(x, c, 1)$ and Burglary$(x, c, 0)$ the fact that a Burglary *did*, respectively, *did not* happen at unit x in city c; likewise for Earthquake and Trig). Finally, c1 is a constraint stating that Alarm and ObservedAlarm have the same tuples.

We now address the semantics of a program. What does it mean for a rule head like Earthquake$(c, \mathsf{Flip}[0.01])$ to be *satisfied*? What if this fact is derived by multiple, or even equivalent, rules? Do we need to sample more than once? The probabilistic semantics of the above PPDL program is established via an extension to Datalog, named Datalog$^{\exists}$, where rule heads can have existential quantifiers. Datalog$^{\exists}$ rules (a.k.a. *existential rules*, which are syntactically isomorphic to tuple-generating dependencies), have been used extensively in many areas, including data exchange [5] and ontological reasoning [2, 9]. Our PPDL

1. $\text{Earthquake}(c, \mathsf{Flip}[0.01]) \leftarrow \text{City}(c, r)$
2. $\text{Unit}(h, c) \leftarrow \text{Home}(h, c)$
3. $\text{Unit}(b, c) \leftarrow \text{Business}(b, c)$
4. $\text{Burglary}(x, c, \mathsf{Flip}[r]) \leftarrow \text{Unit}(x, c)\,,\ \text{City}(c, r)$
5. $\text{Trig}(x, \mathsf{Flip}[0.6]) \leftarrow \text{Unit}(x, c)\,,\ \text{Earthquake}(c, 1)$
6. $\text{Trig}(x, \mathsf{Flip}[0.9]) \leftarrow \text{Burglary}(x, c, 1)$
7. $\text{Alarm}(x) \leftarrow \text{Trig}(x, 1)$

c1. $\text{Alarm}(x) \leftrightarrow \text{ObservedAlarm}(x)$

Fig. 2. PPDL program $\mathcal{P}$ for Pearl's burglar example

1. $\exists y\ \text{Earthquake}_2^{\mathsf{Flip}}(c, y, 0.01) \leftarrow \text{City}(c, r)$
2. $\text{Unit}(h, c) \leftarrow \text{Home}(h, c)$
3. $\text{Unit}(b, c) \leftarrow \text{Business}(b, c)$
4. $\exists y\ \text{Burglary}_3^{\mathsf{Flip}}(x, c, y, r) \leftarrow \text{Unit}(x, c)\,,\ \text{City}(c, r)$
5. $\exists y\ \text{Trig}_2^{\mathsf{Flip}}(x, y, 0.6) \leftarrow \text{Unit}(x, c)\,,\ \text{Earthquake}(c, 1)$
6. $\exists y\ \text{Trig}_2^{\mathsf{Flip}}(x, y, 0.9) \leftarrow \text{Burglary}(x, c, 1)$
7. $\text{Alarm}(x) \leftarrow \text{Trig}(x, 1)$
8. $\text{Earthquake}(c, d) \leftarrow \text{Earthquake}_2^{\mathsf{Flip}}(c, d, p)$
9. $\text{Burglary}(x, c, b) \leftarrow \text{Burglary}_3^{\mathsf{Flip}}(x, c, b, p)$
10. $\text{Trig}(x, y) \leftarrow \text{Trig}_2^{\mathsf{Flip}}(x, y, p)$

Fig. 3. The Datalog$^\exists$ program $\widehat{\mathcal{P}}$ for the PPDL program $\mathcal{P}$ in Figure 2

program $\mathcal{P}$ gives rise to the Datalog$^\exists$ program $\widehat{\mathcal{P}}$ in Figure 3. Note that this program does not take into account the constraints. To illustrate the translation, note how rule 6 in $\mathcal{P}$ becomes rule 6 in $\widehat{\mathcal{P}}$. A special, *distributional* relation symbol $\text{Trig}_2^{\mathsf{Flip}}$ is created for Trig that captures the intention of the rule: whenever the premise holds (there is a Burglary at a unit x), then there exists a fact $\text{Trig}_2^{\mathsf{Flip}}(x, y, 0.9)$ where y is drawn from a Bernoulli distribution with parameter 0.9. Rule 10 is implicitly added to update Trig with the content of $\text{Trig}_2^{\mathsf{Flip}}$, where the additional parameter (i.e., the above 0.9) is projected out.

In the absence of constraints, a possible outcome of an input database instance I (a "possible world") is a minimal super-instance of I that satisfies the rules of the program. One possible outcome of the input instance in Figure 1 with respect to the rules of $\mathcal{P}$ is the database instance formed by the input instance and the relations in Figure 4. Each tuple of a distributional relation has a *weight*, which is the probability of the random choice made for that fact. For presentation's sake, the sampled values are under the attribute name *draw*. Ignoring the constraints (c1 in the example), the probability of this outcome is the product of all of the numbers in the columns titled "$w(f)$," that is, $0.01 \times 0.99 \times 0.03 \times \cdots \times 0.4$. Note that this multiplication is an instance of the *chain rule* $\Pr(A_1 \wedge \cdots \wedge A_n) = \Pr(A_1) \times \Pr(A_2|A_1) \times \dots$, and does not reflect an assumption of independence among the involved draws. One needs to show that this formula gives a proper probability space (i.e., the probabilities of all possible worlds sum up to 1). We do so via an adaptation of the *chase* procedure [1].

Constraints, such as c1 in our example, do not trigger generation of tuples, but rather have the semantics of conditional probability: violating possible worlds (where Alarm is different from ObservedAlarm) are eliminated, and the probability is normalized across the remaining worlds. Hence, we unify the concept of *observations* in Bayesian statistics with that of *integrity constraints* in databases.

In summary, a PPDL program associates to every given input instance a probability distribution over possible outcomes. One can then, for example, ask for the marginal probability of an event such as Burglary(NP1). Standard techniques

$\text{Earthquake}_2^{\mathsf{Flip}}$

city	*draw*	param	$w(f)$
Napa	1	0.01	0.01
Yucaipa	0	0.01	0.99

Earthquake

city	*draw*
Napa	1
Yucaipa	0

$\text{Burglary}_3^{\mathsf{Flip}}$

unit	*city*	*draw*	param	$w(f)$
NP1	Napa	1	0.03	0.03
NP2	Napa	0	0.03	0.97
NP3	Napa	1	0.03	0.03
YU1	Yucaipa	0	0.01	0.99

Alarm

unit
NP1
NP2

Burglary

unit	*city*	*draw*
NP1	Napa	1
NP2	Napa	0
NP3	Napa	1
YU1	Yucaipa	0

Unit

id	*city*
NP1	Napa
NP2	Napa
NP3	Napa
YU1	Yucaipa

$\text{Trig}_2^{\mathsf{Flip}}$

unit	*draw*	param	$w(f)$
NP1	1	0.9	0.9
NP3	0	0.9	0.1
NP1	1	0.6	0.6
NP2	1	0.6	0.6
NP3	0	0.6	0.4

Trig

unit	*draw*
NP1	1
NP2	1
NP3	0

Fig. 4. An outcome of the instance in Figure 1 with respect to the PPDL program $\mathcal{P}$.

from the probabilistic programming literature (analytical, as lifted inference, or sampling-based, as MCMC) can be used to answer such questions.

3 Discussion

Currently, PPDL semantics supports only discrete numerical distributions (e.g., Poisson). But even then, the space of possible outcomes may be uncountable (as possible outcomes may be infinite). We have defined a probability measure over possible outcomes by applying the known concept of *cylinder sets* to a probabilistic chase procedure. We have also shown that the resulting semantics is robust under different chases; moreover, we have identified conditions guaranteeing that all possible outcomes are finite (and then the probability space is discrete) [1]. The framework has a natural extension to continuous distributions (e.g., Gaussian or Pareto), though this requires a nontrivial generalization of our semantics. Additional future directions include an investigation of semantic aspects of expressive power, tractability of inference, and a practical implementation (e.g., corresponding sampling techniques).

Acknowledgements

We are thankful to Molham Aref, Vince Bárány, Todd J. Green and Emir Pasalic Zografoula Vagena for insightful discussions and feedback on this work. We are grateful to Kathleen Fisher and Suresh Jagannathan for including us in DARPA's PPAML initiative; this work came from our efforts to design translations of probabilistic programs into statistical solvers.

References

1. V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative statistical modeling with Datalog. *CoRR*, abs/1412.2221, 2014.
2. A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242, 2010.
3. D. Deutch, C. Koch, and T. Milo. On probabilistic fixpoint and Markov chain query languages. In *PODS*, pages 215–226, 2010.
4. P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Synthesis Lectures on AI and Machine Learning. Morgan & Claypool Publishers, 2009.
5. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, volume 2572 of *LNCS*, pages 207–224. Springer, 2003.
6. N. D. Goodman. The principles and practice of probabilistic programming. In *POPL*, pages 399–402, 2013.
7. G. Gottlob, T. Lukasiewicz, M. Martinez, and G. Simari. Query answering under probabilistic uncertainty in Datalog+/ ontologies. *Annals of Math.& AI*, 69(1):37–72, 2013.
8. B. Kimelfeld and P. Senellart. Probabilistic XML: models and complexity. In *Adv. in Probabl. Databases for Uncertain Information Management*, volume 304 of *Studies in Fuzziness and Soft Computing*, pages 39–66. 2013.
9. M. Krötzsch and S. Rudolph. Extending decidable existential rules by joining acyclicity and guardedness. In *IJCAI*, pages 963–968, 2011.
10. F. Niu, C. Ré, A. Doan, and J. W. Shavlik. Tuffy: Scaling up statistical inference in Markov Logic Networks using an RDBMS. *PVLDB*, 4(6):373–384, 2011.
11. D. Suciu, D. Olteanu, C. Ré, and C. Koch. *Probabilistic Databases*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

Implementing Graph Query Languages over Compressed Data Structures: A Progress Report

Nicolás Lehmann[1,2] and Jorge Pérez[1,2]

[1] Department of Computer Science, Universidad de Chile
[2] Chilean Center for Semantic Web Research
[nlehmann,jperez]@dcc.uchile.cl

Abstract. In this short paper we present our preliminary results on implementing two-way regular-path queries (2RPQs) over a compressed representation of graph data. We report on several experiments comparing our approach with state-of-the-art graph database engines. Our results are encouraging; although we use a naive implementation for 2RPQs, our system exhibits a competitive performance compared with other engines.

1 Introduction

Graph databases have recently gained a lot of attention in theory and practice. This can be explained by the current need of handling Web-related data, such as on-line social networks, and RDF and Semantic Web data. One of the most important challenges in this context, is the need of handling Web-scale amounts of data, while providing a reasonable expressiveness for users that want to explore this data.

In this short paper we present our preliminary results on implementing two-way regular-path queries (2RPQs) over a compressed representation of graph data. We use 2RPQs as they are an expressive language capable of navigating graphs by using paths defined by regular expressions, using forward and backward edges while navigating the graph. 2RPQs are in the core of recently proposed standards for handling RDF data [6]. To compress the graph data we use the k^2-tree data structure [4]. Given a node v, a k^2-tree representation allows us to access the neighbors of v, as well as the nodes pointing to v, in a very efficient way. This feature plus the compression ratio of k^2-trees which permits to maintain the structure of huge graphs in main memory, implied a critical performance gain when implementing 2RPQs.

We implement a naive algorithm for 2RPQs based on the typical automata-theoretic approach [8]. Even with this naive implementation, our system exhibits a competitive performance compared with state-of-the-art commercial engines. We perform several experiments with data generated by the GDBench tool [2], comparing our implementation with Sparksee [7] (formerly known as DEX) and Neo4j [9]. Our system and Sparksee show a similar performance and Neo4j is considerably surpassed by both alternatives. Our solution also exhibits a considerable advantage in a *cold scenario* where the structures have just been loaded and the system is running for the first time.

2 Background and implementation details

Graph databases and query languages We consider a simple model of a graph database as just a graph $G = (V, E)$ in which every element in V is a node ID (or just node for short), and each edge is a triple (v_1, e, v_2) where $v_1, v_2 \in V$ and e is an edge label from an alphabet Σ. We say that (v_1, e, v_2) is a *forward e-edge* from v_1 to v_2. Symmetrically, (v_1, e, v_2) is a *backward e-edge* from v_2 to v_1. As a query language, we consider two-way regular-path queries (2RPQs) which are essentially regular expressions over $\Sigma \cup \Sigma^-$, where $\Sigma^- = \{e^- \mid e \in \Sigma\}$ is the alphabet of *backward edges*. Given a 2RPQ r, a pair of nodes (v_1, v_2) is in the evaluation of r over G, if there exists a path in G from v_1 to v_2 following forward and backward edges, such that the sequence of labels of the path belongs to the regular expression defined by r considering each backward e-edge traversed as the symbol e^-. For example, consider the 2RPQ $r = a/(b^-)^*/c$ and a graph G with edges $(v_1, a, v_2), (v_3, b, v_2), (v_4, b, v_3), (v_4, c, v_5)$. Then we have that (v_1, v_5) is in the evaluation of r over G.

K^2-trees A k^2-tree [4] is a tree-shaped structure for representing graphs that exploits sparseness and clustering features of the adjacency matrix associated to the graph. Given an adjacency matrix, a k^2-tree divides it into k^2 submatrices of the same size. Each submatrix is represented in the tree as a child of the root. For the submatrices containing only 0's the decomposition ends there, using a single 0-node to represent the whole submatrix. The submatrices with at least one 1 are recursively decomposed using the same strategy until an actual cell in the matrix is reached, which is stored as a 0- or 1-node in the last level. The tree is then implemented in a highly compacted way using *bitstrings*; every level of the tree is represented as a *bitstring* and the whole tree as the concatenation of them. Given a node v, searching for the neighbors of v as well as for the nodes pointing to v, can be achieved by just traversing the k^2-tree [4]. The traversal of the tree can be simulated using *rank queries* over the *bitstrings*, which can be implemented very efficiently [5]. Thus, the whole k^2-tree can be represented in a succinct manner while maintaining its traversal properties. Further optimizations are possible, for example, using different values of k for different levels of the tree or stopping the decomposition when the matrices reach size $k_L \times k_L$ and use DACs to compress them [3].

Design and implementation details Let $G = (V, E)$ be a graph database over alphabet Σ. To simplify the correspondence between node IDs and rows and columns in an adjacency matrix representation, we first map every node ID in V and label in Σ to an integer via a dictionary encoding[3]. After the encoding, our design continues by *vertically partitioning* the data, reorganizing it into $|\Sigma|$ independent graphs, each graph containing only edges with a particular edge label. Then the whole graph is represented as an array of k^2-trees, each tree representing the graph induced by a particular edge label. Given a node v and an edge label e, we compute the direct or inverse e-neighbors of v by traversing the k^2-tree corresponding to e. Following the configuration of similar work [1], the k^2-trees we use for evaluation follow a hybrid policy using $k = 4$ for the first 5 levels and $k = 2$ for the rest. The decomposition stop when the submatrices reach size 8×8 and are encoded using DAC's.

[3] The implementation of the dictionary is orthogonal to our proposal and thus it is not considered in our evaluation in Section 3.

The evaluation of the 2RPQs follows a simple algorithm using the typical automata-theoretic approach [8]. Given a 2RPQ r, we first build the Non-deterministic Finite Automaton (NFA) associated to r, considering labels in Σ and inverse labels. Then, the graph is also considered as an NFA and the algorithm performs a breadth first search over the product automaton. In practice the product automaton cannot be constructed, but we perform the traversal implicitly. Thus the algorithm only needs to know neighbors of a node by a single label (or an inverse label) at a time, which can be efficiently computed with the k^2-tree representation as explained above.

The code is implemented in `C++` and available via Github.[4]

3 Experimental results

We compare our implementation with Sparksee [7] (version 5, February 2014) and Neo4j [9] (version 2.1, July 2014), using a machine with the following configuration: 3.40 GHz Intel Core i7-2600k (4 cores), 8 GB RAM, Archlinux OS kernel version 3.18.4. We compare the running time for several 2RPQs considering two evaluation scenarios: the *warm* and the *cold* scenarios. The warm scenario simulates the conditions of an already running server: we first perform a warm-up run, and then report the results for the second run (of the same query). The cold scenario reports the running time of the first run. The idea is to analyze how caching influences the performance. For every query tested, we run 10 000 experiments and report the average time.

In our experiments, we use the data generator provided by the graph database benchmark GDBench [2]. Graphs generated by GDBench have a simple social network structure with nodes representing persons and webpages, friend-edges between persons, and like-edges from persons to webpages. We considered graphs of different size ranging from 10 million to 40 million nodes.

Figs. 1-3 present a comparison for queries `like`, `friend/friend`, and `like/like`$^{-}$ in the warm scenario. Our implementation is labelled as k2tdb in the figures. For these queries, k2tdb and Sparksee show a similar performance (running times are in the same order of magnitude), while Neo4j is considerably slower. Notice that for queries involving only like-edges, k2tdb has a performance twice as good as Sparksee in a warm scenario (Fig. 1 and 3). This is consistent with the characteristics of k^2-trees which are specially suited for sparse graphs, and the subgraph of like-edges enjoys this feature.

Our next experiment considers navigational path queries which are one of the most important features of 2RPQs. Informally, we consider queries that goes from one person to their set of friends, and the friends of its friends, and so on, for several steps. More formally, we consider the queries `friend`, `friend/friend`, `friend/friend/friend`,... until five copies of the friend-edge. These queries are denoted by f1, f2, f3, f4, f5, respectively. We also test the query `friend*`, denoted by f*, which allows to navigate an arbitrary number of friend-edges. We report on the results for a graph with 20 million nodes (Fig. 4 and 5). In the warm scenario Sparksee slightly outperforms our implementation (Fig. 4), but both stays within the same order of magnitude. For the cold scenario our implementation has a better performance (Fig. 5), and the difference is

[4] https://github.com/nilehmann/libk2tree, https://github.com/nilehmann/k2tdb

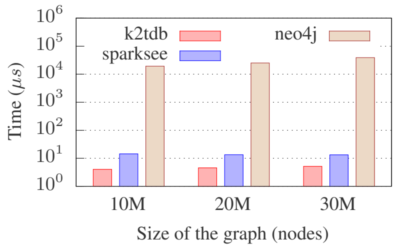


Fig. 1: Running time for query `like`

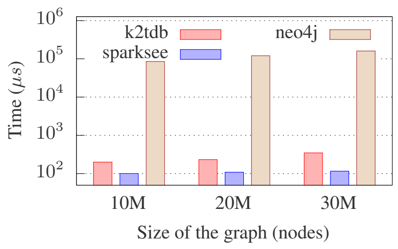


Fig. 2: Running time for `friend/friend`

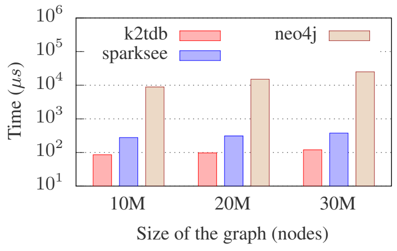


Fig. 3: Running time for `like/like`$^-$

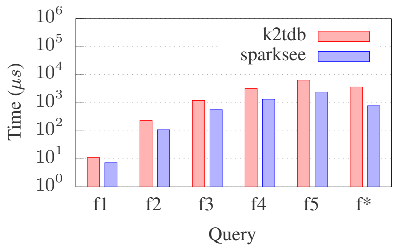


Fig. 4: Path queries in warm scenario

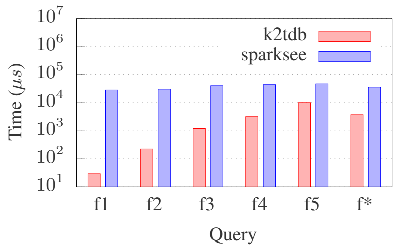


Fig. 5: Path queries in cold scenario

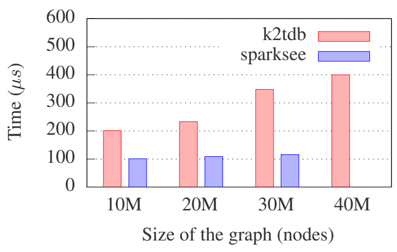


Fig. 6: Scalability test for `friend/friend`

quite substantial for the simpler queries. This behavior can be explained by the caching techniques used in Sparksee. As the portion of the graph being traversed gets larger, the probability of using the cache increases. Our solution best suits a scenario where the explored portion of the graph has not yet been visited. This presents an interesting opportunity for improving our implementation by using similar ideas for caching, for example, by decompressing and caching some portions of the graph as we traverse it.

Our last experiment is a scalability test for query `friend/friend` over graphs of increasing size (Fig. 6). The Sparksee license that we use, allows graphs with at most 1 billion objects (nodes plus edges), which disallows the loading of the 40M-node graph. Thus, we show the time up to 30M nodes for Sparksee. The growth in running time for k2tdb is more pronounced compared with Sparksee, but it still shows a linear behavior. Further experimentation with larger graphs is needed to obtain specific conclusions.

4 Conclusions and future work

Our naive implementation of 2RPQs over compressed graph structures shows a competitive performance compared with highly-optimized graph database engines. This shows the benefits of considering compressed data structures when querying graphs with expressive query languages. Our implementation shows a particularly good performance in the *cold scenario* where no caching is permitted. This presents an interesting opportunity for optimizing our implementation with caching techniques. Our ongoing work includes the implementation of 2RPQs in a less naive way, taking a more specific advantage of the way the graph is actually compressed.

References

1. Álvarez-García, S., Brisaboa, N., Fernández, J., Martínez-Prieto, M., Navarro, G.: Compressed vertical partitioning for efficient RDF management. Knowledge and Information Systems (2014), to appear
2. Angles, R., Prat-Pérez, A., Dominguez-Sal, D., Larriba-Pey, J.L.: Benchmarking database systems for social network applications. In: GRADES. p. 15 (2013)
3. Brisaboa, N., Ladra, S., Navarro, G.: DACs: Bringing direct access to variable-length codes. Information Processing and Management (IPM) 49(1), 392–404 (2013)
4. Brisaboa, N.R., Ladra, S., Navarro, G.: Compact representation of web graphs with extended functionality. Inf. Syst. 39, 152–174 (2014)
5. González, R., Grabowski, S., Mäkinen, V., Navarro, G.: Practical implementation of rank and select queries. In: Wea 2005. pp. 27–38
6. Harris, S., Seaborne, A.: Sparql 1.1 query language. W3C Recommendation (2013)
7. Martínez-Bazan, N., Gómez-Villamor, S., Escale-Claveras, F.: DEX: A high-performance graph database management system. In: ICDE Workshops 2011. pp. 124–127
8. Mendelzon, A.O., Wood, P.T.: Finding regular simple paths in graph databases. In: VLDB 1989. pp. 185–193 (1989)
9. Webber, J.: A programmatic introduction to neo4j. In: SPLASH 2012. pp. 217–218

Tractable Query Answering and Optimization for Extensions of Weakly-Sticky Datalog±

Mostafa Milani and **Leopoldo Bertossi**

Carleton University, School of Computer Science, Ottawa, Canada
{mmilani,bertossi}@scs.carleton.ca

Summary. We consider a semantic class, *weakly-chase-sticky* (WChS), and a syntactic subclass, *jointly-weakly-sticky* (JWS), of *Datalog*$^\pm$ programs. Both extend that of weakly-sticky (WS) programs, which appear in our applications to data quality. For WChS programs we propose a practical, polynomial-time query answering algorithm (QAA). We establish that the two classes are closed under magic-sets rewritings. As a consequence, QAA can be applied to the optimized programs. QAA takes as inputs the program (including the query) and semantic information about the "finiteness" of predicate positions. For the syntactic subclasses JWS and WS of WChS, this additional information is computable.

***Datalog*$^\pm$.** *Datalog*, a rule-based language for query and view-definition in relational databases [5], is not expressive enough to logically represent interesting and useful ontologies, at least of the kind needed to specify conceptual data models. *Datalog*$^\pm$ extends *Datalog* by allowing existentially quantified variables in rule heads ($\exists$-variables), equality atoms in rule heads, and program constraints [2]. Hence the "+" in *Datalog*$^\pm$, while the "−" reflects syntactic restrictions on programs, for better computational properties.

A typical *Datalog*$^\pm$ program, $\mathcal{P}$, is a finite set of rules, $\Sigma \cup E \cup N$, and an extensional database (finite set of *facts*), D. The rules in Σ are *tuple-generating-dependencies* (*tgds*) of the form $\exists \bar{x} P(\bar{x}, \bar{x}') \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, where $\bar{x}' \subseteq \bigcup \bar{x}_i$, and $\bar{x}$ can be empty. E is a set of *equality-generating-dependencies* (*egds*) of the form $x = x' \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, with $\{x, x'\} \subseteq \bigcup \bar{x}_i$. Finally, N contains *negative constraints* of the form $\bot \leftarrow P_1(\bar{x}_1), \ldots, P_n(\bar{x}_n)$, where $\bot$ is false.

Example 1. The following *Datalog*$^\pm$ program shows a tgd, an egd, and a negative constraint, in this order: $\exists x \ \mathit{Assist}(y, x) \leftarrow \mathit{Doctor}(y)$; $x = x' \leftarrow \mathit{Assist}(y, x),\ \mathit{Assist}(y, x')$; $\bot \leftarrow \mathit{Specialist}(y, x, z),\ \mathit{Nurse}(y, z)$. □

Below, when we refer to a class of *Datalog*$^\pm$ programs, we consider only Σ, the tgds. Due to different syntactic restrictions, *Datalog*$^\pm$ can be seen as a class of sublanguages of *Datalog*$^\exists$, which is the extension of *Datalog* with tgds with $\exists$-variables [10].

The rules of a *Datalog*$^\pm$ program can be seen as an ontology $\mathcal{O}$ on top of D, which can be *incomplete*. $\mathcal{O}$ plays the role of: (a) a "query layer" for D, providing ontology-based data access (OBDA) [9], and (b) the specification of a completion of D, usually carried out through the *chase* mechanism that, starting from D, iteratively enforces the rules in Σ, generating new tuples. This leads to a possibly infinite instance extending D, denoted with $\mathit{chase}(\Sigma, D)$.

The answers to a conjunctive query $\mathcal{Q}(\bar{x})$ from D wrt. Σ is a sequence of constants $\bar{a}$, such that $\Sigma \cup D \models \mathcal{Q}(\bar{a})$ (or *yes* or *no* in case $\mathcal{Q}$ is boolean). The answers can be obtained by querying as usual the *universal* instance $chase(\Sigma, D)$. The chase may be infinite, which leads, in some cases, to undecidability of query answering [8]. However, in some cases where the chase is infinite, query answering (QA) is still computable (decidable), and even tractable in the size of D. Syntactic classes of *Datalog*$^\pm$ programs with tractable QA have been identified and investigated, among them: *sticky* [4, 7], and *weakly-sticky* [4] *Datalog*$^\pm$ programs.

Our Need for QA Optimization. In our work, we concentrate on the *stickiness* and *weak-stickiness* properties, because these programs appear in our applications to quality data specification and extraction [11], with the latter task accomplished through QA, which becomes crucial.

Sticky programs [4] satisfy a syntactic restriction on the multiple occurrences of variables (joins) in the body of a *tgd*. Weakly-sticky (WS) programs form a class that extends that of sticky programs [4]. WS-*Datalog*$^\pm$ is more expressive than sticky *Datalog*$^\pm$, and results from applying the notion of *weak-acyclicity* (WA) as found in data exchange [6], to relax acyclicity conditions on stickiness. More precisely, in comparison with sticky programs, WS programs require a milder condition on join variables, which is based on a program's *dependency graph* and the positions in it with finite rank [6].[1]

For QA, sticky programs enjoy *first-order rewritability* [7], i.e. a conjunctive query $\mathcal{Q}$ posed to $\Sigma \cup D$ can be rewritten into a new first-order (FO) query $\mathcal{Q}'$, and correctly answered by posing $\mathcal{Q}'$ to D, and answering as usual. For WS programs, QA is *PTIME*-complete in data, but the polynomial-time algorithm provided for the proof in [4] is not a practical one.

Stickiness of the Chase. In addition to (syntactic) stickiness, there is a "semantic" property of programs, which is relative to the chase (and the data, D), and is called "chase-stickiness" (ChS). Stickiness implies semantic stickiness (but not necessarily the other way around) [4]. For chase-sticky programs, QA is tractable [4].

Intuitively, a program has the chase-stickiness property if, due to the application of a tgd σ: When a value replaces a repeated variable in the body of a rule, then that value also appears in all the head atoms obtained through the iterative enforcement of applicable rules that starts with σ. So, that value is propagated all the way down through all the possible subsequent steps.

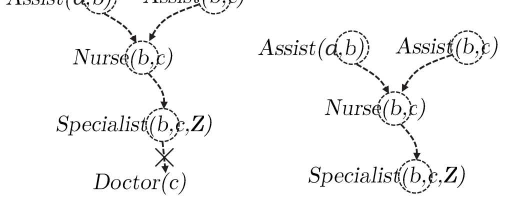


Fig. 1. The chase for a non-ChS program and the chase for a ChS program, resp.

[1] A position refers to a predicate attribute, e.g. *Nurse*[2].

Example 2. Consider $D = \{Assist(a,b), Assist(b,c)\}$, and the following set, Σ_1, of tgds: $Nurse(y,z) \leftarrow Assist(x,y), Assist(y,z)$; $\exists z\ Specialist(x,y,z) \leftarrow Nurse(x,y)$; $Doctor(y) \leftarrow Specialist(x,y,z)$. Σ_1 is not ChS, as the chase on the LHS of Figure1 shows: value b is not propagated all the way down to $Doctor(c)$. However, program Σ_2, which is Σ_1 without its third rule, is ChS, as shown on the RHS of Figure1. □

Weak-Stickiness of the Chase. Weak-stickiness also has a semantic version, called "weak-chase-stickiness" (WChS); which is implied by the former. So as for chase-stickiness, weak-chase-sticky programs have a tractable QA problem, even with a possibly infinite chase. This class is one of the two we introduce and investigate. They appear in double-edged boxes in Figure 2, with dashed edges indicating a semantic class.

By definition, weak-chase-stickiness is obtained by relaxing the condition for ChS: it applies only to values for repeated variables in the body of σ that appear in so-called *infinite positions*, which are semantically defined. A position is infinite if there is an instance D for which an unlimited number of different values appear in $Chase(\Sigma, D)$.

Given a program, deciding if a position is infinite is unsolvable, so as deciding in general if the chase terminates. Consequently, it is also undecidable if a program is WChS. However, there are syntactic conditions on programs [6, 12] that determine some (but not necessarily all) the finite positions. For example, the notion of position *rank*, based on the program's *dependency graph*, are used in [6, 4] to identify a (sound) set of finite positions, those with *finite rank*. Furthermore, finite-rank positions are used in [4] to define weakly-sticky (WS) programs as a syntactic subclass of WChS.

Finite Positions and Program Classes. In principle, any set-valued function S that, given a program, returns a subset of the program's finite positions can be used to define a subclass WChS(S) of WChS. This is done by applying the definition of WChS above with "infinite positions" replaced by "non-S-finite positions". Every class WChS(S) has a tractable QA problem.

S could be computable on the basis of the program syntax or not. In the former case, it would be a "syntactic class". Class $WChS(S)$ grows monotonically with S in the sense that if $S_1 \subseteq S_2$ (i.e. S_1 always returns a subset of the positions returned by S_2), then $WChS(S_1) \subseteq WChS(S_2)$. In general, the more finite positions are (correctly) identified (and the consequently, the less finite positions are treated as infinite), the more general the subclass of WChS that is identified or characterized.

For example, the function $S^{\perp}$ that always returns an empty set of finite positions, $WChS(S^{\perp})$ is the class of sticky programs, because stickiness must hold no matter what the (in)finite positions are. At the other extreme, for function $S^{\top}$ that returns all the (semantically) finite positions, $WChS(S^{\top})$ becomes the class WChS. (As mentioned above, $S^{\top}$ is in general uncomputable.) Now, if S^{rank} returns the set of finite-rank positions (for a program $\mathcal{P}$, usually denoted by $\Pi_F(\mathcal{P})$ [6]), $WChS(S^{rank})$ is the class of WS programs.

Joint-Weakly-Stickiness. The *joint-weakly-sticky* (JWS) programs we introduce form a syntactic class strictly between WS and WChS. Its definition appeals to the notions of *joint-acyclicity* and *existential dependency graphs* introduced in [12]. Figure 2 shows this syntactic class, and the inclusion relationships between classes of *Datalog*$^{\pm}$ programs.[2]

If S^{ext} denotes the function that specifies finite positions on the basis of the *existential dependency graphs* (EDG), implicitly defined in [12], the JWS class is, by definition, the class $\mathit{WChS}(S^{ext})$. EDGs provide a finer mechanism for capturing (in)finite positions in comparison with positions ranks (defined through dependency graphs): $S^{rank} \subseteq S^{ext}$. Consequently, the class of JWS programs, i.e. $\mathit{WChS}(S^{ext})$, is a strict superclass of WS programs, i.e. $\mathit{WChS}(S^{rank})$.[3]

QAA for WChS. Our query answering algorithm for WChS programs is parameterized by a (sound) finite-position function S as above. It is denoted with AL^S, and takes as input Σ, D, query $\mathcal{Q}$, and $S(\Sigma)$, which is a subset of the program's finite positions (the other are treated as infinite by default).

The customized algorithm AL^S is guaranteed to be sound and complete only when applied to programs in $\mathit{WChS}(S)$: $AL^S(\Sigma, D, \mathcal{Q})$ returns all and only the query answers. (Actually, AL^S is still sound for any program in WChS.) AL^S runs in polynomial-time in data; and can be applied to both the WS and the JWS syntactic classes. For them the finite-position functions are computable.

AL^S is based on the concepts of *parsimonious chase* (*pChase*) and *freezing nulls*, as used for QA with *shy Datalog*, a fragment of *Datalog*$^{\exists}$ [10]. At a *pChase* step, a new atom is added only if a homomorphic atom is not already in the chase. Freezing a null is promoting it to a constant (and keeping it as such in subsequent chase steps). So, it cannot take (other) values under homomorphisms, which may create new *pChase* steps. Resumption of the *pChase* means freezing *all* nulls, and continuing *pChase* until no more *pChase* steps are applicable.

Query answering with shy programs has a first phase where the *pChase* runs until termination (which it does). In a second phase, the *pChase* iteratively resumes for a number of times that depends on the number of distinct $\exists$-variables in the query. This second phase is required to properly deal with joins in the query. Our QAA for WChS programs (AL) is similar, it has the same two phases, but a *pChase* step is modified: after every application of a *pChase* step that generates nulls, the latter that appear in S-finite positions are immediately frozen.

Magic-Sets Rewriting. It turns out that JWS, as opposed to WS, is closed under the quite general magic-set rewriting method [5] introduced in [1]. As a

[2] Rectangles with dotted-edges show semantic classes, and double-edged rectangles show the classes introduced in this work. Notice that programs in semantic classes include the instance D, but syntactic classes are data-independent (for any instance as long as the syntactic conditions apply).

[3] The JWS class is different from (and incomparable with) the class of *weakly-sticky-join* programs (WSJ) introduced in [3], which extends the one of WS programs with consideration that are different from those used for JWS programs. WSJ generalizes WS on the basis of the weakly-sticky-join property of the chase [3, 4] and is related to repeated variables in single atoms.

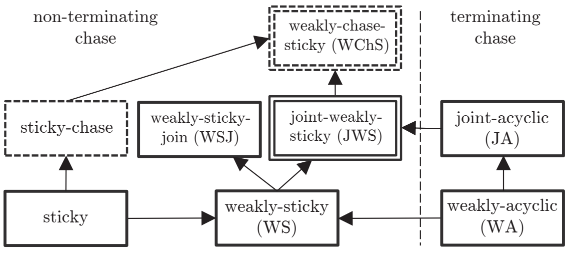


Fig. 2. Generalization relationships among program classes.

consequence, AL can be applied to both the original JWS program and its magic rewriting. (Actually, this also holds for the superclass WChS.)

It can be proved that (our modification of) the magic-sets rewriting method in [1] does not change the character of the original finite or infinite positions. The specification of (in)finiteness character of positions in magic predicates is not required by AL, because no new nulls appear in them during the AL execution. As consequence, the MS method rewriting can be perfectly integrated with our QAA, introducing additional efficiency.

Acknowledgments: We are very grateful to Mario Alviano and the DLV team for providing us with information and support in relation to existential Datalog. We also appreciate useful conversations with Andrea Cali and Andreas Pieris on Datalog±, and important comments from Andrea Cali on an earlier version of this paper.

References

1. M. Alviano, N. Leone, M. Manna, G. Terracina and P. Veltri. Magic-Sets for Datalog with Existential Quantifiers. *Proc. Datalog 2.0*, 2012, pp. 31-43.
2. A. Cali, G. Gottlob, and T. Lukasiewicz. Datalog±: A Unified Approach to Ontologies and Integrity Constraints. *Proc. ICDT*, 2009, pp. 14-30.
3. A. Cali, G. Gottlob and A. Pieris. Query Answering under Non-Guarded Rules in Datalog+/-. *Proc. RR*, 2010, pp. 1-17.
4. A. Cali, G. Gottlob, and A. Pieris. Towards more Expressive Ontology Languages: The Query Answering Problem. *Artificial Intelligence*, 2012, 193:87-128.
5. S. Ceri, G. Gottlob and L. Tanca. *Logic Programming and Databases.* Springer, 1990.
6. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 2005, 336:89-124.
7. G. Gottlob, G. Orsi, and A. Pieris. Query Rewriting and Optimization for Ontological Databases. *ACM TODS*, 2014, 39(3):25.
8. D. S. Johnson, and A. Klug. Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. *Proc. PODS*, 1984, pp. 164-169.
9. M. Lenzerini. Ontology-Based Data Management. Proc. AMW 2012, CEUR Proceedings, Vol. 866, pp. 12-15.
10. N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently Computable Datalog$^{\exists}$ Programs. *Proc. KR*, 2012, pp. 13-23.
11. M. Milani, L. Bertossi and S. Ariyan. Extending Contexts with Ontologies for Multi-dimensional Data Quality Assessment. *Proc. DESWeb*, 2014, pp. 242-247.
12. S. Rudolph, and M. Krötzsch. Extending Decidable Existential Rules by Joining Acyclicity and Guardedness. *Proc. IJCAI*, 2011, pp. 963-968.

Saturation, Definability, and Separation for XPath on Data Trees

Sergio Abriola[1], María Emilia Descotte[1], and Santiago Figueira[1,2]

[1] University of Buenos Aires, Argentina
[2] CONICET, Argentina

Abstract. We study the expressive power of some fragments of XPath equipped with (in)equality tests over data trees.
Our main results are the definability theorems, which give necessary and sufficient conditions under which a class of data trees can be defined by a node expression or set of node expressions, and our separation theorems, which give sufficient conditions under which two disjoint classes of data trees can be separated by a class of data trees definable in XPath.

Keywords: XPath · data tree · bisimulation · definability · first-order logic · ultraproduct · saturation · separation.

1 Introduction

The abstraction of an XML document is a data tree, i.e. a tree whose every node contains a tag or label (such as *LastName*) from a finite domain, and a data value (such as *Smith*) from an infinite domain. XPath is the most widely used query language for XML documents; it is an open standard and constitutes a World Wide Web Consortium (W3C) Recommendation [3]. XPath$_=$ has syntactic operators to navigate the tree using the 'child', 'parent', 'sibling', etc. accessibility relations, and can make tests on intermediate nodes. It can express properties of the underlying tree structure of the XML document, such as "*the root of the tree has a child labeled a and a child labeled b*", and it can express conditions on the actual data contained in the attributes, such as "*the root of the tree has two children with same tag a but different data value*".

First, we provide notions of saturation and ultraproducts that are adequate for XPath$_=$, and show that bisimulation coincides with logical equivalence over saturated data trees. Using these tools, we show definability theorems, giving necessary and sufficient conditions under which a class of data trees can be defined by a node expression or set of node expressions of XPath$_=$. Finally we give separation results, providing sufficient conditions under which two disjoint classes of data trees can be separated by a class of data trees definable in XPath$_=$.

While on this work we will only show results for the fragment of XPath that can only navigate via the 'child' accessibility relation, similar results hold for the vertical fragment having both the 'child' and 'parent' navigational operators.

The results on definability of this paper appeared originally in [1].

2 Preliminaries

Data trees. We say that $\mathcal{T}$ is a **data tree** if it is a tree from $\mathit{Trees}(\mathbb{A}\times\mathbb{D})$, where $\mathbb{A}$ is a finite set of **labels** and $\mathbb{D}$ is an infinite set of **data values**. The data of a node x is denoted $\mathit{data}(x)$, and its label is $\mathit{label}(x)$. The set of nodes of a data tree $\mathcal{T}$ is denoted T.

Downward XPath with data tests. We consider a fragment of XPath that corresponds to the navigational part of XPath 1.0 with data equality and inequality. $\text{XPath}_=$ is a two-sorted language, with **path expressions** (that we write α,β,γ) and **node expressions** (that we write φ,ψ,η). The **downward XPath**, notated $\text{XPath}^{\downarrow}_{=}$ is defined by mutual recursion as follows:

$$\begin{array}{rcll} \alpha,\beta & ::= & o \mid [\varphi] \mid \alpha\beta \mid \alpha\cup\beta & o\in\{\varepsilon,\downarrow\} \\ \varphi,\psi & ::= & a \mid \neg\varphi \mid \varphi\wedge\psi \mid \varphi\vee\psi \mid \langle\alpha\rangle \mid \langle\alpha=\beta\rangle \mid \langle\alpha\neq\beta\rangle & a\in\mathbb{A} \end{array}$$

Node expressions represent properties on nodes. They are evaluated in nodes, and, intuitively, $\langle\alpha=\beta\rangle$ is true at x if there are two paths starting in x, one satisfying the property α, and the other satisfying the property β, which end in nodes with equal data value. On the other hand, path expressions represent properties on paths. They are evaluated in pairs of nodes. For instance $\downarrow$ is true at (x,y) if y is a child of x, and $[\varphi]$ is true at x,y if $x=y$ and x satisfies φ.

Let $\mathcal{T}$ and $\mathcal{T}'$ be data trees, and let $u\in T$, $u'\in T'$. We say that $\mathcal{T},u$ and $\mathcal{T}',u'$ are **logically equivalent for $\text{XPath}^{\downarrow}_{=}$** if no $\text{XPath}^{\downarrow}_{=}$ can distinguish node u from u'.

Bisimulations. Notions of bisimulation present a way to determine whether two pointed data trees can be distinguished by a series of moves in XPath. We do not reproduce them here, but it is worth mentioning that they are forms of back-and-forth conditions over two data trees.

The main previous result in the literature establishing the connection between bisimulation and equivalence is the following:

Theorem 1. *[4] If $\mathcal{T},u$ is bisimilar to $\mathcal{T}',u'$, then they are logically equivalent. If $\mathcal{T}$ and $\mathcal{T}'$ are finitely branching, the other implication also holds.*

3 Saturation and quasi-ultraproducts

We introduce a notion of saturation for the downward fragment of XPath, and show that the reverse implication of Theorem 1 is true over saturated data trees. Saturation is the key ingredient to show the Definability theorems, but their use lays hidden in the proof.

Saturation. Let $\langle\Sigma_1,\dots,\Sigma_n\rangle$ and $\langle\Gamma_1,\dots,\Gamma_m\rangle$ be tuples of sets of $\text{XPath}^{\downarrow}_{=}$-formulas. Given a data tree $\mathcal{T}$ and $u\in T$, we say that $\langle\Sigma_1,\dots,\Sigma_n\rangle$ and $\langle\Gamma_1,\dots,\Gamma_m\rangle$ are **$=^{\downarrow}_{n,m}$-satisfiable** [resp. **$\neq^{\downarrow}_{n,m}$-satisfiable**] at $\mathcal{T},u$ if there exist $v_0\to v_1\to\dots\to v_n\in T$ and $w_0\to w_1\to\dots\to w_m\in T$ such that $u=v_0=w_0$ and

1. for all $i \in \{1, \dots, n\}$, $\mathcal{T}, v_i \models \Sigma_i$;
2. for all $j \in \{1, \dots, m\}$, $\mathcal{T}, w_j \models \Gamma_j$; and
3. $data(v_n) = data(w_m)$ [resp. $data(v_n) \neq data(w_m)$].

We say that $\langle \Sigma_1, \dots, \Sigma_n \rangle$ and $\langle \Gamma_1, \dots, \Gamma_m \rangle$ are $=^{\downarrow}_{n,m}$**-finitely satisfiable** [resp. $\neq^{\downarrow}_{n,m}$**-finitely satisfiable**] at $\mathcal{T}, u$ if for every finite $\Sigma'_i \subseteq \Sigma_i$ and finite $\Gamma'_j \subseteq \Gamma_j$, we have that $\langle \Sigma'_1, \dots, \Sigma'_n \rangle$ and $\langle \Gamma'_1, \dots, \Gamma'_m \rangle$ are $=^{\downarrow}_{n,m}$-satisfiable [resp. $\neq^{\downarrow}_{n,m}$-satisfiable] at $\mathcal{T}, u$.

Definition 2. *We say that a data tree $\mathcal{T}$ is* $\downarrow$***-saturated*** *if for every $n, m \in \mathbb{N}$, every pair of tuples $\langle \Sigma_1, \dots, \Sigma_n \rangle$ and $\langle \Gamma_1, \dots, \Gamma_m \rangle$ of sets of $XPath^{\downarrow}_{=}$-formulas, every $u \in T$, and $\star \in \{=, \neq\}$, the following is true:*

if $\langle \Sigma_1, \dots, \Sigma_n \rangle$ and $\langle \Gamma_1, \dots, \Gamma_m \rangle$ are $\star^{\downarrow}_{n,m}$-finitely satisfiable at $\mathcal{T}, u$ then $\langle \Sigma_1, \dots, \Sigma_n \rangle$ and $\langle \Gamma_1, \dots, \Gamma_m \rangle$ are $\star^{\downarrow}_{n,m}$-satisfiable at $\mathcal{T}, u$.

Proposition 3. *For $\downarrow$-saturated data trees, bisimulation coincides with logical equivalence.*

Quasi-ultraproducts We introduce the notion of quasi-ultraproduct, a variant of the usual notion of first-order model theory, which will be needed for the definability theorems. Some of our results for quasi-ultraproducts make use of the fundamental theorem of ultraproducts (see e.g. [2, Thm. 4.1.9]).

Definition 4. *Suppose $(\mathcal{T}_i, u_i)_{i \in I}$ is a family of pointed data trees, U is an ultrafilter over I, $\mathcal{T}^*$ is the ultraproduct of $(\mathcal{T}_i, u_i)_{i \in I}$, and u^* is the ultralimit of $(u_i)_{i \in I}$. The* $\downarrow$***-quasi ultraproduct*** *of $(\mathcal{T}_i, u_i)_{i \in I}$ modulo U is the pointed data tree $(\mathcal{T}^*|u^*, u^*)$, where $\mathcal{T}^*|u^*$ denotes the subtree of $\mathcal{T}^*$ induced by al the descendants of u^*. As a particular case one has the notion of* $\downarrow$***-quasi ultrapower***.

4 Definability

Definability theorems address the question of which properties of models can be defined via formulas of the logic. If K is a class of pointed data trees, we denote its complement by $\overline{K}$.

Theorem 5. *Let K be a class of pointed data trees. Then K is definable by a set of $XPath^{\downarrow}_{=}$-formulas iff K is closed under $\downarrow$-bisimulations and $\downarrow$-quasi ultraproducts, and $\overline{K}$ is closed under $\downarrow$-quasi ultrapowers.*

Theorem 6. *Let K be a class of pointed data trees. Then K is definable by an $XPath^{\downarrow}_{=}$-formula iff both K and $\overline{K}$ are closed under $\downarrow$-bisimulations and $\downarrow$-quasi ultraproducts.*

The notion of ℓ-bisimulation is a restricted version of $\downarrow$-bisimulations. It has been shown to coincide with the notion of ℓ-equivalence, which informally means *indistinguishable by $XPath^{\downarrow}_{=}$ formulas that cannot "see" beyond ℓ 'child'-steps from the current point of evaluation.*

Theorem 7. *Let K be a class of pointed data trees. Then K is definable by a formula of $XPath^{\downarrow}_{=}$ iff K is closed by ℓ-bisimulations for $XPath^{\downarrow}_{=}$ for some ℓ.*

5 Separation

Separation theorem provide conditions under which two disjoint classes of pointed models can be separated by a class definable in the logic.

Theorem 8. *Let K_1 and K_2 be two disjoint classes of pointed data trees such that K_1 is closed under $\downarrow$-bisimulations and $\downarrow$-quasi ultraproducts and K_2 is closed under $\downarrow$-bisimulations and $\downarrow$-quasi ultrapowers. Then there exists a third class K which is definable by a set of $XPath^{\downarrow}_{=}$-formulas, contains K_1 and is disjoint from K_2.*

Theorem 9. *Let K_1 and K_2 be two disjoint classes of pointed data trees closed under $\downarrow$-bisimulations and $\downarrow$-quasi ultraproducts. Then there exists a third class K which is definable by an $XPath^{\downarrow}_{=}$-formula, contains K_1 and is disjoint from K_2.*

References

1. Sergio Abriola, María Emilia Descotte, and Santiago Figueira. Definability for downward and vertical Xpath on data trees. In *Logic, Language, Information, and Computation - 21st International Workshop, WoLLIC 2014, Valparaíso, Chile, September 1-4, 2014. Proceedings*, pages 20–35, 2014.
2. C.C. Chang and H.J. Keisler. *Model theory.* Studies in logic and the foundations of mathematics. North-Holland, 1990.
3. J. Clark and S. DeRose. XML path language (XPath). Website, 1999. W3C Recommendation. `http://www.w3.org/TR/xpath`.
4. D. Figueira, S. Figueira, and C. Areces. Basic model theory of XPath on data trees. In *ICDT*, pages 50–60, 2014.

Random-Walk Closeness Centrality Satisfies Boldi-Vigna Axioms

Ricardo Mora[1] and Claudio Gutierrez[2]

Center for Semantic Web Research, Dept. Computer Science, Universidad de Chile
{rmora,cgutierr}@dcc.uchile.cl

Abstract. Recently Boldi and Vigna proposed axioms that would characterize good notions of centrality. We study a random-walk version of closeness centrality and prove that is satisfies Boldi-Vigna axioms for non-directed graphs.

Keywords: Random Walks, RDF, Centrality

1 Introduction

Consider the Euclidean plane and a set of n points: Which one is the most central? An intuitive selection would be the point p that minimizes the sum of the distances from the other points to p. Consider now a set of n cities: In which one (abstracting social constraints) would you install a delivery store? Clearly in one that minimizes the sum of distances of each city to it (here distance is not Euclidean, but highways). A similar problem can be found inside a city (where distance is something close to Manhattan's). In this paper we address this problem in the general case of undirected graphs, motivated by its application to semantic networks (particularly RDF graphs).

This is not only a nice theoretical problem. One of the big challenges that the web offers today has to do with the huge quantity of data that it contains. In particular, in the case of large knowledge networks in the form of RDF graphs, it is highly relevant to understand which ones are the "essential" concepts they represent.

What is the "good" distance in this case? There is some evidence [1, 2] that using a distance based on random walks might be a fruitful idea. It turns out, as we will show, that the idea of selecting a node v that minimizes the sum of the random walk distances from each other node u to v works really well in RDF graphs [3]. In this paper we study this notion and test it with the Boldi Vigna axioms.

The problem of detecting central nodes in a graph has been extensively investigated [4] and centrality indicators like degree and others based on shortest distances between elements such as betweenness centrality and closeness have been successfully employed on a variety of networks. By trying to unify these manifold centrality measures, recently Boldi and Vigna [5] proposed a set of axioms that would capture the essential properties that underlie all of them. They show that classic notions such as closeness, degree, betweenness centrality do not satisfy these demanding axioms. In this paper we apply a Bodi Vigna test to random walk closeness centrality. We prove that this centrality notion satisfies these axioms for non-directed graphs.

2 Preliminaries

2.1 Basic Graph Theoretical Notions

An undirected and simple graph (from now on we will work only with this kind of graph) is a pair $G = (V, E)$ where $E \subseteq [V]^2$, and $[V]^2$ is the set of all 2-elements subsets from V. The elements of V are the *vertices* of G, and the ones from E are its *edges*. When necessary, we will use the notation $V(G)$ and $E(G)$ for those sets. From now on, an element $\{u, v\} \in E$ will be denoted simply by uv. An important family of graphs are cliques: for $n \geq 1$ a *n–clique* is a graph $K_n := (V, E)$ with $|V| = n$, such that $E = [V]^2$.

A vertex u is said to be a *neighbor* of another vertex v, when $uv \in E$. Note that the definition of E implies that v is also a neighbor of u. The set of neighbors of v will be denoted by $N_G(v)$. The *degree* of v, $d_G(v)$ is the size of $N_G(v)$. Should the reference be clear, they will simply be denoted by $N(v)$ and $d(v)$.

Let $G = (V, E)$ and $G' = (V', E')$ be two graphs such that $V \subseteq V'$ and $E \subseteq E'$, then G' is said to be a *subgraph* of G (it is also said that G *contains* G'). For a subset $S \subseteq V$, $G[S] := (S, \{uv \in E : u, v \in S\})$ and $G - S := G[V \setminus S]$. Analogously, for $F \subseteq E$, $G - F := (V, E \setminus F)$.

A graph $P_n = \big(\{v_0, v_1, ..., v_n\}, \{v_0v_1, v_1v_2, ..., v_{n-1}v_n\}\big)$ with $n \geq 0$, where all v_i are distinct is called a *path*, and the number of edges in it is its *length*. A *cycle* is a special type of path such that $v_0 = v_n$. We will call a cycle of length n a *n–cycle*.

Let $G = (V, E)$ be a graph and $u, v \in V$ two distinct vertices. A path P_n in G, with $n \geq 1$ such that $v_0 = u$ and $v_n = v$, is called a *u–v path*. Also G is said to be *connected* if for all distinct $u, v \in V$ a u–v path exists in G. A *connected component* of G is a maximally connected subgraph H. Note that a connected graph has only one connected component. An edge uv of G is said to be a *bridge* if the graph $G - uv$ contains at least one more connected component than G.

2.2 Random Walks

The next definitions come from the work of Lovász in random walk theory [6].

Let $G = (V, E)$ be a connected graph such that $|V| = n$ and $|E| = m$, where $n, m \in \mathbb{N}$. Formally, a *random walk* is a sequence of vertices obtained as follows: it starts at a vertex v_0, and if at the t-th step it is at a vertex $v_t = u$, it moves to a neighbor v of u with probability $p_{uv} = 1/d(u)$. Note that the sequence of random vertices $(v_t : t = 0, 1, ...)$ is a Markov chain.

P_t will denote the distribution of v_t: $P_t(v) = \mathbb{P}(v_t = v)$. The vertex v_0 may be fixed, but may also be drawn from an initial distribution P_0. This initial distribution is said to be *stationary* if $P_1 = P_0$ (which will imply that $P_t = P_0\ \forall t \geq 0$, because of the construction of the random walk). It can be easily proved that the distribution $\pi(v) := d(v)/2m$ is stationary for every graph G. From now on π will be referred simply as the *stationary distribution* (it is not difficult to prove that this distribution is unique, which makes this reference valid).

Definition 1. The *hitting time* $H(u, v)$ is the expected number of steps that a random walk starting at vertex u takes to reach vertex v for the first time.

Definition 2. Let S be a subset from V. The *hitting time for a set* $H(u, S)$ is the expected number of steps that a random walk starting at vertex u takes to reach some vertex in S for the first time.

When talking about $H(S, u)$, a distribution (based on S) for the starting vertex of the random walk has to be specified. Therefore, if $\mathbb{P}$ is that distribution, $H_{\mathbb{P}}(S, u)$ will be the expected number of steps that a random walk starting at a vertex of S (selected according to $\mathbb{P}$) takes to reach vertex u for the first time. Note that for every pair of vertices u, v

$$H(u, v) = H(u, \{v\}) = H_{\mathbb{P}}(\{u\}, v) \ .$$

because the only starting distribution $\mathbb{P}$ in $\{u\}$ is the trivial one.

3 Random Walk Closeness Centrality

We are now in a position to formalize our notion of random walk centrality. The definition is motivated by several insights coming from different sources, but mainly from actual semantics graphs (RDF graphs). From a formal point of view –and this is the motivation of this paper– it satisfies (as it will be proven later) the three axioms of centrality proposed by Boldi and Vigna [5]. It is important to note that most centrality measures do not satisfy them all, thus making this notion of centrality interesting.

Definition 3. [cf. [3]] Given a connected graph $G = (V, E)$ and a vertex $v \in V$, the *Random Walk Closeness Centrality* of v is the real number

$$h^{\swarrow}(v) = \sum_{\substack{w \in V \\ w \neq v}} H(w, v).$$

The smaller $h^{\swarrow}(v)$ is, the more central v is. A similar notion of centrality based on random walks was proposed by Noh and Rieger [7]. In fact, the definition proposed here is a particular case of a more general notion that includes both, Noh and Rieger's and ours, but that will not be studied in this paper.

4 General Properties

To prove that random walk closeness centrality satisfies Boldi and Vigna axioms, first we will need some properties.

Proposition 4. *Let u,v be distinct vertices of a connected graph G, and $S \subseteq V \setminus \{u, v\}$ be such that every u–v path contains some vertex from S. Then*

$$H(u, v) = H(u, S) + H_{\mathbb{P}_{u,S}}(S, v),$$

where $\mathbb{P}_{u,S}$ is the distribution for random walks that start in S, such that for all w in that set, $\mathbb{P}_{u,S}(w)$ is the probability that w is the first vertex from S that a random walk starting in u reaches.

Proof. Let Ω_u be the sample space containing all possible outcomes associated to random walks that start in u and occur in G. Similarly, let Ω_S be the one associated to random walks that start in any vertex w of S for which $\mathbb{P}_{u,S}(w) > 0$. Consider the random variables $T_{uS} : \Omega_u \to \mathbb{R}$, $T_{Sv} : \Omega_S \to \mathbb{R}$ and $T_{uv} : \Omega_u \to \mathbb{R}$ defined as follows

$T_{uS}(\omega) :=$ *# of steps that ω takes in order to reach some vertex in S for the first time.*

$T_{Sv}(\omega) :=$ *# of steps that ω takes in order to reach vertex v for the first time.*

$T_{uv}(\omega) :=$ *# of steps that ω takes in order to reach vertex v for the first time.*

Define $X(\omega) := T_{uv}(\omega) - T_{uS}(\omega)$. Namely, X is also a random variable that satisfies $X : \Omega_u \to \mathbb{R}$ and

$X(w) =$ *# of steps that ω takes (after reaching S for the first time) in order to reach vertex v for the first time.*

For $\omega \in \Omega_u$ write $\omega = (u, v_1, v_2 ...)$ and define $i_S(\omega) := \min_{i>0}\{v_i \in S\}$ and $i_v(\omega) := \min_{i>0}\{v_i = v\}$. Also, define $\omega_S := (v_{i_S}, v_{i_S+1}, ...)$ and $j(\omega_S) := \min_{i>0}\{v_{i+i_S} = v\}$. Note that $j(\omega_S) = i_v(\omega) - i_S(\omega)$ and that ω_S is an element of Ω_S, because random walk are Markov process. Then, for $n \in \mathbb{N}$

$$\begin{aligned}\mathbb{P}(X(\omega) = n) &= \sum_{w \in S} \mathbb{P}(v_{i_S(\omega)} = w)\mathbb{P}(i_v(\omega) - i_S(\omega) = n) \\ &= \sum_{w \in S} \mathbb{P}_{u,S}(w)\mathbb{P}(j(\omega_S) = n) = \mathbb{P}(T_{Sv}(\omega_S) = n) \ .\end{aligned}$$

Therefore, X and T_{Sv} are random variables with the same expected value. Finally, by using this

$$\begin{aligned}H(u, v) &= \mathbb{E}(T_{uv}) = \mathbb{E}(T_{uS} + X) = \mathbb{E}(T_{uS}) + \mathbb{E}(X) \\ &= \mathbb{E}(T_{uS}) + \mathbb{E}(T_{Sv}) = H(u, S) + H_{\mathbb{P}_{u,S}}(S, v) \ .\end{aligned}$$

□

Corollary 5. *Let u,w,v be three distinct vertices of a connected graph G such that every u–v path contains w. Then*

$$H(u,v) = H(u,w) + H(w,v) \ .$$

Proof. It follows directly from proposition 4 by considering $S = \{w\}$. □

Before stating the next theorem, we need the following result from Lovász [6] and one more definition.

Lemma 6 (Lovász [6]). *The probability that a random walk starting at u visits v before returning to u is*

$$\frac{1}{(H(u,v) + H(v,u))\pi(u)}$$

Definition 7. For a bridge uv of a connected graph $G = (V, E)$, define G_u as

$$G_u := G[\{w \in V : \forall \ w\text{–}v \textit{ path in } G, u \in w\text{–}v\}],$$

that is, G_u and G_v are the connected components of $G - uv$ (see Fig. 1 below).

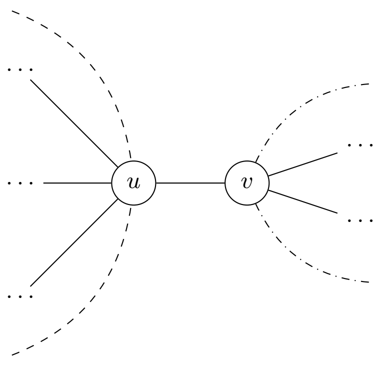


Fig. 1. *An example of a graph G with a bridge uv. To the left of the dashed line is G_u and to the right of the dash-dotted line is G_v. Note that $u \in G_u$.*

Theorem 8. *Let uv be a bridge of a connected graph G. Then*

$$H(u,v) = 2|E(G_u)| + 1 \ .$$

Proof. First note that any random walk starting at u has to go through v before stepping into another vertex of G_v, therefore $H(u,v)$ does not depend on G_v. Because of this, for simplicity, we can assume that $G_v = (v, \emptyset)$. Then, $H(v,u) = 1$

and $|E(G)| = |E(G_u)| + 1$. Now, the probability that a random walk starting at u reaches v before returning to u is $1/d(u)$. Then, it follows from lemma 6 that

$$\begin{aligned} d(u) &= (H(u,v) + H(v,u))\pi(u) = (H(u,v) + 1)\,\frac{d(u)}{2|E(G)|} \\ &= (H(u,v) + 1)\frac{d(u)}{2|E(G_u)| + 2}, \end{aligned}$$

which is equivalent to $H(u,v) = 2|E(G_u)| + 1$, that is what we wanted to prove. □

Finally, using these properties we can prove the following result that will allow us to compare the centrality of different vertices, under certain conditions.

Proposition 9. *Let uv be a bridge of a connected graph G. Then*

$$h^{\swarrow}(u) < h^{\swarrow}(v) \iff (2|E(G_u)| + 1)|V(G_u)| > (2|E(G_v)| + 1)|V(G_v)| \; .$$

Proof. The proof is straightforward and is included in the appendix.

5 Boldi and Vigna Axioms

We can now prove what was previously promised.

Boldi and Vigna [5] seek to define certain axioms that provide a formal and provable piece of information about a centrality measure so it can be assured that it correctly captures the intuitive notion of centrality. To this end, they propose to study the behavior of the measure when making changes of size, local edge density and addition of edges in the graph. It is important to note that the axioms were designed primarily for measures that work with directed, and not necessarily connected graphs. For graphs representing semantic networks, direction is not relevant because the predicate represented by the directed edge represents at the same time the inverse predicate. Therefore, we work with a version of the original axioms adapted for connected and undirected graphs.

First is the *Size Axiom.* The idea is to compare the centrality of vertices from a clique and a cycle joined through a path of large enough size. When fixing the size of one of them, and letting the other grow as much as wanted, one would expect the vertices of the latter to become more central. This is formalized as follows:

Axiom 1: (Size axiom) Consider the graph $S_{k,p}$ made by a k-clique and a p-cycle connected by a path of length l (see Fig. 2). A centrality measure satisfies the size axiom if for every k there are two constants P_k, L_k such that for all $p \geq P_k$, $l \geq L_k$, the centrality of any vertex of the p-cycle is strictly better than the centrality of any vertex in the k-clique, and the same holds when inverting the situation (that is, fixing p and letting k be as big as desired).

Proof. We will prove it for the first case as for the inverted situation the proof is analogous. For simplicity of notation we will use the labels proposed on Fig. 2 when referring to the nodes of $S_{k,p}$. Also, we will denote by $\mathcal{K}$ the subgraph of $S_{k,p}$ that corresponds to the clique.

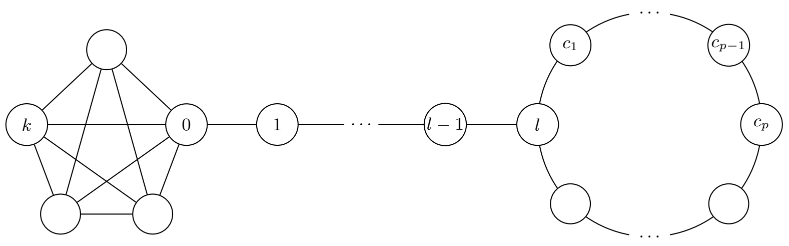


Fig. 2. *An example of graph $S_{k,p}$.*

First note that it is not difficult to prove that vertex 0 is the most central vertex of the clique, and that c_p is the one from the cycle with worst centrality. Therefore, if we prove that c_p is more central than 0, we will have proved the result. Indeed, we have that

$$\begin{aligned} h^{\swarrow}(0) &= \sum_{\substack{w \in V(\mathcal{K}) \\ w \neq 0}} H(w,0) + \sum_{j=1}^{l} H(j,0) + 2\sum_{j=1}^{p-1} H(c_j,0) + H(c_p,0) \\ &= (k-1)^2 + \sum_{j=1}^{l-1} H(j,0) + 2\sum_{j=1}^{p-1} H(c_j,l) + H(c_p,l) + 2pH(l,0) \ . \end{aligned}$$

On the other hand, the value of $h^{\swarrow}(c_p)$ is

$$\begin{aligned} &= \sum_{\substack{w \in V(\mathcal{K}) \\ w \neq 0}} H(w,c_p) + H(0,c_p) + \sum_{j=1}^{l} H(j,c_p) + 2\sum_{j=1}^{p-1} H(c_j,c_p) \\ &= \sum_{\substack{w \in V(\mathcal{K}) \\ w \neq 0}} H(w,0) + kH(0,c_p) + H(l,c_p) + \sum_{j=1}^{l-1} H(j,c_p) + 2\sum_{j=1}^{p-1} H(c_j,c_p) \\ &= (k-1)^2 + kH(0,l) + H(l,c_p)(k+l) + \sum_{j=1}^{l-1} H(j,l) + 2\sum_{j=1}^{p-1} H(c_j,c_p) \ . \end{aligned}$$

Therefore, $h^{\swarrow}(0) - h^{\swarrow}(c_p)$ equals

$$\begin{aligned} &= 2\sum_{j=1}^{p-1}[H(c_j,l) - H(c_j,c_p)] + \sum_{j=1}^{l-1}[H(j,0) - H(j,l)] + H(c_p,l) + 2pH(l,0) \\ &\quad - kH(0,l) - H(l,c_p)(k+l) \ . \end{aligned} \tag{1}$$

Now, fix $l \in \mathbb{N}$ so that $l > \left\lceil \frac{k(k+1)}{12} \right\rceil$. Then

$$l > \frac{k(k+1)}{12} \iff 6l > k + \frac{k(k-1)}{2} \quad . \tag{2}$$

Note that for k, l fixed, we can make p big enough so that the second sum of (1) is strictly greater than 0. Also, we have the following

$$\begin{aligned} &= 2\sum_{j=1}^{p-1}[H(c_j,l) - H(c_j,c_p)] \\ &= 2\sum_{j=1}^{p-1}\left[j(2p-j) - (p-j)\left(j + \frac{k(k-1)}{2} + l + p\right)\right] \\ &= 2\sum_{j=1}^{p-1}\left[j\left(\frac{k(k-1)}{2} + l + 2p\right) - p\left(\frac{k(k-1)}{2} + l + p\right)\right] \\ &= p(p-1)\left(\frac{k(k-1)}{2} + l + 2p\right) - p(p-1)(k(k-1) + 2l + 2p) \\ &= p(p-1)\left(-\frac{k(k-1)}{2} - l\right) \quad . \end{aligned}$$

Using these two facts and (1), we have that $h^{\swarrow}(0) - h^{\swarrow}(c_p)$ is strictly greater than

$$\begin{aligned} &> H(c_p,l) + 2pH(l,0) - kH(0,l) - H(l,c_p)(k+l) - p(p-1)\left(\frac{k(k-1)}{2} + l\right) \\ &> 2pH(0,l) - kH(0,l) - H(l,c_p)(k+l) - p(p-1)\left(\frac{k(k-1)}{2} + l\right) \\ &= 2pl(4p+l) - kl(k(k-1)+1) - p\left(p + \frac{k(k-1)}{2} + l\right)(k+l) \\ &\quad - p(p-1)\left(\frac{k(k-1)}{2} + l\right) \\ &> p^2\left(6l - k - \frac{k(k-1)}{2}\right) + p\left(\frac{k(k-1)}{2} + l\right)(1-k-l) - kl(k(k-1)+1) \ . \end{aligned}$$

Finally, note that the second and third therm have order $O(p)$ and $O(1)$ respectively. Therefore, because of (2) we have that for p large enough

$$h^{\swarrow}(0) - h^{\swarrow}(c_p) > 0,$$

that is, c_p is more central than vertex 0. □

Second is the *Density Axiom.* For this case two cycles of the same size are connected through a bridge. By symmetry both end points of the bridge have the same centrality. What should happen if we increase the number of edges of one of the cycles until it becomes a clique? Well, the centrality of the endpoint connected to the cycle that is becoming a clique should also increase. Formally stated:

Axiom 2: (Density axiom) Consider the graph $D_{k,p}$ made by a k-clique and a p-cycle $(p, k > 3)$ connected by a bridge uv, where u is a vertex of the clique and v one from the cycle. A centrality measure satisfies the density axiom if for $k = p$, u is strictly more central than v.

Proof. First, remember definition 7 made for bridges and note that in this case G_u corresponds to the k-clique and G_v to the p-cycle. Also, note that a k-clique has exactly $k(k-1)/2$ edges, while a p-cycle has p. Therefore, by using this fact and proposition 9 we have that

$$\begin{aligned}
k > 3 \ \wedge \ k = p &\implies k^3 - 3k^2 > 0 \\
&\iff k^3 - k^2 + k > 2k^2 + k \\
&\iff k(k(k-1)+1) > k(2k+1) \\
&\iff k\left(2\frac{k(k-1)}{2} + 1\right) > p(2p+1) \\
&\iff |V(G_u)|(2|E(G_u)|+1) > |V(G_v)|(2|E(G_v)|+1) \\
&\iff h^{\swarrow}(u) < h^{\swarrow}(v)
\end{aligned}$$

that is, u is strictly more central than v. □

Finally, there is the *Monotonicity Axiom.* It states that when adding an edge to a graph that originally did not have it, the centrality of both endpoints should increase.

Axiom 3: (Monotonicity axiom) Consider an arbitrary graph $G = (V, E)$ (with $|V| \geq 2$) and a pair of vertices u, v of G such that $uv \notin E$. A centrality measure satisfies the monotonicity axiom if when we add uv to G, the centrality of both vertices improves.

Proof. Note that is enough to prove it for vertex u. Write $e = uv$ and define $G' := (V, E \cup e)$. Also, we will use the notation $H_G(u,v)$ and $H_{G'}(u,v)$ for the old and new hitting times respectively, and we will do similarly for the centrality values $h^{\swarrow}_{G'}(u)$ and $h^{\swarrow}_{G}(u)$. Note that because we are only adding an edge, the set of vertices remain the same for both graphs, and therefore, we can refer to it simply by V.

We have that $\forall\ w \in V \setminus \{u\}$, $H_G(w,u) > H'_G(w,u)$. Indeed, whenever a random walk steps into v, in G' has the opportunity of going through e to reach u in only one more step, whereas in G it has to neccesarily take one more step into a neighbor of v, and then in the best case scenario, another one to reach u. Therefore, by using this we have that

$$h^{\swarrow}_{G'}(u) = \sum_{\substack{w\in V\\ w\neq u}} H_{G'}(w,u) < \sum_{\substack{w\in V\\ w\neq u}} H_G(w,u) = h^{\swarrow}_{G}(u)$$

that is, u has strictly better centrality in G' than in G. □

6 Conclusions

We studied a notion of centrality based on random walks over non-directed graphs. Besides experimental evidence (that we do not show in this article) this notion has nice theoretical properties.

Although in this paper we concentrated in proving that it satisfies the recently introduced axioms of centrality by Boldi-Vigna, the techniques used give an insight of their potential. In fact, it can be proved that our notion of centrality captures fine properties of central nodes in undirected graphs. In a future paper we will present these results.

Acknowledgments. The authors thank funding to Millennium Nucleus Center for Semantic Web Research under Grant NC120004.

References

1. Thad Huges & Daniel Ramge (2007). Lexical semantics relatedness with random graph walks. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp.581-589.
2. Joshua T. Abbott, Joseph L. Austerweil, Thomas L. Griffiths (2012). Human memory search as a random walk in a semantic network. In *Advances in Neural Information Processing Systems* 25, pp.3050-3058.
3. Camilo Garrido García (2013). Resúmenes Semiautomáticos de Conocimiento: caso de RDF. In *Memoria para optar al título de Ingeniero Civil en Computación*, `http://repositorio.uchile.cl/bitstream/handle/2250/113509/cf-garrido_cg.pdf?sequence=1&isAllowed=y`.
4. Linton C. Freeman (1978/79). Centrality in Social Networks Conceptual Clarification. In *Social Networks* 1, pp.2125-239.
5. Paolo Boldi, Sebastiano Vigna (2013). Axioms for Centrality. In *CoRR, vol. abs/*1308.2140.

6. L.Lovász (1993). Random Walks on Graphs; A Survey. In *Boltai Soc., Math. Studies* 2, pp.1-46.
7. Joe Dong Noh, Heiko Rieger (2004). Random Walks on Complex Networks. In *Physical Review Letters*, Volume 92, Number 11.
8. Shravas K Rao, *Finding Hitting Times in Various Graphs.*
9. M.E.J. Newman (2005). A measure of betweenness centrality based on random walks. In *Social Networks* 27, pp.39-54.

7 Appendix

Proposition 9. *Let uv be a bridge of a connected graph G. Then*

$$h^{\swarrow}(u) < h^{\swarrow}(v) \iff (2|E(G_u)|+1)|V(G_u)| > (2|E(G_v)|+1)|V(G_v)|$$

Proof. Note that $h^{\swarrow}(u)$ is equal to

$$\begin{aligned}
&= \sum_{\substack{w\in V(G)\\ w\neq u}} H(w,u)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,u) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,u) + H(v,u)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,u) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} [H(w,v)+H(v,u)] + H(v,u)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,u) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,v) + |V(G_v)|H(v,u)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,u) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,v) + |V(G_v)|(2|E(G_v)|+1)\\
&< \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,u) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,v) + |V(G_u)|(2|E(G_u)|+1)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,u) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,v) + |V(G_u)|H(u,v)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} [H(w,u)+H(u,v)] + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,v) + H(u,v)\\
&= \sum_{\substack{w\in V(G_u)\\ w\neq u}} H(w,v) + \sum_{\substack{w\in V(G_v)\\ w\neq v}} H(w,v) + H(u,v)\\
&= \sum_{\substack{w\in V(G)\\ w\neq v}} H(w,v)\\
&= h^{\swarrow}(v)\ .\square
\end{aligned}$$

Exploiting Semantics to Predict Potential Novel Links from Dense Subgraphs

Alejandro Flores, Maria-Esther Vidal, Guillermo Palma

Universidad Simón Bolívar, Caracas, Venezuela
{aflores, mvidal, gpalma}@ldc.usb.ve

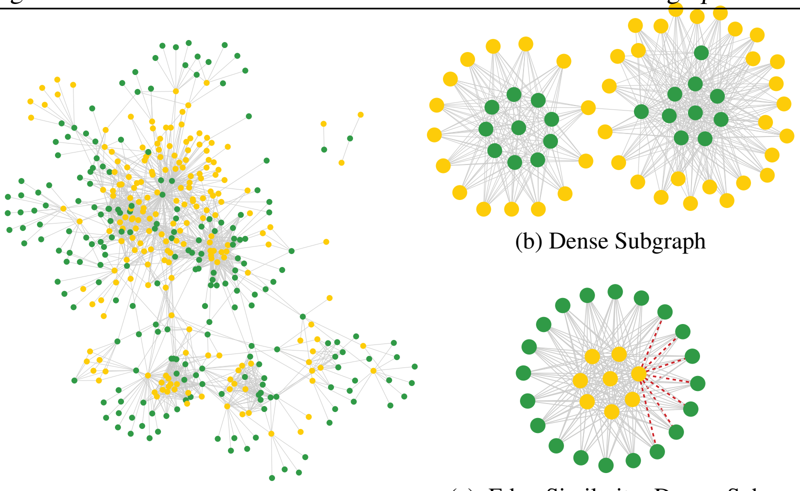
Abstract. Knowledge graphs encode semantic knowledge that can be exploited to enhance different data management tasks, e.g., query answering, ranking, or data mining. We tackle the problem of predicting interactions between drugs and targets, and propose esDSG, an unsupervised approach able to predict links from subgraphs that are not only highly dense, but that comprise both similar drugs and targets. The esDSG approach extends a state-of-the-art approximate densest subgraph algorithm with knowledge about the semantic similarity of the nodes in the original graph, and then predicts potential novel interactions from the computed dense subgraph. We have conducted an initial experimental study on a benchmark of drug-target interactions. Our observed results suggest that esDSG is able to identify interactions in graphs where existing approaches cannot perform equality well. Further, a large number of esDSG predictions can be validated using external databases as STITCH[1] and Kegg[2]. These results, although initial, reveal how semantics in conjunction with topological information of the knowledge graph may have a great impact on pattern discovery tasks.

1 Introduction

Knowledge graphs are networks of semantically related concepts, described in terms of attributes, types, and relationships. For example, biomedical knowledge graphs represent relationships among drugs, targets, side effects, and so on. Exploring these semantically rich networks can lead to novel discoveries [3], e.g., interactions between drugs and targets, or drug side effects. We focus on the problem of suggesting novel associations between concepts from knowledge graphs where concepts of the same type are characterized by similarity measures. We apply our approach to predict drug-target interactions; however, the approach is general enough to be used in other domains.

Different approaches have been proposed to predict interactions in networks [1, 4, 5, 10, 14, 15]; these approaches have been applied to drug-target graphs, where edges between drugs and targets represent drug-target interactions, and drug-drug and target-target similarity functions are defined. The *prediction hypothesis* is that similar drugs interact with the same targets, and similar targets interact with the same drugs [4]. The main challenge addressed by existing approaches is the development of strategies that allow for detecting portions of the network where novel predictions can be suggested. To illustrate the problem of suggesting drug-target interactions, consider Figures 1(a),

[1] `http://stitch.embl.de/`
[2] `http://www.genome.jp/kegg/`



(a) Drug-target interactions

(b) Dense Subgraph

(c) Edge-Similarity Dense Subgraph

Fig. 1: Ion Channel dataset, representing known interactions between targets (yellow) and drugs (green). Predicted interactions are shown in red-dashed lines.

(b), and (c). Figure 1(a) visualizes the network of drug-target interactions of Ion Channels in the dataset studied by Ding et al. [3]; drug-drug and target-target relationships are not drawn. To avoid the pair-wise comparison of all the drugs and targets in the graph, existing approaches limit the search to dense portions of the graph that comprise both similar drugs and similar targets. For example, the semEP [10] approach represents the network of drug-target interactions as a bipartite graph, and partitions the edges (interactions) of the graph according to a density function that measures how similar are the drugs and targets that are placed in one partition. Further, the number of interactions between drugs and targets in each partition are taken into account by this measure. semEP is able to explore portions of the search space and suggest predictions that other state-of-the-art approaches do not discover. However, in graphs such as the one reported in Figure 1(a), semEP and state-of-the-art approaches do not exhibit as good behavior as in other networks. The Ion Channel graph is composed of 414 nodes and 1,476 edges. It also has three connected components; two of these components are comprised of five nodes and three edges. The rest of the nodes are part on the main connected component, from which the dense subgraph of the original network can be computed; this dense subgraph is presented in Figure 1(b). We hypothesize that in graphs with this topology, limiting the search to highly dense components could lead to better predictions. However, dissimilar drugs or targets may comprise these highly connected components. We address the problem of identifying subgraphs that are not only dense, but are composed of highly similar drugs and targets. We name these graphs *edge-similarity dense subgraphs*; Figure 1(c) shows an *edge-similarity dense subgraph* for the Ion Channel dataset. Evaluating the *prediction hypothesis* in this subgraph allows for suggesting seven predictions shown in red-dashed lines.

We propose a novel technique named esDSG, that is able to produce edge-similarity dense subgraphs and to suggest novel associations from these subgraphs. We have conducted an experimental study on the benchmark of drug-target interactions studied by Ding et al. [3]. Interactions suggested by semEP and five state-of-the-art machine learning based techniques are compared to the interactions produced by esDSG. The observed results suggest that the search spaces explored by all these approaches are different. Further, esDSG identifies interactions in graphs where existing approaches cannot perform equality well. More importantly, a large number of the esDSG predictions can be validated using the external databases STITCH and Kegg. These results suggest that considering the topology in conjunction with semantics encoded in the graph, e.g., similarity values, can have a positive impact on the effectiveness of discovery tasks.

The rest of the paper is as follows: Section 2 describes the esDSG approach and Section 3 summarizes related approaches. Experimental results are presented in Section 4. Finally, we conclude in Section 5 with an outlook to future work.

2 Edge-Similarity Densest Subgraph Problem

Our goal is to discover novel drug-target interactions by exploiting relationships among existing interactions and values of similarity between both drugs and targets. However, to effectively suggest such interactions, the *prediction hypothesis* should be evaluated in graphs that maximize: *i*) graph density, and *ii*) drug-drug and target-target similarities. We cast the problem of generating these graphs into a single-objective optimization problem, called the edge-similarity Densest Subgraph (esDSG) problem.

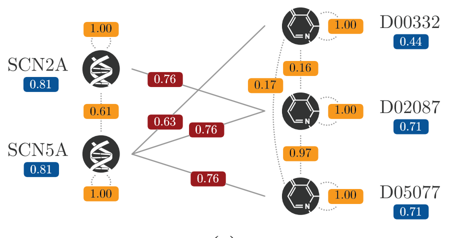
Let $D = \{d_1, d_2, \ldots, d_n\}$ be a set of drugs, and let $s_d(d_i, d_j) \in [0 \ldots 1]$ be the similarity score function defined between every two drugs in D. Similarly, let $T = \{t_1, t_2, \ldots, t_m\}$ be a set of targets, and let $s_t(t_i, t_j) \in [0 \ldots 1]$ be the similarity score function defined between every two targets in T. Given the sets T and D, we call E the set of interactions between targets and drugs, where $E \subseteq T \times D$. These interactions can be naturally represented as a bipartite graph $G = (T, D, E)$.

For each node v in the subgraph, the goal is to maximize not only the number of interactions (i.e., the degree), but also the average of the pair-wise similarity values of v with respect to the rest of the nodes in the subgraph. To do this, we define the similarity measure of a single node v as follows:

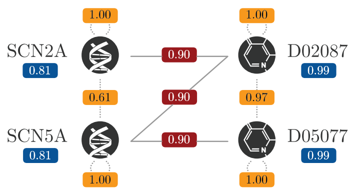
Definition 1 (Single-node similarity). *Consider a bipartite graph $G = (T, D, E)$ of drug-target interactions. The single-node similarity of $v \in T$ (resp., $v \in D$) corresponds to the arithmetic mean of the similarity measure values of v with every node $v' \in T$ (resp., $v' \in D$). This is denoted as $s_n(v)$.*

$$s_n(v) = \begin{cases} \frac{1}{|T|} \sum\limits_{t \in T} s_t(v, t) & \textit{if } v \in T \\ \frac{1}{|D|} \sum\limits_{d \in D} s_d(v, d) & \textit{if } v \in D \end{cases} \tag{1}$$

Figure 2(a) presents a subgraph of the Ion Channel dataset. Similarity values allow for computing the *single-node similarity* of each node. For example, the drug D00332 is dissimilar to the rest of the drugs in the subgraph. This is represented with low values of the *single-node similarity*, i.e., $s_n(\text{D00332}) = \frac{1}{3}\big(1 + 0.16 + 0.17\big)$. On the other hand, the drug D05077 is highly similar to the rest of the drugs in this subgraph; thus, a higher value of *single-node similarity* is assigned to D05077, i.e., $s_n(\text{D05077}) = \frac{1}{3}\big(1 + 0.97 + 0.17\big)$. Note that *single-node similarity* values depend on the context in which they are computed, i.e., only the nodes in a particular subgraph are considered.



(a)



(b)

Fig. 2: Subgraphs of drug-target interactions of the Ion Channel dataset, where (b) is subgraph of (a). Drug-Drug and Target-Target similarities in yellow, single-node similarities in blue, and edge similarities in red.

We can characterize edges in a graph based on the *single-node similarity* values of the nodes that connect each edge. For example, because D00332 is less similar than D05077 to the rest of the drugs in the subgraph of Figure 2(a), the interaction that relates D00332 with the target SCN5A is not as important in terms of similarity as the interaction between D05077 the same target SCN5A. We model the importance of an interaction in terms of similarity as a function named *edge similarity*, that averages the single node similarities of the target and the drug that are related with this interaction.

Definition 2 (Edge similarity). *Consider a bipartite graph $G = (T, D, E)$ of drug-target interactions, and the single-node similarity value of every drug and target in the graph. The edge similarity score of an interaction $(t, d) \in E$, denoted as $s_e(t, d)$, corresponds to the arithmetic mean of the single-node similarity values of t and d.*

$$s_e(t, d) = \frac{1}{2}\big(s_n(t) + s_n(d)\big) \tag{2}$$

Based on this definition, the *edge similarity* of the interaction between D00332 and SCN5A is 0.63, while the *edge similarity* of the interaction between D05077 and SCN5A is 0.76. These values of the *edge similarity* function indicate the importance of these two interactions in terms of the similarity of the drugs and targets that they relate, with respect to the rest of the drugs and targets in the subgraph.

Normally, the definition of density for a bipartite graph, denoted as $D(G)$, is given by $D(G) = \frac{|E|}{\sqrt{|T||D|}}$. Using this definition, each edge of the graph is equally relevant in terms of the density of the graph. Our approach relies on the importance of edges in terms of similarity, i.e., the *edge similarity*, to compute a similarity-based density.

Definition 3 (Edge-Similarity Density). *Consider a bipartite graph $G = (T, D, E)$, where T represents the set of targets, D the set of drugs, and E the set of drug-target interactions. Let $s_e(t, d)$, for $(t, d) \in E$, be the edge similarity value for every edge in the graph. The Edge-Similarity Density of G can be defined as:*

$$D_{es}(G) = \frac{\sum\limits_{(t,d)\in E} s_e(t,d)}{\sqrt{|T||D|}} \tag{3}$$

Using the *edge similarity* values computed for every edge in the subgraph in Figure 2(a), the *edge-similarity density* is equal to 1.19, while the graph density for this subgraph is 1.63. Removing the drug D00332 from this subgraph results in the subgraph G' in Figure 2(b) with a decreased density value of 1.50. However, the *edge-similarity density* in this new graph is higher, i.e., $D_{es}(G') = 1.35$. Applying the *prediction hypothesis* to this subgraph would result in the prediction of an interaction between the target SCN2A and the drug D05077; this interaction can be validated in Kegg.

Based on the definition of the *edge-similarity density*, we define the Edge-Similarity Densest Subgraph problem (esDSG) as follows:

Definition 4 (Edge-Similarity Densest Subgraph problem (esDSG)). *Given a bipartite graph $G = (T, D, E)$ as described above, representing the set of interactions between targets and drugs, esDSG identifies the subgraph $G^* \subseteq G$ such that the edge-similarity density of G^*, $D_{es}(G^*)$, is maximized.*

2.1 A greedy approximate algorithm for esDSG

We propose a greedy approximate algorithm to the esDSG problem. The resulting subgraphs are both highly dense and similar, and they will be used to predict novel interactions between targets and drugs. The *edge-similarity density* is used to guide the greedy algorithm into spaces of dense and similar subgraphs.

The greedy decision of removing the nodes that contribute the least to the *edge-similarity density* may result in higher values of the measure. For example, as shown in Figure 2, removing the drug D00332 with both minimum *single-node similarity* and degree values, as well as removing the interaction between this drug and the target SCN5A, leads to an increase of the *edge-similarity density* from 1.19 to 1.35.

After applying the definition of *edge similarity* and some algebraic manipulation, the *edge-similarity density* can be reformulated as a function of the degree values and the *single-node similarity* of each node in a graph G, as follows:

$$D_{es}(G) = \frac{\sum\limits_{v\in T\cup D} degree(v) \cdot s_n(v)}{2\sqrt{|T||D|}} \tag{4}$$

We propose a modification of the densest subgraph 2-approximation algorithm proposed by Khuller and Barna [7]. In our algorithm, the greedy decision will be made not only in terms of the degree, but also the *single-node similarity* of the nodes of the graph. Along with the use of the *edge-similarity density*, this constitutes the most important difference with Khuller & Barna's algorithm, where the greedy strategy indicates the removal of nodes, according only to the degree.

Algorithm 1: Edge-Similarity Dense Subgraph

```
  Input: A bipartite graph $G = (T, D, E)$
  Output: A subgraph $H^* \subseteq G$
1 $i \leftarrow |T| + |D|, H_i \leftarrow G$
2 while $H_i \neq \varnothing$ do
3     Let $v$ be a vertex with minimum value $degree(v) \cdot s_n(v)$ in $H_i$
4     Delete all incident edges in $v$
5     Delete all nodes with no incident edges
6     Update single-node similarity values of all nodes
7     Call the new graph $H_{i-1}, i \leftarrow i - 1$
8 return $H^*$, a graph with maximum edge-similarity density, $D_{es}(H^*)$, from all the $H_i$'s
```

Algorithm 1 performs an ordered removal of the nodes of a bipartite graph, removing a node v with minimum product of the *single-node similarity* and degree values, i.e., a node such that $degree(v) \cdot s_n(v)$ is the minimum value. Further, the incident edges on v are removed from the graph. The algorithm iteratively performs this greedy decision until no more edges can be removed. In each iteration i of the algorithm (**while loop** in lines *2* to *7*), a subgraph H_i is created; the algorithm outputs the subgraph H^* with the maximum value of $D_{es}(H^*)$. Predictions correspond to the missing edges in H^*. For example, consider the subgraph in Figure 1(c), this is a dense graph produced by Algorithm 1 for the Ion Channel dataset; seven red-dashed lines represent missing edges in this subgraph and correspond to the interactions predicted by our approach. It is important to highlight that four out of the seven predicted interactions could be validated in STITCH with prediction score greater than 0.16; only two of these interactions were among the top-10 predictions of semEP.

Finally, for a bipartite graph $G = (T, D, E)$, Khuller's algorithm requires $\mathcal{O}(|E| + |T \cup D|)$ time, while the time complexity of Algorithm 1 is $\mathcal{O}(|T|^2 + |D|^2)$. The increased complexity is a consequence of the need to compute the *single-node similarity* values. i.e., after each removal the sets of drugs and targets changes, thus, the *single-node similarity* values of the nodes in the graph need to be updated.

3 Related Work

We briefly describe existing approaches for graph data mining and link prediction. Graph mining approaches focus on the problem of detecting patterns in graphs by conducting clustering and ranking methods on knowledge graphs [6, 9, 12]. RankClus [12], GNetMine [6], and JointCluster [9] apply clustering techniques to place multi-typed concepts in the same cluster. RankClus exploits link analysis-based ranking with clustering to assign highly ranked entities to highly ranked clusters. GNetMine applies learning based methods on different graph properties, e.g., graph topology, type attributes, or edge labels, to classify and partition nodes in a graph. Finally, JointCluster [9] is a *simultaneous clustering* such that nodes within each cluster in the partition are highly connected, and the number of inter-cluster edges is minimized. These clustering methods have shown to be effective, but they cluster nodes instead of edges. Further, they mainly focus on the topology of the graph and do not exploit any semantics, constraints, or similarity information between the nodes to enhance the quality of the clustering. Dissimilar nodes may be placed in a cluster, reducing thus the chances to predict edges whenever the *prediction hypothesis* is evaluated.

Several approaches have been proposed for link prediction, and particularly, to suggest annotations for genes [11, 13] and interactions between drugs and targets [1, 4, 5, 10, 14, 15]. Techniques to identify dense subgraphs [11] and graph summarization [13] have been extended to predict Gene Ontology annotations of genes that comprise a clique or from super nodes in a summary of a graph, respectively. These approaches have shown to successfully analyze patterns in graphs. However, because they do not consider semantics encoded in the graph or similarity between the graph concepts, they may be grouping dissimilar concepts in a dense graph or a summary of a graph. Thus, applying the *prediction hypothesis* graphs produced by these approaches may lead to a small dataset of predictions or just to produce no predictions at all.

Machine learning methods [1, 4, 5, 14, 15] have been exploited to identify the portions of the knowledge graph where the application of the *prediction hypothesis* can lead to the discovery of drug-target interactions; a detailed evaluation of all these approaches can be found at [3]. Additionally, semEP [10] implements an unsupervised method able to partition edges that represent drug-target interactions; the *prediction hypothesis* is applied to each component of the partition to predict new interactions. Components of a partition maximize the graph density and pair-wise similarity of the drugs and the targets. Although all these approaches are able to suggest drug-target interactions that can be validated, they may not exhibit the same good performance in graphs as the one illustrated in Figure 1(a). We hypothesize that because esDSG reduces the prediction search to the space of edge-similarity dense graphs, the overlap between the esDSG predictions and the ones suggested by the other approaches will be small; further, esDSG may be able to exhibit better performance from this type of graphs.

4 Evaluation of esDSG and State-of-the-Art Methods

Dataset: We use the dataset created by Bleakley et al. [1] to evaluate esDSG quality. This dataset is used in the comparison of existing interaction prediction methods [3, 10]. The dataset comprises 900 drugs, 1,000 targets, and 5,000 interactions from KEGG BRITE[3], BRENDA[4], SuperTarget[5], and DrugBank[6]. The dataset provides a drug-drug chemical similarity score based on the hashed fingerprints from the SMILES resource, and a target-target similarity score based on the normalized Smith-Waterman sequence similarity score. Data is divided into four groups: Nuclear receptors (NR), Gprotein-coupled receptors (GPCR), Ion channels (IC) and Enzymes (E); details in Table 1.
State-of-the-art Methods: We use semEP [10] and the following machine learning methods reported in [3] to evaluate the quality of the esDSG predictions: *i*) BLM:

[3] `http://www.genome.jp/kegg/brite.html`
[4] `http://www.brenda-enzymes.de/`
[5] `http://bioinf-apache.charite.de/supertarget_v2/`
[6] `http://www.drugbank.ca/`

Table 1: Statistics for the Drug-Target Interaction Dataset [1].

Statistics	**NR**	**GPCR**	**IC**	**E**
Number of drugs	54	223	210	445
Number of targets	26	95	204	664
Interactions	90	635	1,476	2,926
Average interaction count per target	3.46	6.68	7.23	4.4
Average interaction count per drug	1.66	2.84	7.02	6.57
Density	2.40	4.36	7.13	5.38

Bipartite Local Method [2]; *ii*) LapRLS: Laplacian Regularized Least Squares [15]; *iii*) GIP: Gaussian Interaction Profile [14]; *iv*) KBMF2K: Kernelized Bayesian Matrix Factorization with twin Kernels [5]; and *v*) NBI: Network-Based Inference [2]. Additionally, Khuller & Barna's algorithm is used as baseline.

4.1 Dense Subgraphs and Edge-Similarity Dense Subgraphs

We compare the dense subgraphs, DSG and esDGS, produced by Khuller & Barna's algorithm and our approach, respectively. Table 2 describes these subgraphs in terms of number of drugs, targets, interactions, density, and average similarities. We can observe that DSGs are larger than esDSGs and also have greater values of the *standard* density. For datasets NR, IC, and E, the esDGS graphs are denser in terms of *edge-similarity density*, and result on less interactions but all predicted from similar drugs or targets. However, although the DSG approximation algorithm does not maximize

Table 2: Statistics of the Dense Subgraphs and Edge-Similarity Dense Subgraphs computed for each graph in the Drug-Target Interaction Dataset [1].

Statistics	**NR**		**GPCR**		**IC**		**E**	
	DSG	esDSG	DSG	esDSG	DSG	esDSG	DSG	esDSG
Targets	6	5	9	9	74	8	61	13
Drugs	3	3	16	13	53	20	37	13
Interactions	18	15	107	87	815	153	994	168
Predictions	0	0	37	30	3107	7	1263	1
Density	**4.24**	3.87	**8.91**	8.04	**13.01**	12.09	**20.92**	12.9
Edge-Similarity Density	1.94	**2.01**	**2.66**	2.65	1.79	**6.88**	3.47	**5.53**
Average Target Similarity	0.40	**0.52**	0.33	0.33	0.06	**0.69**	0.10	**0.54**
Average Drug Similarity	0.50	0.50	0.25	**0.32**	0.21	**0.43**	0.23	**0.31**

edge-similarity density, DSG has higher values of this measure than esDSG on GPCR. Further, DSG and esDSG have 24 common predicted interactions. This suggests that GPCR regions that are dense or similar are the same, i.e., the search spaces of the approximation algorithms for DSG and esDSG overlap.

4.2 Validation using STITCH and Kegg

We evaluate the quality of esDSG predictions in terms of the number of interactions that can be validated. We use the latest online version of STITCH [8] and Kegg to validate each drug-target interaction predicted by esDSG. Table 3(a) reports on both the number of esDSG validated drug-target interactions, and the total number of predicted interactions. We can observe that esDSG predicted 12 out of 30 predictions that could be validated in STITCH and Kegg; while the 57% of the esDSG predicted interactions in the Ion Channel dataset can be also validated. As observed in Table 2, the IC esDSG is the one with the highest values of *edge-similarity density*, and average target and drug similarities. Furthermore, Table 3(b) reports on the number of validated interactions out of the top-5 novel predicted interactions of all methods; novel interactions are not in the original datasets and have the highest prediction score; these scores are computed by each prediction method. It is clear that esDSG outperforms existing approaches in the Ion Channel dataset. This supports our hypothesis that novel interactions can be predicted from highly dense graphs composed of similar drugs and targets.

Table 3: (a) Number of esDSG predicted interactions and the number of those interactions that can be manually validated with STITCH and Kegg. (b) Top-5 novel interactions manually validated with STITCH; entries highlighted in bold correspond to the largest number of novel validations

(a) esDSG Validated Predictions

Dataset	**Number of predictions**	**Validation**		
		STITCH	**Kegg**	**Total**
NR	0	-	-	-
GPCR	30	11	3	12
IC	7	4	0	4
E	1	0	0	0

(b) Validated Predictions of State-of-the Prediction Methods

Method	**NR**	**GPCR**	**IC**	**E**
semEP	4	**5**	1	**4**
BLM	2	1	0	0
NBI	1	1	1	2
GIP	3	3	1	1
LapRLS	**5**	3	**2**	2
KBMF2K	3	4	**2**	2

4.3 Effectiveness of esDSG Predictions

Effectiveness of esDSG is also measured as the overlap between the esDSG predicted interactions and the top-10 interactions predicted by semEP [10], and the machine learning methods: BLM, NBI, GIP, LapRLS, and KBMF2K [3]. Top-10 predictions correspond to the interactions with the highest prediction score. Table 4 compares esDSG and the corresponding prediction method in terms of both: *i*) the number of interactions that both methods predicted, labelled as *Equal*, and *ii*) the number of interactions predicted by esDSG and that are not part of the top-10 predictions of the corresponding method, labelled as *Diff*. We can observe that esDSG predicts interactions that are not predicted by any of the other methods. These results suggest that the edge-similarity subgraphs from where esDSG generates the predicted interactions, correspond to portions of the search space that none of the other approaches is able to explore.

5 Conclusions

We defined the *edge-similarity densest subgraph* problem to predict drug-target interactions. We extended Kuller & Barna's algorithm with the greedy decision of removing the nodes that contribute less to the *edge-similarity density* values. Experimental results suggest our approach is able to predict novel drug-target interactions from highly dense

Table 4: Overlap of esDSG predictions and the top-10 predictions generated by existing methods [3, 10]. Entries in bold indicate that all esDSG predictions are different

Method	**NR**		**GPCR**		**IC**		**E**	
	Equal	Diff.	Equal	Diff.	Equal	Diff.	Equal	Diff.
semEP	0	0	0	**30**	3	4	0	**1**
BLM	0	0	0	**30**	0	**7**	0	**1**
NBI	0	0	4	26	0	**7**	1	0
GIP	0	0	2	28	4	3	1	0
LapRLS	0	0	0	**30**	3	4	1	0
KBMF2K	0	0	0	**30**	0	**7**	0	**1**

subgraphs composed of both similar drugs and targets. We plan to extend esDSG to traverse the different connected components and identify esDSGs from each one.

References

1. K. Bleakley and Y. Yamanishi. Supervised prediction of drug–target interactions using bipartite local models. *Bioinformatics*, 25(18):2397–2403, 2009.
2. F. Cheng, C. Liu, J. Jiang, W. Lu, W. Li, G. Liu, W. Zhou, J. Huang, and Y. Tang. Prediction of drug-target interactions and drug repositioning via network-based inference. *PLoS computational biology*, 8(5):e1002503, 2012.
3. H. Ding, I. Takigawa, H. Mamitsuka, and S. Zhu. Similarity-based machine learning methods for predicting drug-target interactions: A brief review. *Briefings in Bioinformatics*, 2013.
4. S. Fakhraei, B. Huang, L. Raschid, and L. Getoor. Network-based drug-target interaction prediction with probabilistic soft logic. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2014.
5. M. Gönen. Predicting drug–target interactions from chemical and genomic kernels using bayesian matrix factorization. *Bioinformatics*, 28(18):2304–2310, 2012.
6. M. Ji, Y. Sun, M. Danilevsky, J. Han, and J. Gao. Graph regularized transductive classification on heterogeneous information networks. In *Machine Learning and Knowledge Discovery in Databases*, pages 570–586. Springer, 2010.
7. S. Khuller and B. Saha. On finding dense subgraphs. In *Automata, Languages and Programming*, pages 597–608. Springer, 2009.
8. M. Kuhn, D. Szklarczyk, A. Franceschini, C. von Mering, L. J. Jensen, and P. Bork. Stitch 3: zooming in on protein–chemical interactions. *Nucleic acids research*, 40(D1), 2012.
9. M. Narayanan, A. Vetta, E. E. Schadt, and J. Zhu. Simultaneous clustering of multiple gene expression and physical interaction datasets. *PLoS Computational Biology*, 6(4), 2010.
10. G. Palma, M. Vidal, and L. Raschid. Drug-target interaction prediction using semantic similarity and edge partitioning. In *ISWC*, 2014.
11. B. Saha, A. Hoch, S. Khuller, L. Raschid, and X. Zhang. Dense subgraphs with restrictions and applications to gene annotation graphs. In *RECOMB*, 2010.
12. Y. Sun, J. Han, P. Zhao, Z. Yin, H. Cheng, and T. Wu. Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, 2009.
13. A. Thor, P. Anderson, L. Raschid, S. Navlakha, B. Saha, S. Khuller, and X. Zhang. Link prediction for annotation graphs using graph summarization. In *ISWC*, 2011.
14. T. van Laarhoven, S. B. Nabuurs, and E. Marchiori. Gaussian interaction profile kernels for predicting drug–target interaction. *Bioinformatics*, 27(21), 2011.
15. Z. Xia, L.-Y. Wu, X. Zhou, and S. T. Wong. Semi-supervised drug-protein interaction prediction from heterogeneous biological spaces. *BMC systems biology*, 4(Suppl 2):S6, 2010.

On the CALM Principle for BSP Computation

Matteo Interlandi [1] and Letizia Tanca [2]

[1] University of California, Los Angeles
minterlandi@cs.ucla.edu
[2] Politecnico di Milano,
letizia.tanca@polimi.it

Abstract. In recent times, considerable emphasis has been given to two apparently disjoint research topics: *data-parallel* and *eventually consistent, distributed* systems. In this paper we propose a study on an *eventually consistent, data-parallel computational model*, the keystone of which is provided by the recent finding that a class of programs exists that can be computed in an eventually consistent, coordination-free way: *monotonic programs*. This principle is called CALM and has been proven by Ameloot et al. for distributed, asynchronous settings. We advocate that CALM should be employed as a basic theoretical tool also for data-parallel systems, wherein computation usually proceeds synchronously in rounds and where communication is assumed to be reliable. We deem this problem relevant and interesting, especially for what concerns *parallel workflow optimization*, and make the case that CALM does not hold in general for data-parallel systems if the techniques developed by Ameloot et al. are directly used. In this paper we sketch how, using novel techniques, the satisfiability of the if direction of the CALM principle can still be obtained, although just for a subclass of monotonic queries.

1 Introduction

Recent research has explored ways to exploit different levels of *consistency* in order to improve the performance of distributed systems *w.r.t.* specific tasks and network configurations, while maintaining correctness [18]. A topic strictly related to consistency is *coordination*, usually informally interpreted as a mechanism to accomplish a distributed agreement on some system property [8]. Indeed, coordination can be used to enforce consistency when, in the natural execution of a system, this is not guaranteed in general. In this paper we sketch some theoretical problems springing from the use of eventually consistent, coordination-free computation over *synchronous systems with reliable communication* (*rsync*). Informally, such systems have the following properties: (i) a global clock is defined and accessible by every node; (ii) the relative difference between the time clock values of any two nodes is bounded; and (iii) the results emitted by a node arrive at destination at most after a certain *bounded physical time* (the so-called *bounded delay guarantee*).

Rsync is a common setting in modern data-parallel frameworks - such as MapReduce - in which computation is usually performed in *rounds*, where each task is blocked and cannot start the new round until a *synchronization barrier* is reached, *i.e.,* every other task has completed its local computation. In this work we consider synchronization (barrier) and coordination as two different, although related entities: the former is a *mechanism* enforcing the *rsync* model, the latter a *property of executions*. Identifying under what circumstances eventually consistent, coordination-free computation can be employed over *rsync* systems would enable us to "stretch" the declarativeness of parallel programs, freeing execution plans of the restriction to follow predefined (synchronous) patterns. In fact, all recent high-level data-parallel languages suffer from this limitation, for instance both Hive [16] and Pig [15] sacrifice pipelining in order to fit query plans into MapReduce workflows. Our aim is then to understand when a synchronous "blocking" computation is actually required by the program semantics – and therefore must be strictly enforced by the system – and when, instead, a pipelined execution can be performed as optimization. For batch parallel processing, the benefits of understanding where the former can be replaced by the latter are considerable [7]: thanks to the fact that data is processed as soon as it is produced, *online computation* is possible, *i.e.,* the final result can be refined during the execution; as a consequence, new data can be incrementally added to the input, making *continuous computation* possible. Overall, pipelining is highly desirable in the Big Data context, where full materialization is often problematic.

Recently, a class of programs that can be computed in an eventually consistent, coordination-free way has been identified: *monotonic programs* [9]; this principle is called *CALM* (Consistency and Logical Monotonicity) and has been proven in [4]. While CALM was originally proposed to simplify the specification of distributed (asynchronous) data management systems, in this paper we advocate that CALM should be employed as a basic theoretical tool also for the declarative specification of data-parallel (synchronous) systems. As a matter of fact, CALM permits to link a property of the execution (coordination-freedom) with a class of programs (monotonic queries). But to which extent CALM can be applied over data-parallel systems? Surprisingly enough, the demonstration of the CALM principle in *rsync* systems is not trivial and, with the communication model and the notion of coordination as defined in [4], the CALM principle does not hold in general in *rsync* settings (cf. Example 3). Thus, in order to extend CALM over data-parallel synchronous computation, in this paper we sketch a new *generic parallel computation model* leveraging previous works on *synchronous Datalog* [10, 12] and *transducer networks* [4], and grounding *rsync* computation on the well-known *Bulk Synchronous Parallel* (BSP) model [17] equipped with content-based addressing. With BSP, computation proceeds as a series of global rounds, each composed by three phases: (i) a *computation phase*, in which nodes parallelly perform local computations; (ii) a *communication phase*, where data are exchanged among the nodes; and (iii) the synchronization barrier. Exploiting this new type of transducer network, we will then show that the CALM principle is satisfied for synchronous and reliable systems *under a new definition of coordination-freedom*, although, surprisingly enough, just for a subclass of monotonic queries, *i.e.,* the *chained monotonic queries* (cf. Definition 5.7). When defining coordination-freedom we will take advantage of recent results describing how knowledge can be acquired in synchronous systems [5, 6].

Organization: The paper is organized as follows: Section 2 introduces some preliminary notation. Section 3 defines our model of synchronous and reliable parallel system, and shows that the CALM principle is not satisfied for systems of this type. Section 3.2 proposes a new computational model based on hashing, while Section 4 introduces the new definition of coordination. Finally, Section 5 discusses CALM under the new setting. The paper ends with some concluding remarks. We refer the reader to [11] for proofs and more detailed discussions.

2 Relational Transducers

In this paper we expect the reader to be familiar with the basic notions of database theory and relational transducer (networks). In this section we use some example to set forth our notation, which is close to that of [1] and [4].

We employ a transducer (resp. a transducer network) as an abstraction modeling the behavior of a single computing node (resp. a network of computing nodes): this abstract computational model permits us to make our results as general as possible without having to rely on a particular framework, since transducers and transducer networks can be easily imposed over any modern data-parallel system. We consider each node to be equipped with an immutable *database* and a *memory* used to store useful data between any two consecutive computation steps. In addition, a node can produce an *output* for the user and can also *communicate* some data to other nodes (the concept of data communication in a transducer network will appear clearer in Section 3). Finally, an internal *time*, and *system* data are kept mainly for configuration purposes. Every node executes a *program* that operates on (input) instances of the database, the memory and the communication channel, and produces new instances that are either saved in memory, or directly output to the user, or addressed to other nodes.

Example 1. A first example of relational transducer is the following UCQ-transducer $\mathcal{T}$, with schema Υ, that computes the ternary relation Q as the join between two binary relations R and T:

Schema: $\Upsilon_{db} = \{R^{(2)}, T^{(2)}\}, \Upsilon_{mem} = \emptyset, \Upsilon_{com} = \emptyset, \Upsilon_{out} = \{Q^{(3)}\}$
Program: $Q_{out}(u, v, w) \leftarrow R(u, v), T(v, w).$

Let $\mathbf{I}$ be an initial instance over which we want to compute the join. Then, let us define $I_{db} = \mathbf{I}$ as an instance over the $database$ schema Υ_{db}. A transition $I \rightarrow J$ for $\mathcal{T}$ is such that $I = \mathbf{I} \cup I_{sys}$, I_{rcv} and J_{snd} are empty (no communication query exists), and $J = \mathbf{I} \cup I_{out} \cup I_{sys}$, where I_{out} is the result of the query Q_{out}, *i.e.,* the join between R and T. Note that the subscript in Q_{out} means that this is an *output* query, that is, it specifies the final result of the whole computation.

3 Computation in rsync

In order to allow *query evaluation in parallel settings*, we will sketch a novel *transducer network* [4], where computation is *synchronous*, and communication is *reliable*. This permits us to define how a set of relational transducers can be assembled to obtain an abstract computational model for *distributed data-parallel systems*. To be consistent with [4], we will assume broadcasting as the addressing model.

Example 2. Assume we want to compute a distributed version of the join of Example 1. We can implement it using a broadcasting synchronous transducer network which emits one of the two relations, say T, and then joins R with the received facts over T. Note that the sent facts will be used just starting from the successive round, and the program will then employ two rounds to compute the distributed join. UCQ is again expressive enough. The transducer network can be written as follows – where S_{snd} denotes a *communication* query and this time schema Υ_{com} is non-empty because communication is needed:

Schema: $\Upsilon_{db} = \{R^{(2)}, T^{(2)}\}, \Upsilon_{com} = \{S^{(2)}\}, \Upsilon_{out} = \{Q^{(3)}\}$
Program: $S_{snd}(u, v) \leftarrow T(u, v).$
$Q_{out}(u, v, w) \leftarrow R(u, v), S(u, w).$

Synchronous specifications have the required expressive power:

Lemma 1 *Let $\mathcal{L}$ be a language containing* UCQ *and contained in* DATALOG$^\neg$. *Every query expressible in $\mathcal{L}$ can be distributively computed in 2 rounds by a broadcasting $\mathcal{L}$-transducer network.*

The above lemma permits us to draw the following conclusion: under the *rsync* semantics, monotonic and non-monotonic queries behave in the same way: two rounds are needed in both cases. This is due to the fact that, contrary to what happens in the asynchronous case of [4], starting from the second round we are guaranteed – by the reliability of the communication and the synchronous assumption – that every node will compute the query over every emitted instance. Conversely, in the asynchronous case, as a result of the non-determinism of the communication, we are never guaranteed, without coordination, that every sent fact will be actually received.

3.1 The CALM Conjecture

The *CALM conjecture* [9] specifies that a well-defined class of programs can be distributively computed in an *eventually consistent*, *coordination-free* way: *monotonic programs*. CALM has been proven in this (revisited) form for asynchronous systems [4]:

Conjecture 1 *A query can be distributively computed by a coordination-free transducer network if and only if it is monotonic.*

The concept of coordination suggests that all the nodes in a network must exchange information and wait until an agreement is reached about a common property of interest. Following this intuition, Ameloot et al. established that a specification is coordination-free if communication is not strictly necessary to obtain a consistent final result. Surprisingly enough, under this definition of coordination-freedom, CALM does not hold in *rsync* settings under the broadcasting communication model:

Example 3. Let $\mathcal{Q}_{out}$ be the "emptiness" query of [4]: given a nullary database relation $R^{(0)}$ and a nullary output relation $T^{(0)}$, $\mathcal{Q}_{out}$ outputs true (*i.e.,* a nullary fact over T) iff I_R is empty. The query is non-monotonic: if I_R is initially empty, then T is produced, but if just one fact is added to R, T is not derived, *i.e.,* I_T must be empty. A FO-transducer network $\mathcal{N}$ can be easily generated to distributively compute $\mathcal{Q}_{out}$: first every node emits R if its local partition is not empty, and then each node locally evaluates the emptiness of R. Since the whole initial instance is installed on every node when R is checked for emptiness, T is true only if R is actually empty on the initial instance. The complete specification follows.

Schema: $\Upsilon_{db} = \{R^{(0)}\}, \Upsilon_{mem} = \{\mathtt{Ready}^{(0)}\}, \Upsilon_{com} = \{S^{(0)}\}, \Upsilon_{out} = \{T^{(0)}\}$.
Program: $S_{snd}() \leftarrow R()$.
$\mathtt{Ready}_{ins}() \leftarrow \neg\mathtt{Ready}()$.
$T_{out}() \leftarrow \neg S(), \mathtt{Ready}()$.

One can show [11] that, if communication is switched off, the above transducer is still able to obtain the correct result if, for example, **I** is installed on every node. That is, a partitioning exists, making communication not strictly necessary to reach the proper result. Note that the same query requires coordination in asynchronous settings: since emitted facts are non-deterministically received, the only way to compute the correct result is that nodes coordinate to understand if the input instance is *globally* empty.

The result we have is indeed interesting although expected: when we move from the general asynchronous model to the more restrictive *rsync* setting, we no longer have a complete understanding of which queries can be computed without coordination, and which ones, instead, do require coordination. It turns out that both the communication model and the definition of coordination proposed in [4] are not strong enough to work in general for synchronous systems. As the reader may have realized, this is due to the fact that, in broadcasting synchronous systems, coordination – as defined by Ameloot et al. – is already "baked" into the model. In the next sections we will see that our definition of coordination-freedom guarantees eventually consistent computation for those queries that do not rely on broadcasting in order to progress. That is, the discriminating condition for eventual consistency is not monotonicity, but the fact that it *is not necessary* to send a fact to all the nodes composing a network.

3.2 Hashing Transducer Networks

Broadcasting specifications are not really convenient from a practical perspective. Following other parallel programming models such as MapReduce, in this section we are going to introduce *hashing transducer networks*: *i.e.,* synchronous networks of relational transducers equipped with a content-based communication model founded on hashing. Under this new model, the node to which an emitted fact must be addressed is derived using a hash function applied to a subset of its terms called *keys*.

Example 4. This program is the hashed version of Example 2, where every tuple emitted over S and U is hashed on the first term (this is specified by the schema definition

$S^{(1,2)}$ and $U^{(1,2)}$, where the pair $(1, 2)$ means that the related relation has arity 2 and the first term is the key-term). In this way we are assured that, for each pair of joining tuples, at least a node exists containing the pair. This because S and U are joined over their key-terms, and hence the joining tuples are addressed to the same node.

Schema: $\Upsilon_{db} = \{R^{(2)}, T^{(2)}\}, \Upsilon_{com} = \{S^{(1,2)}, U^{(1,2)}\}, \Upsilon_{out} = \{J^{(3)}\}$
Program: $S_{snd}(u, v) \leftarrow R(u, v).$
$U_{snd}(u, v) \leftarrow T(u, v).$
$J_{out}(u, v, w) \leftarrow S(u, v), U(u, w).$

---

4 Coordination-freedom Refined

We have seen in Section 3.1 that, for *rsynch* systems, a particular notion of coordination-freedom is needed. In fact we have shown that, under such model, certain non-monotonic queries – Example 3 – requiring coordination under the asynchronous model can be computed in a coordination-free way. The key-point is that, as observed in [4], in asynchronous systems coordination-freedom is directly related to communication-freedom under ideal partitioning. That is, if the partitioning is correct, no communication is required to correctly compute a coordination-free query because (i) no data must be sent (the partition is correct), and (ii) no "control message" is required to obtain a consistent result (the query is coordination-free). However, due to its synchronous nature, in *rsync* settings non-monotonic queries can be computed in general without resorting to coordination because coordination is already "baked" into the *rsync* model: each node is synchronized with every other one, hence "control messages" are somehow implicitly assumed. In this section we introduce a novel knowledge-oriented perspective linking coordination with the way in which explicit and implicit information flows in the network. Under this perspective, we will see that coordination is needed if, to maintain consistency, a node must have some form of information exchange with all the other nodes.

4.1 Syncausality

Achieving coordination in asynchronous systems is a costly task. A necessary condition for coordination in such systems is the existence of primitives that enforce some control over the ordering of events. In a seminal paper [13], Lamport proposed a synchronization algorithm based on the relation of *potential causality* ($\rightarrow$) over asynchronous events. According to Lamport, given two events e, e', we have that $e \rightarrow e'$ if e happens before e' and e might have caused e'. From a high-level perspective, the potential causality relation models how information flows among processes, and therefore can be employed as a tool to reason on the patterns which cause coordination in asynchronous systems. A question now arises: what is the counterpart of the potential causality relation for synchronous systems? *Synchronous potential causality* (*syncausality* in short) has been recently proposed [5] to generalize Lamport's potential causality to synchronous systems. Using syncausality we are able to model how information flows among nodes with the passing of time. Consider a parallel execution trace ρ – called a *run* – and two *points* in this execution (ρ^i, t), (ρ^j, t') for (possibly not distinct) nodes i, j, identifying the local state for i, j at time t and t' respectively. We say that (ρ^j, t') *causally depends* on (ρ^i, t) if either $i = j$ and $t \leq t'$ – *i.e.,* a local state depends on the previous one – or a tuple has been emitted by node i at time t, addressed to node j, with $t < t'$[1]. We refer to these two types of dependencies as *direct*.

Definition 4.1. *Given a run ρ, we say that two points (ρ^i, t), (ρ^j, t') are related by a* direct potential causality *relation $\rightarrow$, if one of the following is true:*

1. $t' = t + 1$ *and* $i = j$*;*
2. $t' \geq t + 1$ *and node i sent a tuple at time t addressed to j;*
3. *there is a point (ρ^k, t'') s.t. $(\rho^i, t) \rightarrow (\rho^k, t'')$ and $(\rho^k, t'') \rightarrow (\rho^j, t')$.*

Note that direct dependencies define precisely Lamport's happen-before relation – and hence we maintain the same signature $\rightarrow$.

Differently from asynchronous systems, we however have that a point on node j can occasionally *indirectly* depend on another point on node i even if no fact addressed to j is actually sent by i. This is because j can still draw some conclusion simply as a consequence of the *bounded delay guarantee* of synchronous systems. That is, each node can use the common knowledge that every sent tuple is received at most after a certain bounded delay to reason about the state of the system. The bounded delay guarantee can be modelled as an imaginary $NULL$ fact, like in [14]. Under this perspective, indirect dependencies appear the same as the direct ones, although, instead of a flow generated by "informative" facts, with the indirect relationship we model the flow of "non-informative", $NULL$ facts.

Definition 4.2. *Given a run ρ, we say that two points (ρ^i, t), (ρ^j, t') are related by an* indirect potential causality *relation $\dashrightarrow$, if $i \neq j$, $t' \geq t + 1$ and a $NULL^i_R$ fact addressed to node j has been (virtually) sent by node i at round t.*

An interesting fact about the bounded delay guarantee is that it can be employed to specify when negation can be safely applied to a predicate. In general, negation can be applied to a literal $R(\bar{u})$ when the content of R is sealed for what concerns the current round. In local settings, we have that such condition holds for a predicate at round t' if its content has been completely generated at round t, with $t' > t$. In distributed settings, we have that if R is a $communication$ relation, being in a new round t' is not enough, in general, for establishing that its content is sealed. This is because tuples can still be floating, and therefore, until we are assured that every tuple has been delivered, the above condition does not hold. The result is that negation cannot be applied safely. We can reason in the same way also for every other negative literal depending on R. We will then model the fact that the content of a $communication$ relation R is stable because of the bounded delay guarantee, by having every node i emit a fact $NULL^i_R$ at round t, for every $communication$ relation R, which will be delivered at node j exactly by the

[1] Note that a point in a synchronous system is what Lamport defines as an event in an asynchronous system.

next round. We then have that the content of R is stable once j has received a $NULL^i_R$ fact from every node i contained in the set N of nodes composing the network. The sealing of a *communication* relation at a certain round is then ascertained only when $|N|$ $NULL_R$ facts have been counted. Recall that not necessarily the $NULL^i_R$ facts must be physically sent. This in particular is true under our *rsync* model, where the strike of a new round automatically seals all the *communication* relations. Example 5 shows one situation in which this applies.

Example 5. Consider the hashing version of the program of Example 3. Let **I** be an initial instance. At round $t+1$ we have that the relation S is stable, and hence negation can be applied. Note that if R is empty in the initial instance, no fact is sent. Despite this, every node can still conclude at round $t+1$ that the content of S is stable. In this situation we clearly have an indirect potential causality relation.

We are now able to introduce the definition of syncausality: a generalization of Lamport's happen-before relation which considers not only the direct information flow, but also the flow generated by indirect dependencies.

Definition 4.3. *Let ρ be a run. The* syncausality *relation $\rightsquigarrow$ is the smallest relation s.t.:*

1. *if $(\rho^i, t) \rightarrow (\rho^j, t')$, then $(\rho^i, t) \rightsquigarrow (\rho^j, t')$;*
2. *if $(\rho^i, t) \dashrightarrow (\rho^j, t')$, then $(\rho^i, t) \rightsquigarrow (\rho^j, t')$; and*
3. *if $(\rho^i, t) \rightsquigarrow (\rho^j, t')$ and $(\rho^j, t') \rightsquigarrow (\rho^k, t'')$, then $(\rho^i, t) \rightsquigarrow (\rho^k, t'')$.*

4.2 From Syncausality to Coordination

We next propose the *predicate-level syncausality* relationship, modeling causal relations at the predicate level. That is, instead of considering how (direct and indirect) information flows between nodes, we introduce a more fine-grained relationship modelling the flows between *predicates and nodes*.

Definition 4.4. *Given a run ρ, we say that two points (ρ^i, t), (ρ^j, t') are linked by a relation of* predicate-level syncausality $\overset{R}{\rightsquigarrow}$*, if any of the following holds:*

1. *$i = j$, $t' = t+1$ and a tuple over $R \in \Upsilon_{mem} \cup \Upsilon_{out}$ has been derived by a query in $Q_{ins} \cup Q_{out}$ at time t';*
2. *$R \in \Upsilon_{com}$ and node i sends a tuple over R at time t addressed to node j, with $t' \geq t+1$;*
3. *$R \in \Upsilon_{com}$ and node i (virtually) sends a $NULL^i_R$ fact at time t addressed to node j, with $t' \geq t+1$;*
4. *there is a point (ρ^k, t'') s.t. $(\rho^i, t) \overset{R}{\rightsquigarrow} (\rho^k, t'')$ and $(\rho^k, t'') \overset{R}{\rightsquigarrow} (\rho^j, t')$.*

We are now able to specify a condition for achieving coordination. Informally, we have that coordination exists when all the nodes of a network reach a common agreement that some event happened. But the only way to reach such an agreement is that a (direct or indirect) information flow exists between the node in which the event actually occurs, and every other node. This is a sufficient and necessary condition because of the reliability and bounded-delay guarantee of *rsync* systems. Formalizing this intuition by means of the (predicate level) syncausality relationship we have that:

Definition 4.5. *Let N be a set of nodes. We say that a synchronous relational transducer network manifests the* coordination pattern *if, for all possible initial instances $\boldsymbol{I} \in inst(\Upsilon_{db})$, whichever run we select, a point (ρ^i, t) and a communication relation R exist so that $\forall j \in N$ there is a predicate-level syncausality relation such that $(\rho^i, t) \overset{R}{\rightsquigarrow} (\rho^j, t')$.*

We call node i the *coordination master*. A pattern with a similar role has been named *broom* in [6].

Remark: The reader can now appreciate to which extent coordination was already "baked" inside the broadcasting synchronous specifications of Section 3. Note that broadcasting, in *rsync*, brings coordination. This is not true in asynchronous systems.

Intuitively, the coordination master is where the event occurs. If a broadcasting of (informative or non-informative) fact occurs, then such event will become *common knowledge* [8] among the nodes. On the contrary, if broadcasting is not occurring, common knowledge cannot be obtained and therefore, if the correct final outcome is still reached, this is obtained without coordination. That is, if at least a non-trivial configuration exists s.t. the coordination pattern doesn't manifest itself, we have coordination-freedom.

5 CALM in rsync Systems

The original version of the CALM principle is not satisfiable in *rsync* systems because a monotonic class of queries exists—*i.e., unchained queries*, introduced next—which is not coordination-free. Informally, a query is chained if every relation is connected through a join-path with every other relation composing the same query.

Definition 5.6. *Let body(q_R) be a conjunction of literals defining the body of a query q_R. We say that two different positive litteral occurrences $R_i(\bar{u}_i)$, $R_j(\bar{u}_j) \in$ body(q_R) are* chained *in q_R if either:*

- *$\bar{u}_i \cap \bar{u}_j \neq \emptyset$; or*
- *a third relation $R_k \in q_R$ different from R_i, R_j exists such that R_i is chained with R_k, and R_k is chained with R_j.*

Definition 5.7. *A query $\mathcal{Q}_{out}$ is said* chained *if, for every rule $q_R \in \mathcal{Q}_{out}$, each relation occurrence $R_i \in$ body(q_R) is chained with every other relation occurrence $R_j \in$ body(q_R).*

Remark: Nullary relations are not chained by definition.

Example 6. Assume two relations $R^{(2)}$ and $T^{(1)}$, and the following query $\mathcal{Q}_{out}$ returning the full R-instance if T is nonempty.

$$Q(u, v) \leftarrow R(u, v), T(\_).$$

The query is clearly monotonic. Let $\mathcal{T}$ be the following broadcasting UCQ-transducer program computing $\mathcal{Q}_{out}$.

Schema: $\Upsilon_{db} = \{R^{(2)}, T^{(1)}\}, \Upsilon_{com} = \{S^{(2)}, U^{(1)}\}, \Upsilon_{out} = \{Q^{(2)}\}$

Program: $S_{snd}(u, v) \leftarrow R(u, v).$

$U_{snd}(u) \leftarrow T(u).$

$Q_{out}(u, v) \leftarrow S(u, v), U(\_).$

Assume now we want to make the above transducer a hashing one. We have that, whichever key we chose, the related specification might be no more consistent. Indeed, consider an initial instance **I** and a set of keys spanning all the terms of S and U. Assume **I** such that $adom(I_R) \supset adom(I_T)$, and a network composed by a large number of nodes. In this situation, it may happen that a nonempty set of facts over R is hashed to a certain node i, while no fact over T is hashed to i. This because a constant may exist in $adom(I_R)$ that is not in $adom(I_T)$ and for which the hashing function returns a node i not returned by hashing any constant in $adom(I_T)$. Hence no tuple emitted to i will ever appear in the output, although they do appear in $\mathcal{Q}_{out}(\mathbf{I})$. Thus this transducer is not eventually consistent.

From the above example we can intuitively see that, for *rsync*, a final consistent result can be obtained without coordination only for queries that are chained and monotonic. That is, the following restricted version of the CALM conjecture holds for *rsync* systems:

Theorem 1 *A query can be parallelly computed by a coordination-free transducer network if it is chained and monotonic [11].*

We will leave for future works the investigation on whether every monotone and chained query is also coordination-free.

Remark: For the readers familiar with the works [2, 3] our result state that under the *rsync* model, a query is computable in a coordination-free way if monotonic and distributing over components.

6 Conclusions

In this paper the CALM principle is analyzed under synchronous and reliable settings. By exploiting CALM, in fact, we would be able to break the synchronous cage of modern parallel computation models, and provide pipelined coordination-free executions when allowed by the program logic. In order to reach our goal, we have introduced a new abstract model emulating BSP computation, and a novel interpretation of coordination with sound logical foundations in distributed knowledge reasoning. By exploiting such techniques, we have shown that the if direction of the CALM principle indeed holds also in *rsync* settings, but just for the subclass of monotonic queries defined as chained.

REFERENCES

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] T. J. Ameloot, B. Ketsman, F. Neven, and D. Zinn. Weaker forms of monotonicity for declarative networking: a more fine-grained answer to the calm-conjecture. In *PODS*, pages 64–75. ACM, 2014.

[3] T. J. Ameloot, B. Ketsman, F. Neven, and D. Zinn. Datalog queries distributing over components. In *ICDT*. ACM, 2015.

[4] T. J. Ameloot, F. Neven, and J. Van Den Bussche. Relational transducers for declarative networking. *J. ACM*, 60(2):15:1–15:38, May 2013.

[5] I. Ben-Zvi and Y. Moses. Beyond lamport's *happened-before*: On the role of time bounds in synchronous systems. In N. A. Lynch and A. A. Shvartsman, editors, *DISC*, volume 6343 of *Lecture Notes in Computer Science*, pages 421–436. Springer, 2010.

[6] I. Ben-Zvi and Y. Moses. On interactive knowledge with bounded communication. *Journal of Applied Non-Classical Logics*, 21(3-4):323–354, 2011.

[7] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. Mapreduce online. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[8] R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning About Knowledge*. MIT Press, Cambridge, MA, USA, 2003.

[9] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Rec.*, 39:5–19, September 2010.

[10] M. Interlandi. Reasoning about knowledge in distributed systems using datalog. In *Datalog*, pages 99–110, 2012.

[11] M. Interlandi and L. Tanca. On the calm principle for bulk synchronous parallel computation. `arXiv:1405.7264`.

[12] M. Interlandi, L. Tanca, and S. Bergamaschi. Datalog in time and space, synchronously. In L. Bravo and M. Lenzerini, editors, *AMW*, volume 1087 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2013.

[13] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[14] L. Lamport. Using time instead of timeout for fault-tolerant distributed systems. *ACM Trans. Program. Lang. Syst.*, 6(2):254–280, Apr. 1984.

[15] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 1099–1110, New York, NY, USA, 2008. ACM.

[16] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy. Hive: A warehousing solution over a map-reduce framework. *Proc. VLDB Endow.*, 2(2):1626–1629, Aug. 2009.

[17] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, Aug. 1990.

[18] W. Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, Jan. 2009.

Chase Termination for Guarded Existential Rules

Marco Calautti[1], Georg Gottlob[2], and Andreas Pieris[3]

[1] DIMES, University of Calabria, Italy calautti@dimes.unical.it
[2] Department of Computer Science, University of Oxford, UK
georg.gottlob@cs.ox.ac.uk
[3] Institute of Information Systems, Vienna University of Technology, Austria
pieris@dbai.tuwien.ac.at

Abstract. The chase procedure is considered as one of the most fundamental algorithmic tools in database theory. It has been successfully applied to different database problems such as data exchange, and query answering and containment under constraints, to name a few. One of the central problems regarding the chase procedure is all-instance termination, that is, given a set of tuple-generating dependencies (TGDs) (a.k.a. existential rules), decide whether the chase under that set terminates, for every input database. It is well-known that this problem is undecidable, no matter which version of the chase we consider. The crucial question that comes up is whether existing restricted classes of TGDs, proposed in different contexts such as ontological reasoning, make the above problem decidable. In this work, we focus our attention on the oblivious and the semi-oblivious versions of the chase procedure, and we give a positive answer for classes of TGDs that are based on the notion of guardedness.

1 Introduction

The *chase procedure* (or simply chase) is considered as one of the most fundamental algorithmic tools in databases — it accepts as input a database D and a set Σ of constraints and, if it terminates (which is not guaranteed), its result is a finite instance D_Σ that enjoys two crucial properties:

1. D_Σ is a *model* of D and Σ, i.e., it contains D and satisfies the constraints of Σ; and
2. D_Σ is *universal*, i.e., it can be homomorphically embedded into every other model of D and Σ.

In other words, the chase is an algorithmic tool for computing universal models of D and Σ, which can be conceived as representatives of all the other models of D and Σ. This is precisely the reason for the ubiquity of the chase in database theory. Indeed, many key database problems can be solved by simply exhibiting a universal model.

A central class of constraints, which can be treated by the chase procedure and is of special interest for this work, are the well-known *tuple-generating dependencies (TGDs)* (a.k.a. *existential rules*) of the form $\forall \mathbf{X} \forall \mathbf{Y} (\varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z} (\psi(\mathbf{Y}, \mathbf{Z})))$, where φ and ψ are conjunctions of atoms. Given a database D and a set Σ of TGDs, the chase adds new atoms to D (possibly involving nulls) until the final result satisfies Σ.

Example 1. Consider the database $D = \{person(Bob)\}$, and the TGD

$$\forall X(person(X) \rightarrow \exists Y\ hasFather(X, Y) \wedge person(Y)),$$

which asserts that each person has a father who is also a person. The database atom *triggers* the TGD, and the chase will add in D the atoms $hasFather(Bob, z_1)$ and $person(z_1)$ in order to satisfy it, where z_1 is a (labeled) null representing some unknown value. However, the new atom $person(z_1)$ triggers again the TGD, and the chase is forced to add the atoms $hasFather(z_1, z_2), person(z_2)$, where z_2 is a new null. The result of the chase is the instance

$$\{person(Bob), hasFather(Bob, z_1)\} \ \cup \ \bigcup_{i>0}\{person(z_i), hasFather(z_i, z_{i+1})\},$$

where $z_1, z_2, \ldots$ are nulls. ■

As shown by the above example, the chase procedure may run forever, even for extremely simple databases and constraints. In the light of this fact, there has been a long line of research on identifying syntactic properties on TGDs such that, for every input database, the termination of the chase is guaranteed; see, e.g., [4, 8, 10, 12, 13] — this list is by no means exhaustive, and we refer to [9] for a comprehensive survey. With so much effort spent on identifying sufficient conditions for the termination of the chase procedure, the question that comes up is whether a sufficient condition that is also *necessary* exists. In other words, given a set Σ of TGDs, is it possible to determine whether, for every database D, the chase on D and Σ terminates? This interesting question has been recently addressed in [6], and unfortunately the answer is negative for all the versions of the chase that are usually used in database applications, namely the oblivious, semi-oblivious and restricted chase. In fact, the problem remains undecidable even if the database is known. This has been established in [4] for the restricted chase, and it was observed in [12] that the same proof shows undecidability also for the oblivious and the semi-oblivious chase.

Although the chase termination problem is undecidable in general, the proof given in [6] does not show the undecidability of the problem for TGDs that enjoy some structural conditions, which in turn guarantee favorable model-theoretic properties. Such a key condition is *guardedness*, a well-accepted paradigm that gives rise to robust rule-based languages that capture important databases constraints and lightweight description logics. A TGD is guarded if it has an atom in the left-hand side that contains (or guards) all the universally quantified variables [2]. Guardedness guarantees the tree-likeness of the underlying models, and thus the decidability of central database problems. The question that comes up is whether guardedness has the same positive impact on chase termination.

We focus on the (semi-)oblivious versions of the chase, and we show that the problem of deciding the termination of the chase for guarded TGDs is decidable, and we establish precise complexity results. Surprisingly, the present work is to our knowledge the first one that establishes positive results for the (semi-)oblivious chase termination problem. For more details, we refer the reader to [1].

2 The Chase Termination Problem

The *TGD chase procedure* (or simply *chase*) takes as input an instance I and a set Σ of TGDs, and constructs a universal model of I and Σ. The chase works on I by applying the so-called *trigger* for a set of TGDs on I. The trigger for a set Σ of TGDs on an instance I is a pair (σ, h), where $\sigma = \varphi \to \psi \in \Sigma$ and h is a homomorphism that maps φ to I. An *application* of (σ, h) to I returns $J = I \cup h'(\psi)$, where $h' \supseteq h$ maps each existentially quantified variable in ψ to a new null value. Such a trigger application is written $I\langle\sigma, h\rangle J$. The choice of the type of the next trigger to be applied is crucial since it gives rise to different versions of the chase procedure. In this work, we focus our attention on the *oblivious* [2] and *semi-oblivious* [7, 12] chase.

A finite sequence $I_0, I_1, \ldots, I_n$, where $n \geqslant 0$, is said to be a *terminating oblivious chase sequence* of I_0 w.r.t. a set Σ of TGDs if: (i) for each $0 \leqslant i < n$, there exists a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle I_{i+1}$; (ii) for each $0 \leqslant i < j < n$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j = \sigma$ implies $h_i \neq h_j$, i.e., h_i and h_j are different homomorphisms; and (iii) there is no trigger (σ, h) for Σ on I_n such that $(\sigma, h) \notin \{(\sigma_i, h_i)\}_{0 \leqslant i \leqslant n-1}$. In this case, the result of the chase is the (finite) instance I_n. An infinite sequence $I_0, I_1, \ldots$ of instances is said to be a *non-terminating oblivious chase sequence* of I_0 w.r.t. Σ if: (i) for each $i \geqslant 0$, there exists a trigger (σ, h) for Σ on I_i such that $I_i\langle\sigma, h\rangle I_{i+1}$; (ii) for each $i, j > 0$ such that $i \neq j$, assuming that $I_i\langle\sigma_i, h_i\rangle I_{i+1}$ and $I_j\langle\sigma_j, h_j\rangle I_{j+1}$, $\sigma_i = \sigma_j = \sigma$ implies $h_i \neq h_j$; and (iii) for each $i \geqslant 0$, and for every trigger (σ, h) for Σ on I_i, there exists $j \geqslant i$ such that $I_j\langle\sigma, h\rangle I_{j+1}$; this is known as the fairness condition, and guarantees that all the triggers eventually will be applied. The result of the chase is defined as the infinite instance $\cup_{i \geqslant 0} I_i$.

The semi-oblivious chase is a refined version of the oblivious chase, which avoids the application of some superfluous triggers. Roughly speaking, given a TGD σ of the form $\varphi \to \psi$, for the semi-oblivious chase, two homomorphisms h and g that agree on the universally quantified variables of σ occurring in ψ are indistinguishable.

Henceforth, we write o-chase and so-chase for oblivious and semi-oblivious chase, respectively. A $\star$-chase sequence, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, may be infinite.

Example 2. Let $D = \{p(a, b)\}$, and $\Sigma = \{\forall X \forall Y (p(X, Y) \to \exists Z (p(Y, Z)))\}$. There exists only one $\star$-chase sequence of D w.r.t. Σ, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, which is non-terminating, i.e., $I_0, I_1, \ldots$ with

$$I_0 = \{p(a, b)\} \quad I_1 = \{p(a, b), p(b, z_1)\} \quad I_i = I_{i-1} \cup \{p(z_{i-1}, z_i)\}, \text{ for } i \geqslant 2,$$

where $z_1, z_2, \ldots$ are nulls of $\mathbf{N}$. ■

For a set of TGDs, a key question is whether all or some $\star$-chase sequences are terminating on *all* databases. Before formalizing the above decision problems, let us recall the following key classes of TGDs:

$$\mathsf{CT}^{\star}_{\forall} = \{\Sigma \mid \forall D, \text{ all } \star\text{-chase sequences of } D \text{ w.r.t. } \Sigma \text{ are terminating}\}$$
$$\mathsf{CT}^{\star}_{\exists} = \{\Sigma \mid \forall D, \text{ there exists a terminating } \star\text{-chase sequence of } D \text{ w.r.t. } \Sigma\}.$$

The decision problems tackled in this work are as follows: for $q \in \{\forall, \exists\}$:

q-**SEQUENCE** $\star$-**CHASE TERMINATION:**
Instance: A set Σ of TGDs.
Question: Does $\Sigma \in \mathsf{CT}^{\star}_{q}$?

We recall that $\mathsf{CT}^{\mathsf{o}}_{\forall} = \mathsf{CT}^{\mathsf{o}}_{\exists} \subset \mathsf{CT}^{\mathsf{so}}_{\forall} = \mathsf{CT}^{\mathsf{so}}_{\exists}$ [7]. This implies that the preceding decision problems coincide for the (semi-)oblivious chase. Henceforth, we refer to the $\star$-chase termination problem, and we write $\mathsf{CT}^{\star}$ for $\mathsf{CT}^{\star}_{\forall}$ and $\mathsf{CT}^{\star}_{\exists}$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$.

3 The Complexity of Chase Termination

We focus on the class of guarded TGDs [2], and two key subclasses of it, namely simple linear and linear TGDs [3], and we investigate the complexity of the (semi-)oblivious chase termination problem. Recall that linear TGDs are TGDs with just one atom in the body, while simple linear TGDs forbid the repetition of variables in the body. Notice that, despite their simplicity, simple linear TGDs are powerful enough for capturing prominent database dependencies, and in particular inclusion dependencies, as well as key description logics such as DL-Lite. In the sequel, we denote by G the class of guarded TGDs, which is defined as the family of all possible sets of guarded TGDs. Analogously, we denote by SL and L the classes of simple linear and linear TGDs, respectively; clearly, $\mathsf{SL} \subset \mathsf{L} \subset \mathsf{G}$. Let us first consider the less expressive classes.

3.1 Linearity

By exploiting syntactic conditions that ensure the termination of each (semi-)oblivious chase sequence on all databases, we syntactically characterize the classes $(\mathsf{CT}^{\star} \cap \mathsf{SL})$ and $(\mathsf{CT}^{\star} \cap \mathsf{L})$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$. We rely on weak-acyclicity [5] and rich-acyclicity [11]. Both weak- and rich-acyclicity are defined by posing an acyclicity condition on a graph, which encodes how terms are propagated among the positions of the underlying schema during the chase. In fact, weak-acyclicity forbids the existence of dangerous cycles (which involve the generation of new null values) in the dependency graph [5], while rich-acyclicity pose the same condition on the so-called extended dependency graph [11]. Let WA and RA be the classes of weakly- and richly-acyclic TGDs, respectively; notice that $\mathsf{RA} \subset \mathsf{WA}$. For simple linear TGDs we show that:

Theorem 1. $(\mathsf{CT}^{\mathsf{o}} \cap \mathsf{SL}) = (\mathsf{RA} \cap \mathsf{SL})$ *and* $(\mathsf{CT}^{\mathsf{so}} \cap \mathsf{SL}) = (\mathsf{WA} \cap \mathsf{SL})$.

In simple words, the above theorem states that, given a set $\Sigma \in \mathsf{SL}$: $\Sigma \in \mathsf{CT}^{\mathsf{o}}$ iff Σ is richly-acyclic, and $\Sigma \in \mathsf{CT}^{\mathsf{so}}$ iff Σ is weakly-acyclic. This result is established by showing that a dangerous cycle in the extended dependency graph (resp., dependency graph) necessarily gives rise to a non-terminating o-chase (resp., so-chase) sequence.

Let us now focus on (non-simple) linear TGDs. It is possible to show, by exhibiting a counterexample, that a dangerous cycle does not necessarily correspond to an infinite chase derivation. Thus, rich- and weak-acyclicity are not powerful enough for syntactically characterize the fragment of linear TGDs that guarantees the termination of the oblivious and semi-oblivious chase, respectively. Interestingly, it is possible to extend rich- and weak-acyclicity, focussing on linear TGDs, in such a way that the above key property holds. The obtained formalisms are dubbed *critical-rich-acyclicity* and *critical-weak-acyclicity*, and the corresponding classes are denoted as $\mathsf{LCriticalRA}$ and $\mathsf{LCriticalWA}$, respectively. We show that:

Theorem 2. $(\mathsf{CT}^{\mathsf{o}} \cap \mathsf{L}) = \mathsf{LCriticalRA}$ *and* $(\mathsf{CT}^{\mathsf{so}} \cap \mathsf{L}) = \mathsf{LCriticalWA}$.

The above syntactic characterizations, apart from being interesting in their own right, allow us to obtain optimal upper bounds for the $\star$-chase termination problem for $(\mathsf{S})\mathsf{L}$ — we simply need to analyze the complexity of deciding whether a set of (simple) linear TGDs enjoys the above acyclicity-based conditions, which can be formulated as a reachability problem on a graph. In particular, we obtain the following results:

Theorem 3. *Consider a set Σ of TGDs. The problem of deciding whether $\Sigma \in \mathsf{CT}^{\star}$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, is*

1. NL-*complete, even for unary and binary predicates, if $\Sigma \in \mathsf{SL}$; and*
2. PSPACE-*complete, and* NL-*complete for predicates of bounded arity, if $\Sigma \in \mathsf{L}$.*

For the hardness results, a generic technique, called the *looping operator*, is proposed, which allows us to obtain lower bounds for the chase termination problem in a uniform way. In fact, the goal of the looping operator is to provide a generic reduction from propositional atom entailment to the complement of chase termination.

3.2 Guardedness

We proceed to investigate the (semi-)oblivious chase termination problem for guarded TGDs. Although there is no way (at least no obvious one) to syntactically characterize the classes $(\mathsf{CT}^{\star} \cap \mathsf{G})$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, via rich- and weak-acyclicity, as we did for (simple) linear TGDs, it is possible to show that the problem of recognizing the above classes is decidable. For technical reasons, we focus on *standard databases*, that is, databases that have two constants, let say 0 and 1, that are available via the unary predicates $0(\cdot)$ and $1(\cdot)$, respectively. In particular, we show the following:

Theorem 4. *Consider a set $\Sigma \in \mathsf{G}$. The problem of deciding whether $\Sigma \in \mathsf{CT}^{\star}$, where $\star \in \{\mathsf{o}, \mathsf{so}\}$, focussing on standard databases, is* 2EXPTIME-*complete, and* EXPTIME-*complete for predicates of bounded arity.*

The upper bounds are obtained by exhibiting an alternating algorithm that runs in exponential space, in general, and in polynomial space in case of predicates of bounded arity. The lower bounds are obtained by reductions from the acceptance problem of alternating exponential (resp., polynomial) space clocked Turing machines, i.e., Turing machines equipped with a counter. These reductions are obtained by modifying significantly existing reductions for the problem of propositional atom entailment under guarded TGDs, and then exploiting the looping operator mentioned above. The fact that the database is standard, is crucial for establishing the above lower bounds; the upper bounds hold even for non-standard databases.

4 Future Work

Our next step is to perform similar analysis focussing on the restricted version of the chase. We already have some preliminary positive results. In particular, if we focus on single-head linear TGDs, where each predicate appears in the head of at most one TGD, then we can syntactically characterize, via a careful extension of weak-acyclicity, the fragment that guarantees the termination of the restricted chase, and obtain a polynomial time upper bound. We are currently working towards the full settlement of the problem.

Acknowledgements. M. Calautti was supported by the European Commission, European Social Fund and Region Calabria. G. Gottlob was supported by the EPSRC Programme Grant EP/M025268/ "VADA: Value Added Data Systems – Principles and Architecture", and the Grant ERC-POC-2014 Nr. 641222 "ExtraLytics: Big Data for Real Estate". A. Pieris was supported by the Austrian Science Fund (FWF), projects P25207-N23 and Y698, and Vienna Science and Technology Fund (WWTF), project ICT12-015.

References

1. Calautti, M., Gottlob, G., Pieris, A.: Chase termination for guarded existential rules. In: PODS (2015), to appear
2. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. J. Artif. Intell. Res. 48, 115–174 (2013)
3. Calì, A., Gottlob, G., Lukasiewicz, T.: A general Datalog-based framework for tractable query answering over ontologies. J. Web Sem. 14, 57–83 (2012)
4. Deutsch, A., Nash, A., Remmel, J.B.: The chase revisisted. In: PODS. pp. 149–158 (2008)
5. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: Semantics and query answering. Theor. Comput. Sci. 336(1), 89–124 (2005)
6. Gogacz, T., Marcinkowski, J.: All-instances termination of chase is undecidable. In: ICALP. pp. 293–304 (2014)
7. Grahne, G., Onet, A.: Anatomy of the chase. CoRR abs/1303.6682 (2013), http://arxiv.org/abs/1303.6682
8. Grau, B.C., Horrocks, I., Krötzsch, M., Kupke, C., Magka, D., Motik, B., Wang, Z.: Acyclicity conditions and their application to query answering in description logics. In: KR (2012)
9. Greco, S., Molinaro, C., Spezzano, F.: Incomplete Data and Data Dependencies in Relational Databases. Morgan & Claypool Publishers (2012)
10. Greco, S., Spezzano, F., Trubitsyna, I.: Stratification criteria and rewriting techniques for checking chase termination. PVLDB 4(11), 1158–1168 (2011)
11. Hernich, A., Schweikardt, N.: CWA-solutions for data exchange settings with target dependencies. In: PODS. pp. 113–122 (2007)
12. Marnette, B.: Generalized schema-mappings: From termination to tractability. In: PODS. pp. 13–22 (2009)
13. Meier, M., Schmidt, M., Lausen, G.: On chase termination beyond stratification. PVLDB 2(1), 970–981 (2009)

Data integration with many heterogeneous sources and dynamic target schemas (extended abstract)

Luigi Bellomarini, Paolo Atzeni, Luca Cabibbo

Università Roma Tre, Italy

1 Introduction and motivation

Information integration is the general problem that arises in applications that need to consolidate (in a virtual or materialized way) data coming from different sources [9, Ch.21]. In this paper, we consider a scenario for data integration, very common in practice, for which existing solutions are not effective. We refer to those applications where there are many (dozens or hundreds or even more) sources, in the same domain, that have to contribute to one, single global ("target") system. A common case is that of "central" organizations that receive data from a large set of "local" companies or administrations; a specific case is that of a national central bank that receives data from all the banks in the country. This scenario is often handled by imposing to all local sources an exchange format, so that data are transferred to the central institution in a standardized form. In some cases, this solution is just inapplicable, as companies might refuse the adoption of the standard unless forced by regulations. Moreover, exchange formats (especially for statistics and finance) are inherently flexible, versatile and unstable, allowing a number of different source schemas. Finally, due to the complexity of exchange formats, their adoption is often partial or incorrect. In particular, a diverging interpretation of the exchange format can be even accepted in case the central institution realizes that the various sources have specific features that, though not general, are nevertheless interesting and deserve to appear in the target.

An approach based on the theory of mappings [6] can be interesting in this case, but the large quantity of source schemas would render the classical techniques not much effective, as it would require the specification of many different mappings, between each of the various sources and the target. Also, the sources are not completely known beforehand (and there are often new ones, also similar, but with additional differences) and therefore mappings cannot be directly specified in advance. Finally, given the interest in considering specific features of sources, it turns out that even the target schema should not be fixed in advance, as some portions depend on sources (their schema and even their data). All these three aspects are not addressed by current mapping-based approaches.

We also have an observation that mitigates the difficulties: in many cases the sources are indeed different from one another, but they do share similarities that can be exploited.

On the basis of the above requirements we propose a new approach, where there is one source schema S_0, used as a reference, and the other source schemas can be seen as variations of S_0. Then, we consider variations on the target schema, induced by specific features of the source schemas, including their data, in order to support "schematic transformations" [11], that is, the possibility to generate schema elements in the final target schema G, given data and schema elements in S_0.

G

RepositoryOfBalances

Bank	*IoBName*	*Amount*
1005	Asset	8
1005	Liability	19
1006	Asset	-42
1006	Liability	21

G'

RepositoryOfBalancesAlt

Bank	*Asset*	*Liability*
1005	8	19
1006	-42	21

Fig. 1. Sample target data

For example, Figure 1 represents two desired global schemas, each of which is able to store summaries of balance sheets of banks. The relation RepositoriesOfBalances stores the *Amount* (that is, the difference between credits and debits) aggregated by *Bank* and *IoBName* (item of balance name). RepositoriesOfBalancesAlt ("Alt" stands for "alternate") stores the same items for each bank with the difference that *Asset* and *Liabilities* are attributes.

Figure 2 represents the data as they are collected from the sources. S_0 is the established exchange format, where BalSheetTemplate prescribes the local institution to structure its balance in relations (one or many) having the attributes *Year*, *Bank*, *IoBName*, *IoBCred*, the amount of credit for the item of balance, *IoBDeb*, the amount of debit for the item of balance.

In the example, S_1 is identical to S_0 and reports the balance sheet as a single relation. In S_2 the balance sheet is (horizontally) partitioned into two relations, with the same attributes as S_0: tuples having *Year* preceding 2012 are stored in BalSheet2010-2011, the others in BalSheet2013. In S_3, the balance is (vertically) decomposed into credits and debits, with a coarser level of granularity, as data are reported and grouped by *Bank*, disregarding the years.

In the rest of the paper we study the data integration scenario we have just sketched. In Section 2 we show how we handle a large number of heterogeneous but similar sources and in Section 3 we discuss how we cope with the issue of generating new features in target schemas on the basis of source schemas and data. Finally, in Section 4 we draw our conclusions and briefly discuss related work.

S_0 BalSheetTemplate

Year	*Bank*	*IoBName*	*IoBCred*	*IoBDeb*

S_1 BalSheet

Year	*Bank*	*IoBName*	*IoBCred*	*IoBDeb*
2010	1005	Asset	35	27
2011	1005	Liability	29	10
2010	1006	Asset	41	30
2010	1006	Liability	31	30
2013	1006	Asset	0	53
2013	1006	Liability	33	13

S_2 BalSheet2010-2011

Year	*Bank*	*IoBName*	*IoBCred*	*IoBDeb*
2010	1005	Asset	35	27
2011	1005	Liability	29	10
2010	1006	Asset	41	30
2010	1006	Liability	31	30

BalSheet2013

Year	*Bank*	*IoBName*	*IoBCred*	*IoBDeb*
2013	1006	Asset	0	53
2013	1006	Liability	33	13

S_3 BalSheetCredit

Bank	*IoBName*	*IoBCred*
1005	Asset	35
1005	Liability	29
1006	Asset	41
1006	Liability	64

BalSheetDebit

Bank	*IoBName*	*IoBDeb*
1005	Asset	27
1005	Liability	10
1006	Asset	83
1006	Liability	43

Fig. 2. Sample source data

2 Handling many sources

Let us first concentrate on cases in which the global schema is completely defined beforehand (and so coinciding with the baseline global schema G_0), while in Section 3, we give some insights about how we handle dynamic global schemas.

We recognize that each source schema S_i can be considered as a variation of S_0. In this, we notice an analogy with the *schema evolution* problem [3, 8], where schemas are derived from one another through the application of simple and standardized operations, namely schema evolution operators.

Thus, we propose a new approach where each S_i is described as an evolution of S_0, that is, as if it were the result of a "virtual" application of one or more schema evolution operators to S_0. In the remainder of the paper we will concentrate only on transformations involving a single operation. We consider a small set of operators, which is however sufficiently expressive and general for most common real scenarios: addition/deletion of attributes, partition of tuples into separate relations, union of relations into a single relation, projection decomposition of a relation into multiple ones, join of multiple relations into a single one, attribute dereferencing (that is extracting an attribute into a foreign key-related relation).

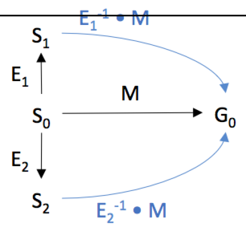


Fig. 3. Mappings for the scenario of variations

2.1 Transformations

Transformations could be modeled in various ways, procedural or declarative. We adopt the notion of schema mapping [4], where the relationships between source and target schemas are described in terms of first-order *s-t tgd* (*source-to-target tuple generating dependencies*). [1]

Hence, each transformation relating the reference schema S_0 to any S_i can be associated with a schema mapping E_i. In the example, S_1 is identical to S_0 (so we have the identity mapping), while S_2 is obtained with a partition on the basis of *Year* values and S_3 with decomposition by projection.

Then, in our example, the mappings between S_0, S_2 and S_3 would be represented as follows:

E_2: BalSheet(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d), $y < 2012$
$\rightarrow$ BalSheet2010-2011(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d);

BalSheet(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d), $y \geq 2012$
$\rightarrow$ BalSheet2013(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d)

E_3: BalSheet(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d)
$\rightarrow$ BalSheetCredit(Bank: b, IoBName: i_n, IoBCred: sum(i_c, groupBy(b, i_n))),
BalSheetDebit(Bank: b, IoBName: i_n, IoBDeb: sum(i_d, groupBy(b, i_n)))

The idea we have just discussed, that the various source schemas are variations of a reference one S_0, allows us to concentrate on the mapping between S_0 and the target schema G_0. This is summarized in Figure 3, where M relates S_0 to G_0 and E_i is the representation of the transformation between S_0 and a specific source S_i. Then, with our approach, the user should focus just on the reference source schema S_0 and describe its mapping M to the global schema. Indeed, we are interested in mappings between the various S_i's and G. Intuitively, this can be obtained by composing the inverse of each E_i with M. Let us go back to our example; with respect to G in Figure 1, we have:

[1] Our tgd's are indeed a bit more complex than those usually found in data exchange settings [4], but their semantics can be defined as an extension of the classical one. Specifically they contain scalar operations (for example a difference of values) and aggregations (like *sum* with *group by* in E_3) in the rhs [2].

M: BalSheet(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d)
 $\rightarrow$ RepoOfBalances(Bank: b, IoBName: i_n, Amount: sum($i_c - i_d$, groupBy(b, i_n)))

where for each *Bank* and *IoBName*, the *Amount* is calculated by subtracting debits from credits. The aggregation then sums the contributions from different *Year*s. In order to make M work also for S_2, we should "undo" the transformations from S_0 to S_2 (E_2) and then apply the original mapping M, as summarized in Figure 3. In other words, the goal would be to find a new mapping $M_2^* = E_2^{-1} \circ M$ and similarly for every other possible source S_i. In most cases, the various results would need to be consolidated, usually by means of a *merge* operator [3], for which there can be various versions whose details are not relevant here.

2.2 Technical issues

Let us now discuss the technical issues. Indeed, our goal is to find a mapping M_i^* between S_i and G_0, given the mappings E_i between S_0 and S_i and M between S_0 and G_0. For this kind of problems, solutions have been discussed in the literature, based on *composition* and *inversion* operators [6, 7][2]. However, such solutions are not sufficient for our case as they would require the existence of an "exact inverse" (a *maximum recovery* [1]) for E_i.

One major problem is that rarely do schema mappings have such an exact inverse, as they may involve information loss; in such cases, source instances cannot be rebuilt from target ones. This is also the case for our transformations: E_3, for instance, is lossy, since it decomposes a relation into two other relations where aggregations are applied. This causes loss of information and the original relation cannot be reconstructed with a join. Vice versa, E_2 is lossless, because the partition (which, by definition, forbids overlapping) can be reversed with a union. However, let us consider, in this case, a relaxed notion of inverse, for example *quasi-inverse*s of schema mappings [7] [3]. It does not exactly rebuild the original instance, but another one, affected by information loss. For example a quasi-inverse of E_3 is

E_3^{-1}: BalSheetCredit(Bank: b, IoBName: i_n, IoBCred: i_c),
 BalSheetDebit(Bank: b, IoBName: i_n, IoBDeb: i_d)
 $\rightarrow \exists y$ (BalSheet(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d))

Notice that E_3^{-1} is far from being an exact inverse because the aggregated contributions of different *Year*s have not been decomposed; moreover *Year* is existentially quantified and its original values have not been restored. Nevertheless, we deem that this inverse is sufficient for our purpose since, in this case, E_3 loses less information than M. In facts, with a simple substitution, the composition

[2] The inverse M^{-1} of a schema mapping M is such that $M^{-1} \circ M = \mathrm{Id}$, where Id is the identity mapping, transforming each instance into itself.

[3] Notice that the original definition does not foresee aggregations, which however we support here.

with M yields:

$$\begin{aligned}
E_3^{-1} \circ M\colon\ & \text{BalSheetCredit(Bank: } b\text{, IoBName: } i_n\text{, IoBCred: } i_c\text{),}\\
& \quad \text{BalSheetDebit(Bank: } b\text{, IoBName: } i_n\text{, IoBDeb: } i_d\text{)}\\
& \to \text{RepoOfBalances(Bank: } b\text{, IoBName: } i_n\text{, Amount: sum(}i_c - i_d\text{, groupBy(}b, i_n\text{)))}
\end{aligned}$$

An interesting idea is the definition of some kind of order relation $\preceq_s$, based on the amount of "transferred information" [1]: $M \preceq_{S_0} E_i$ holds if M transfers less information than E_i when applied to the same source S_0. In this case we observe that $M \preceq_{S_0} E_3$; indeed it is possible to verify this by observing that the instances of G can be generated from the ones of S_3.

Our result is that whenever the order relation $M \preceq_{S_0} E_i$ holds for every S_i, it is possible to integrate all the differing sources, also in presence of aggregations, calculating M_i^* through a suitable notion of inverse, for example quasi-inverses. An intuitive proof for this consists in verifying that the order relation guarantees the presence of some mappings from S_i to G (since E_i transfers more information than M) and one of them, M_i^*, can be calculated by suitably inverting E_i and composing with M.

3 Dynamic target schemas

Let us now consider the more general case, where the global schema is not defined in advance, but it may depend on the specific sources. Different scenarios are indeed possible and the global schema may depend on source data or metadata (names of relations and attributes) or both. Here we concentrate on the most interesting case, which is the one where attributes of the global schema derive from values in the sources. This is also the most relevant condition in real contexts, as it can be the consequence of a common practice in exchange formats between a local and a central organization, that of embedding "schematic" information into data.

An example appears in Figure 2, where the values of *IoBName* ("Asset" and "Liability") are schematic elements as they qualify the other two attributes *IoBCred* and *IoBDeb*: a credit or a debit, in facts, is meaningful only if referred to a specific item of balance. An alternative representation would have four attributes, for expressing credits and debits for each of the two possible item names. In this latter representation, the introduction of a new item of balance, for instance "Equity", would require to alter the schema giving rise to a new variant and so on.

Here we sketch a novel technique to handle such situations. Basically, it consists in the definition of *template tgd's*, an extended version of usual tgd's that allow metadata-data correspondences. They are intended to be used for expressing correspondences between S_0 and the baseline global schema G_0, in the mapping M. We allow quantified variables not only to denote values, but also attribute names. Indeed, the baseline global schema referred to in the rhs of template tgd's is somehow polymorphic, as it contains variables for the attributes that are not known in advance, but their presence and name depend on the source data.

A key feature of our approach is a *rewriting algorithm*, generating usual tgd's out of the template ones, given the source data. For each tuple matching the lhs of a template tgd, the algorithm generates one or more attributes in the schema of the rhs, eventually specifying the global schema G, where all the attributes have been made explicit. The tgd's generated in this way respond to the usual definition and can be then enforced with the common *chase* procedure [5].

An example of template tgd is the following mapping M' between S_0 and G_0:

M': BalSheet(Year: y, Bank: b, IoBName: i_n, IoBCred: i_c, IoBDeb: i_d)
$\to$ RepoOfBalancesAlt(Bank: b, i_n: sum($i_c - i_d$, groupBy(b)))

The rhs describes the baseline global schema G_0, with an attribute variable, i_n, used for unknown attributes. For its value, the expression sum($i_c{-}i_d$, groupBy(b)) specifies that in RepositoryOfBalancesAlt, *Asset* and *Liability* are all calculated as aggregations (an aggregation each), grouping by the only disaggregated attribute, which is *Bank*.

Initially G' contains all the attributes that are fully specified in G_0 (only *Bank* in this case). In the rewriting phase, for each tuple in the lhs, an attribute named after the value bound to i_n ("Asset" or "Liability") is added. [4] At the end, the rewritten tgd, which can be chased and enforced in the usual way, is:

$\widehat{M'}$: BalSheet(Year: y_a, Bank: b, IoBName: "Asset", IoBCred: i_{ca}, IoBDeb: i_{da}),
BalSheet(Year: y_l, Bank: b, IoBName: "Liability" , IoBCred: i_{cl}, IoBDeb: i_{dl})
$\to$ RepoOfBalancesAlt(Bank: b, Asset: sum($i_{ca} - i_{da}$, groupBy(b)),
Liability: sum($i_{cl} - i_{dl}$, groupBy(b)))

4 Conclusions and related work

We provided a solution to the problem of data integration in contexts where there are a large number of different sources whose schema is not completely known beforehand and the global schema can depend on the source data. We consider a single data source as a reference and map it, with schema mappings, into the global schema. All the other sources are described as if they were obtained as a schema evolution of the reference.

While data integration has been studied in a variety of theoretical and practical contexts [12], approaches to handling differences between schemas and representing transformations with mappings are more typically studied at higher level in the model management literature, for example in [3]. More recently, the schema evolution problem has been pursued in [8, 1], however without any connection to the data integration problem or adoption of a declarative relaxed

[4] Some variable renamings are also needed to avoid unwanted matches.

notion of inverse. Theoretical details about inversion and composition are presented in [6], while a formal definition of quasi-inverse schema mappings can be found in [7].

Dynamic global schemas rely on the notion of "schematic transformation", introduced in [11]. Some known approaches are specifically oriented to *query answering* and concentrate on defining appropriate extensions to SQL or relational algebra to have result sets with a schema that varies depending on input data [11, 14]. Others use schematic information to provide a certain degree of schema independence to queries [13]. Since our specific target is data integration, we proposed an extension to the common language of mappings. A similar operation has also been done in [10], with an approach more oriented towards *data exchange* and the goal of solving specific issues such as nesting.

References

1. M. Arenas, J. Perez, J. L. Reutter, and C. Riveros. Foundations of schema mapping management. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 227–238, 2010.
2. P. Atzeni, L. Bellomarini, and F. Bugiotti. Exlengine: Executable schema mappings for statistical data processing. In *Proceedings of the 16th International Conference on Extending Database Technology*, EDBT '13, pages 672–682, New York, NY, USA, 2013. ACM.
3. P. A. Bernstein. Applying model management to classical meta data problems. In *CIDR Conference*, pages 209–220, 2003.
4. R. Fagin, P. Kolaitis, R. Miller, and L. Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.
5. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
6. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
7. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Quasi-inverses of schema mappings. *ACM Trans. Database Syst.*, 33(2), 2008.
8. R. Fagin, P. G. Kolaitis, L. Popa, and W. C. Tan. Schema mapping evolution through composition and inversion. In *Schema Matching and Mapping*, pages 191–222. 2011.
9. H. Garcia-Molina, J. D. Ullman, and J. Widom. *Database Systems: The Complete Book*. Prentice-Hall, Englewood Cliffs, New Jersey, second edition, 2008.
10. M. A. Hernández, P. Papotti, and W. C. Tan. Data exchange with data-metadata translations. *PVLDB*, 1(1):260–273, 2008.
11. L. V. S. Lakshmanan, F. Sadri, and S. N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. pages 476–519, 2001.
12. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
13. U. Masermann and G. Vossen. Sisql: Schema-independent database querying (on and off the web). In *IDEAS*, pages 55–64, 2000.
14. C. M. Wyss and E. L. Robertson. Relational languages for metadata integration. *ACM Trans. Database Syst.*, 30(2):624–660, 2005.

Rewriting-based Check of Chase Termination

Marco Calautti, Sergio Greco, Cristian Molinaro, and Irina Trubitsyna
{calautti,greco,cmolinaro,trubitsyna}@dimes.unical.it

DIMES, Università della Calabria, 87036 Rende (CS), Italy

Keywords: Chase, Data Dependencies, Data Exchange, Data Integration.

1 Introduction

The Chase is a fixpoint algorithm enforcing satisfaction of data dependencies (also called constraints) in databases. It has been proposed more than thirty years ago [2,18] and has seen a revival of interest in recent years in both database theory and practical applications. Indeed, the availability of data coming from different sources easily results in inconsistent or incomplete data (i.e., data not satisfying data dependencies) and, therefore, techniques for fixing inconsistencies are crucial [1,3,5,7,8,13,17].

The chase algorithm is used, directly or indirectly, on an everyday basis by people who design databases, and it is used in commercial systems to reason about the consistency and correctness of a data design. New applications of the chase in meta-data management, ontological reasoning, data exchange, data cleaning, and query optimization have been proposed as well [6,9].

The chase algorithm solves possible violations of constraints by inserting new tuples, possibly containing null values [4]. The following example shows a case where a given database does not satisfy a set of *tuple generating dependencies* (TGDs) and the application of the chase algorithm produces a new consistent database by adding tuples with nulls.

Example 1. Consider the following set of constraints Σ_1 describing departments and their employees:

$$\begin{array}{l} \forall x\,\forall y\ Department(x) \wedge Managed(x,y) \rightarrow Employee(y) \\ \forall x\ Employee(x) \rightarrow \exists\, y\ WorksFor(x,y) \\ \forall x\,\forall y\ WorksFor(x,y) \rightarrow \exists\, z\ Managed(y,z) \end{array}$$

Consider the database $D = \{Department(d), Managed(d,m)\}$. Since the first constraint is not satisfied, the tuple $Employee(m)$ is inserted. This update operation fires the second constraint to insert the tuple $WorksFor(m,\eta_1)$, which in turn fires the third constraint so that the tuple $Managed(\eta_1,\eta_2)$ is added to the database (η_1 and η_2 are new labeled nulls). At this point, the chase terminates since the database is consistent, that is, all dependencies are satisfied. □

Unfortunately, the chase algorithm may not terminate. For instance, in Example 1 if we delete from the first constraint the atom $Department(x)$, the chase never terminates and adds an infinite number of tuples to the database. It has been formally proved in [12] that the problem of deciding whether the chase procedure terminates is semi-decidable. The first and basic effort concerning the formalization of a (decidable) sufficient condition guaranteeing chase termination is *weak acyclicity* [11]. Informally, it checks whether the constraints do not allow for nulls to cyclically propagate inside predicates' positions. Considering the example above, we have that a value in the second position of predicate $Managed$ is copied to $Employee$ (denoted by $M_2 \rightarrow E_1$). This forces the introduction of a new null value in the second position of $WorksFor$, (denoted as $E_1 \overset{*}{\rightarrow} WF_2$); this value is then copied in the first position of $Managed$ ($WF_2 \rightarrow M_1$) and it also forces the introduction of a new null value in the second position of $Managed$ ($WF_2 \overset{*}{\rightarrow} M_2$). Since it is possible to reach position M_2 from itself through a connection of the form $\overset{*}{\rightarrow}$, an infinite number of nulls could be introduced during the chase procedure.

Different extensions of *weak acyclicity* have been proposed. *Safety* [20] and *super-weak acyclicity* [19] identify the positions in which null values can be propagated. *Stratification*-based approaches [10,20,16] analyse whether dependencies may fire each other and thus propagate null values from one to another. See [14] for a comprehensive survey on this topic. Nevertheless, despite the previously mentioned results, there are still important classes of terminating data dependencies which are not identified by any of the previously mentioned criteria: Example 1 showed one such a case. To overcome such limitations, rewriting techniques have been proposed [15,16]. In the following section we give an overview on them and show how the constraints of Example 1 can be rewritten by using predicate adornments in order to allow simple termination conditions (even the simplest one, weak-acyclicity) to understand that the chase procedure terminates. Issues regarding the extension of these techniques to managing also *equality generating dependencies (EGDs)* are discussed in Section 3.

2 Constraint Rewriting

We start by introducing the basic idea of the *Adn* technique [15], which can be used in conjunction with current termination criteria, enabling us to detect more sets of constraints as terminating. The technique consists of rewriting a set of TGDs Σ into a new set Σ^α which is "better" than the original one for the purpose of checking termination. Rather than applying a termination criterion to Σ, the new set Σ^α is used and if Σ^α satisfies the criterion then chase termination for Σ is guaranteed. This allows us to recognize larger classes of constraints for which chase termination is guaranteed: if Σ satisfies chase termination criterion C, then the rewritten set Σ^α satisfies C as well, but the vice versa is not true, that is, there are significant classes of constraints for which Σ^α satisfies C and Σ does not.

Example 2. Consider again the set of TGDs Σ_1. The *Adn* technique first rewrites TGDs by associating strings of b symbols to body atoms and to head positions containing universally quantified variables. Then, f symbols are associated for existentially quantified variables. This new set of TGDs is denoted by $Base(\Sigma_1)$:

$$\forall x\, \forall y\ Department^{b}(x) \wedge Managed^{bb}(x,y) \rightarrow Employee^{b}(y)$$
$$\forall x\ Employee^{b}(x) \rightarrow \exists\, y\ WorksFor^{bf}(x,y)$$
$$\forall x\, \forall y\ WorksFor^{bb}(x,y) \rightarrow \exists\, z\ Managed^{bf}(y,z)$$

Subsequently, because of the presence of atoms $WorksFor^{bf}(x,y)$ and $Managed^{bf}(x,y)$ in $Base(\Sigma_1)$, the rewriting continues by producing the following set of TGDs $Derived(\Sigma_1)$:

$$\forall x\, \forall y\ WorksFor^{bf}(x,y) \rightarrow \exists\, z\ Managed^{ff}(y,z)$$
$$\forall x\, \forall y\ Department^{b}(x) \wedge Managed^{bf}(x,y) \rightarrow Employee^{f}(y)$$
$$\forall x\ Employee^{f}(x) \rightarrow \exists\, y\ WorksFor^{ff}(x,y)$$
$$\forall x\, \forall y\ WorksFor^{ff}(x,y) \rightarrow \exists\, z\ Managed^{ff}(y,z)$$

At this point, the generation of $Derived(\Sigma_1)$ terminates, since the atom $Department^{b}(x)$ cannot be joined with $Managed^{ff}(x,y)$ to produce a new adorned TGD. The rewritten set of TGDs $Adn(\Sigma_1)$ is weakly-acyclic, whereas the original set Σ_1 is not recognized by any chase termination criteria. □

Rewriting Algorithm Improvement. The rewriting algorithm *Adn* has been further improved into the Adn^{+} algorithm [16] by using different adornments for each existentially quantified variable and by considering how TGDs may fire each other in the generation of adorned atoms. During the rewriting process, this algorithm also performs a basic cyclicity check, allowing to eventually determine the termination of the chase, without necessarily relying on other criteria. The new criterion is called *Acyclicity*. To the best of our knowledge, the class of TGDs recognized by this criterion is the most general class known so far.

3 Adding EGDs

In the previous sections, we have considered the case where all constraints are TGDs. In this section, we show how the chase termination problem radically changes when we allow also EGDs.

Given a set of TGDs Σ for which the chase does not terminate, we can show that the addition of EGDs to Σ may allow to have a terminating chase sequence. On the other hand, if the chase always terminates for Σ, adding EGDs to Σ may make the chase of Σ non-terminating.

Example 3. Consider the following two sets of constraints Σ_3 (left) and Σ_3' (right):

$$\begin{array}{llll} r_1: & \forall x\ A(x) \rightarrow \exists y\ N(y) & r_1': & \forall x\ N(x) \rightarrow \exists y \exists z\, S(x,y,z) \\ r_2: & \forall x\ N(x) \rightarrow \exists y\ E(x,y) & r_2': & \forall x \forall y \forall z\ S(x,y,y) \rightarrow N(y) \\ r_3: & \forall x \forall y\ E(x,y) \rightarrow N(y) & r_3': & \forall x \forall y \forall z\ S(x,y,z) \rightarrow T(x,y,z) \\ r_4: & \forall x \forall y\ E(x,y) \rightarrow x = y & r_4': & \forall x \forall y \forall z\ T(x,y,z) \rightarrow y = z \end{array}$$

and the database $D = \{N(a)\}$. The chase applied to the database D and the subset of TGDs $\{r_1, r_2\}$ of Σ_3 is not terminating as it introduces an infinite number *of tuples* $E(\eta_1, \eta 2), E(\eta_3, \eta_1), ...$ *The introduction of the EGD* r_3 *allows to have a terminating sequence, which produces the universal solution* $\{N(a), E(a, a)\}$.

The subset of TGDs $\{r'_1, r'_2, r'_3\}$ *of* Σ'_3 *is terminating for all database instances as recognized by several criteria (e.g., super-weak acyclicity). However, the chase fixpoint applied to* Σ'_3 *and the database* D *is non-terminating as it introduces an infinite number of tuples* $S(a, \eta_1, \eta_1), T(a, \eta_1, \eta_1), N(\eta_1)$, $S(\eta_1, \eta_2, \eta_2)$, $T(\eta_1, \eta_2, \eta_2), N(\eta_2), ...$ □

As shown in the previous example, when for a set of dependencies it is not the case that every chase sequence is terminating, the existence of at least one terminating chase sequence, for every database, might still be guaranteed. Thus, one could extend rewriting techniques such as Adn^+ to sets of TGDs and EGDs, in order to find whether there exists, for every database, at least one terminating chase sequence. In order to cope with the aforementioned issues, algorithm Adn^+ can be extended in such a way that some adornments generated by rewriting TGDs are changed in order to satisfy the head equalities of EGDs. Specifically, the algorithm first tries to adorn as many EGDs as possible, and then consider the rewriting of a single TGD. The basic idea is illustrated in the following example.

Example 4. Consider the set of dependencies Σ_3 of Example 3. As initially EGD r_4 cannot be adorned, TGD r_1 is rewritten into:

$$\forall x\ A^b(x) \to \exists y\ N^{f_1}(y)$$

and r_2 is rewritten into:

$$\forall x\ N^b(x) \to \exists y\ E^{bf_2}(x, y)$$

Now, EGD r_4 can be used to "merge" distinct symbols. This is accomplished by constructing the following adorned version of r_4 using the atom $E^{bf_2}(x, y)$:

$$\forall x\,\forall y\ E^{bf_2}(x, y) \to x = y$$

This indicates that every occurrence of the symbol f_2 in the obtained adorned dependencies has to be replaced with b, thereby obtaining:

$$\begin{array}{l}\forall x\ A^b(x) \to \exists y\ N^{f_1}(y)\\ \forall x\,\forall y\ N^b(x) \to \exists y\ E^{bb}(x, y)\\ \forall x\,\forall y\ E^{bb}(x, y) \to x = y\end{array}$$

Then, TGD r_3 is adorned, obtaining:

$$\forall x\,\forall y\ E^{bb}(x, y) \to N^b(y)$$

Then, TGD r_2 is adorned using atom $N^{f_1}(x)$, obtaining:

$$\forall x\,\forall y\ N^{f_1}(x) \to \exists y\ E^{f_1 f_3}(x, y)$$

Again, EGD r_4 is used, getting:

$$\forall x \, \forall y \; E^{f_1 f_3}(x, y) \rightarrow x = y$$

Consequently, f_3 is replaced with f_1 and we get:

$$\begin{array}{l}
\forall x \; A^b(x) \rightarrow \exists y \; N^{f_1}(y) \\
\forall x \, \forall y \; N^b(x) \rightarrow \exists y \; E^{bb}(x, y) \\
\forall x \, \forall y \; E^{bb}(x, y) \rightarrow N^b(y) \\
\forall x \, \forall y \; N^{f_1}(x) \rightarrow \exists y \; E^{f_1 f_1}(x, y) \\
\forall x \, \forall y \; E^{bb}(x, y) \rightarrow x = y \\
\forall x \, \forall y \; E^{f_1 f_1}(x, y) \rightarrow x = y
\end{array}$$

Finally, atom $E^{f_1 f_1}(x, y)$ is used to adorn r_3, obtaining:

$$\forall x \, \forall y \; E^{f_1 f_1}(x, y) \rightarrow N^{f_1}(y)$$

At this point, the rewriting stops, since no new adorned dependency can be constructed. Intuitively, the algorithm identifies the existence of a terminating chase sequence because symbols f_1, f_2, f_3, which represent nulls constructed w.r.t. the symbols occurring in the body of the TGD, are not "cyclic" in the following sense. Symbol f_1 "depends on" symbol b in $body(r_1)$, f_2 depends on symbol b in $body(r_2)$, and f_3 depends on symbol f_1. Since no pair of symbols f_i, f_j in the final set of dependency is such that f_i depends on f_j and vice versa, the set Σ_3 has a terminating chase sequence. □

References

1. F. N. Afrati and P. G. Kolaitis. Repair checking in inconsistent databases: algorithms and complexity. In *ICDT*, pages 31–41, 2009.
2. A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM Trans. Database Syst.*, 4(3):297–314, 1979.
3. M. Arenas, L. E. Bertossi, and J. Chomicki. Consistent query answers in inconsistent databases. In *PODS*, pages 68–79, 1999.
4. C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM J. Comput.*, 13(1):76–98, 1984.
5. L. E. Bertossi. Consistent query answering in databases. *SIGMOD Record*, 35(2):68–76, 2006.
6. A. Cali, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *PVLDB*, 3(1):554–565, 2010.
7. L. Caroprese, S. Greco, and E. Zumpano. Active integrity constraints for database consistency maintenance. *IEEE Trans. Knowl. Data Eng.*, 21(7):1042–1058, 2009.
8. J. Chomicki. Consistent query answering: Five easy pieces. In *ICDT*, pages 1–17, 2007.
9. G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, pages 133–142, 2007.
10. A. Deutsch, A. Nash, and J. B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.

11. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Th. Comp. Sc.*, 336(1):89–124, 2005.
12. T. Gogacz and J. Marcinkowski. All-instances termination of chase is undecidable. In *ICALP*, pages 293–304, 2014.
13. G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *TKDE*, 15(6):1389–1408, 2003.
14. S. Greco, C. Molinaro, and F. Spezzano. *Incomplete Data and Data Dependencies in Relational Databases.* Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.
15. S. Greco and F. Spezzano. Chase termination: A constraints rewriting approach. *PVLDB*, 3(1):93–104, 2010.
16. S. Greco, F. Spezzano, and I. Trubitsyna. Stratification criteria and rewriting techniques for checking chase termination. *PVLDB*, 4(11):1158–1168, 2011.
17. M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, pages 233–246, 2002.
18. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
19. B. Marnette. Generalized schema-mappings: from termination to tractability. In *PODS*, pages 13–22, 2009.
20. M. Meier, M. Schmidt, and G. Lausen. On chase termination beyond stratification. *CoRR*, abs/0906.4228, 2009.

Navigational Queries Based on Frontier-Guarded Datalog: Preliminary Results[⋆]

Meghyn Bienvenu[1], Magdalena Ortiz[2], and Mantas Šimkus[2]

[1] LRI - CNRS & Université Paris Sud
[2] Institute of Information Systems, Vienna University of Technology

Abstract. In this paper, we introduce a navigational query language that extends binary frontier-guarded Datalog by allowing regular expressions in rule bodies and a limited use of higher-arity intensional predicates. Our query language strictly extends conjunctive two-way regular path queries (C2RPQs) and captures some of the key features advocated in recent works aimed at extending C2RPQs. We compare our language to existing proposals and establish decidability with elementary complexity of query evaluation in the presence of ontologies.

1 Introduction

The current importance of graph databases has led to renewed interest in navigational query languages that ask for nodes connected by paths that satisfy given patterns. Conjunctive two-way regular path queries (C2RPQs) and their unions (UC2RPQs), which simultaneously extend both the well-known conjunctive queries (CQs) and two-way regular path queries (RPQs), have established themselves as fundamental query languages for accessing graph databases. However, recent works have argued that such queries are lacking important features, which has motivated several extensions. For example, C2RPQs have been extended with *path variables* to support naming, comparing, and outputting paths [2]. Other extensions are motivated by the fact that C2RPQs and similar languages can only describe patterns over graphs labeled with a finite alphabet, and aim to overcome this by allowing values from possibly infinite datasets [11]. A third direction aims at increasing the *navigational power* of C2RPQs and related languages, that is, allowing queries to express more complex patterns that cannot be expressed using regular languages. This paper pursues the latter direction.

A very natural way to increase the navigational power of C2RPQs is to use *nested regular expressions (NREs)*, that can enforce that at some points along a path there exists an outgoing path, which is itself described by a (nested) regular expression. *(Conjunctive) nested 2RPQs ((C)N2RPQs)*, which extend (C)2RPQs by allowing NREs in the place of plain regular expressions, have received significant attention recently [4,3,14], due to their improved navigational capabilities. For example, consider a graph database of academic relationships between researchers that contains advisor and co-author relationships, as well as the fields of expertise of the researchers. With (conjunctive) regular path queries, one can find pairs of people that are connected by an arbitrary long chain of advisor or co-author relations, or find three researchers from different fields that are

[⋆] This work has been supported by ANR project PAGODA (ANR-12-JS02-007-01) and the Austrian Science Fund (FWF) projects T515 and P25207-N23.

pairwise connected by such a chain. However, we cannot find pairs connected by an arbitrary long chain of advisor or co-author relations, where additionally each node must, for example, have an academic ancestor that is a biologist. The latter can be easily done using nested regular expressions. However, neither C2RPQs nor CN2RPQs can express the query that requires that all people along such a chain have *the same* biologist as academic ancestor. Another query that cannot be expressed is to find pairs connected by an arbitrary chain where all nodes are connected by both the co-author and the advisor relationship. These examples illustrate two shortcomings of UCN2RPQs that have been pointed out in the literature, namely, the inability to express transitive closure over complex relations defined using UCN2RPQs and the impossibility of joining objects inside a nested expression.

Recent works have aimed at overcoming these limitations. A query language that tackles the first issue was introduced by Bourhis et al. [5] with the name of *nested positive 2RPQs*, and further studied more recently (using a different syntax) by Reutter et al. [13] under the name of *regular queries*. Essentially, this language allows full UCN2RPQs to be nested inside the regular expressions of UCN2RPQs, allowing one to define relations such as the pair of all researchers that are connected by a chain of parallel advisor and coauthor relation. This language strictly increases the expressive power of UC2NRPQs at little or no computational cost: its query evaluation problem over plain graph databases remains complete for NL in data complexity and for NP complete in combined complexity, while its query containment problem is complete for 2EXPSPACE, and even in EXPSPACE for a less succinct version of equal expressive power [13]. A way to overcome the second limitation can be found in *monadically defined queries (MODEQs)* [15], which extend monadic Datalog by using some special 'flag' constants, that are then instantiated with a given tuple of ordinary constants that behave as 'parameters'. These queries can easily express relations like pairs connected by a chain of coauthors where all of them have the same biologist academic ancestor, using one flag constant as a 'placeholder' for this common object. However, they cannot express even the simple nested patterns in CN2RPQs. This limitation is overcome by *nested MODEQs* [15]. Some variations and generalizations of MODEQs and nested MODEQs, under the generic name of *(nested) flag-and-check queries* were studied recently in [6]. Many of these languages strictly generalize regular queries and nested MODEQs, and can express all of the example queries we have mentioned. However, query evaluation becomes P-complete for data complexity, and PSPACE-complete for combined complexity, and query containment becomes non-elementary.

In this paper, we introduce a navigational query language that extends binary frontier-guarded Datalog by allowing regular expressions in rule bodies. It also allows the use of intensional predicates of higher arity, but the additional positions are designated as 'parameter positions' and subjected to special syntactic restrictions. These parameter positions can simulate the use of flag constants in flag-and-check queries and allow our query language to refer to common points inside nested subqueries as in the examples discussed above. They cannot fully simulate the power of the flag-and-check special constants in the nested versions of the query languages though, but this restriction seems to play a crucial role in avoiding the non-elementary increase in the complexity of reasoning. Our query language also allows for nesting subqueries inside regular expressions, and can in fact express all the examples discuss so far. However, in terms of nesting, our language is orthogonal to regular queries and to the nested flag-and-check queries, since they only allow for acyclic nesting, by considering only non-recursive Datalog or by allowing to nest only queries of a strictly lower nesting level. By contrast, our query language naturally supports recursive nesting: that is, an intensional predicate can occur in the C2RPQ defining it. In exchange for this recursiveness, our queries impose a 'guardedness' requirement that only allows to define complex nested relations for pairs of objects that are guaranteed to be related by an extensional relation in the input database. The resulting language seems to provide an interesting trade-off between expressiveness and complexity. As mentioned earlier, it can express all of the queries mentioned above, and the complexity of query evaluation is similar to that of nested MODEQs: P-complete for data complexity, NP-complete for combined complexity if the number of parameters is bounded, and PSPACE-hard and in EXPTIME for the full language (without arity restrictions). We do not study query containment in this preliminary note, but consider instead another challenging problem: query evaluation in the presence of ontological knowledge.

In the paradigm of *ontology-mediated query answering*, (graph) databases are enriched with an *ontology* that captures domain knowledge and defines additional relations for querying. These ontologies are usually expressed in *description logics* (DLs) [1] or in closely related formalisms based on existential rules. In general, these languages support recursion and can imply the existence of additional objects (unnamed constants or nulls). In the presence of such ontologies, the query answering problem consists in computing the *certain answers* to the queries over the set of all the potentially infinitely databases that extend the input data and satisfy all the formulas in the ontology. Hence, query answering is algorithmically much harder than the usual query evaluation over databases. In fact, for most query languages studied so far in this setting, query evaluation in the presence of ontologies formulated using *expressive DLs* is at least as hard as query containment. Query evaluation in the presence of rich ontologies has been studied for plain CN2RPQs [4], but not for any of the extensions mentioned above. Bourhis et al. provided tight non-elementary bounds for a closely related language, positive first order logic with a parametrized unary transitive closure operator (PFO+TC$_1$), which falls strictly between regular queries and nested MODEQs [5], thus implying non-elementary hardness of query answering for other variations of nested flag-and-check queries that generalize nested MODEQs. The language we consider here, in contrast, allows for elementary query answering even for very expressive DLs, strongly suggesting that its containment problem may also remain elementary.

2 Guarded Regular Queries

In frontier-guarded rules, all variables in a rule head must occur together in a body atom. We extend such rules by allowing regular expressions in the rule bodies. We allow the use of *intensional predicates* of arbitrary arity, but we distinguish two different kinds of positions: each predicate has one or two *main positions* and any number of *parameter positions*. Inside the regular expressions, and with respect to frontier-guardedness, we consider only the main positions and the intensional predicates behave as unary and binary. The additional parameter positions are not subjected to guardedness, but an additional syntactic requirement that ensures they are not 'forgotten'.

Syntax Let $\mathsf{N_P}$, $\mathsf{N_V}$, and $\mathsf{N_I}$ be countably infinite sets of *predicate names*, *variable names*, and *individual names*, respectively. The set $\mathsf{N_V}$ is the disjoint union of the set N_V^o of *ordinary* variables and the set N_V^p of *parameter* variables. We assume that there is a subset $\mathsf{N_{Ans}} \subseteq \mathsf{N_P}$ that consists of a single distinguished k-ary answer predicate ans^k for every $k \geq 0$; when convenient, we will omit the arity and use ans in place of ans^k.

Each predicate $p \in \mathsf{N_P} \setminus \mathsf{N_{Ans}}$ has a *main arity* of either 1 or 2, and a *parameter arity* that can be any natural number; we will write $\mathsf{arity}(p) = (k, n)$ to indicate that p has main arity k and parameter arity n. Note that usual unary and binary predicates correspond to predicates having parameter arity 0.

We define unary and binary alphabets, Σ_1 and Σ_2, as follows:

$$\Sigma_1 = \{p_{\boldsymbol{z}} \mid p \in \mathsf{N_P} \setminus \mathsf{N_{Ans}}, \mathsf{arity}(p) = (1, n), \boldsymbol{z} \in (\mathsf{N}_\mathsf{V}^p \cup \mathsf{N_I})^n\} \cup \{\{a\} \mid a \in \mathsf{N_I}\}$$
$$\Sigma_2 = \{p_{\boldsymbol{z}},\ p_{\boldsymbol{z}}^- \mid p \in \mathsf{N_P} \setminus \mathsf{N_{Ans}}, \mathsf{arity}(p) = (2, n), \boldsymbol{z} \in (\mathsf{N}_\mathsf{V}^p \cup \mathsf{N_I})^n\} \cup \{U? \mid U \in \Sigma_1\}$$

We consider regular languages over Σ_2, which are defined in the usual way and represented as regular expressions or non-deterministic finite automata (NFAs).

There are three kinds of *atoms*: *unary atoms* $E_1(x)$ with $E_1 \in \Sigma_1$ and $x \in \mathsf{N_V} \cup \mathsf{N_I}$, *binary atoms* $E_2(x, y)$ with E_2 a regular language over Σ_2 and $x, y \in \mathsf{N_V} \cup \mathsf{N_I}$, and *answer atoms* $\mathsf{ans}^k(\boldsymbol{z})$ with $\boldsymbol{z} \in (\mathsf{N_V} \cup \mathsf{N_I})^k$. We use the term *base atom* to mean unary atoms, answer atoms, and binary atoms $E_2(x, y)$ with $E_2 \in \Sigma_2$. For a unary or binary atom α, its set $\mathsf{mvars}(\alpha)$ of *main variables* is defined as follows: $\mathsf{mvars}(E_1(x)) = \{x\} \cap \mathsf{N_V}$ and $\mathsf{mvars}(E_2(x, y)) = \{x, y\} \cap \mathsf{N_V}$. We also define the set $\mathsf{pvars}(\alpha)$ of *parameter variables* of α: $\mathsf{pvars}(\{a\}(x)) = \emptyset$, $\mathsf{pvars}(p_{\boldsymbol{z}}(x)) = \boldsymbol{z} \cap \mathsf{N_V}$, and $\mathsf{pvars}(E_2(x, y)) = \{z \in \mathsf{N_V} \mid p_{\boldsymbol{z}}$ appears in E_2 and $z \in \boldsymbol{z}\}$.

A *rule* ρ is an expression of the form $h \leftarrow b_1, \ldots, b_n$, where every b_i is an atom and h is a base atom. We call h the *head* of ρ and $b_1, \ldots, b_n$ the *body*. As usual, we require that every head variable also appears in the body.

A *query* is a set Q of rules such that exactly one predicate ans^k from $\mathsf{N_{Ans}}$ occurs, and this predicate only appears in rule heads. If Q contains ans^k, then the *arity* of Q is k. We will distinguish two types of predicates relative to Q: *intensional predicates* that occur in some rule head in Q, and *extensional predicates* that do not appear in any rule head. We denote by $\mathsf{int}(Q)$ (resp. $\mathsf{ext}(Q)$) the set of intensional (resp. extensional) predicates relative to Q. Observe that $\mathsf{ext}(Q) = \mathsf{N_P} \setminus \mathsf{int}(Q)$.

In this paper, we will mainly focus on *guarded regular queries*, in which the following conditions hold for every rule $h \leftarrow b_1, \ldots, b_n$ with head predicate p from $\mathsf{N_P} \setminus \mathsf{N_{Ans}}$:

1. $\mathsf{mvars}(h) \subseteq \mathsf{mvars}(b_i)$ for some basic atom b_i, $1 \leq i \leq n$,
2. $\mathsf{pvars}(b_i) \subseteq \mathsf{pvars}(h)$ for all $1 \leq i \leq n$.

Note that Condition 1 imposes frontier-guardedness w.r.t. the main variables, and Condition 2 ensures that parameter variables occurring in the body of a rule occur also its head, hence they are not 'forgotten', in a similar spirit to the stickiness property considered for existential rules [7].

Semantics The queries we have defined are a special class of Datalog programs. Indeed, an atom $p_{\boldsymbol{z}}(x, y)$ provides an alternative syntax for the standard Datalog atom

$p(x, y, \boldsymbol{z})$, and regular expressions can be naturally expressed in Datalog. Hence, the semantics of our queries is readily obtained from the semantics of Datalog, where rules are viewed as Horn clauses in first-order logic. However, for our purposes, it will prove more convenient to have a direct semantics for our queries, which we introduce next.

A *(predicate) signature* is any set Σ of predicate names. A *Σ-interpretation* $\mathcal{I}$ is a tuple $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set of *domain elements* and $\cdot^{\mathcal{I}}$ is a function that assigns (i) to each individual c an element $c^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, and (ii) to each $p \in \Sigma$ a relation $p^{\mathcal{I}} \subseteq (\Delta^{\mathcal{I}})^{k+n}$ where $\mathsf{arity}(p) = (k, n)$. If Σ is irrelevant, we simply call $\mathcal{I}$ an *interpretation*. For $\Sigma \subseteq \Sigma'$, we say that a Σ'-interpretation $\mathcal{I}'$ *extends* a Σ-interpretation $\mathcal{I}$ if the following conditions hold: (i) $\Delta^{\mathcal{I}} = \Delta^{\mathcal{I}'}$, (ii) $c^{\mathcal{I}} = c^{\mathcal{I}'}$ for every individual name c, and (iii) $p^{\mathcal{I}} = p^{\mathcal{I}'}$ for all $p \in \Sigma$.

Consider a Σ-interpretation $\mathcal{I}$, a predicate $p \in \Sigma$, and a function $\mu : \mathsf{N}_{\mathsf{V}}^{p} \to \Delta^{\mathcal{I}}$. The interpretation of symbols from $\Sigma_1 \cup \Sigma_2$ in $\mathcal{I}$ under μ is defined as follows:

$$\begin{array}{lr}
\{a\}^{\mathcal{I},\mu} = a^{\mathcal{I}} & \text{for } \{a\} \in \Sigma_1 \\
(p_{\boldsymbol{z}})^{\mathcal{I},\mu} = \{e \in \Delta^{\mathcal{I}} \mid (e, \mu(\boldsymbol{z}^{\mathcal{I}})) \in p^{\mathcal{I}}\} & \text{for } p_{\boldsymbol{z}} \in \Sigma_1 \\
(p_{\boldsymbol{z}})^{\mathcal{I},\mu} = \{(e, e') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (e, e', \mu(\boldsymbol{z}^{\mathcal{I}})) \in p^{\mathcal{I}}\} & \text{for } p_{\boldsymbol{z}} \in \Sigma_2 \\
(p_{\boldsymbol{z}}^{-})^{\mathcal{I},\mu} = \{(e, e') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (e', e) \in (p_{\boldsymbol{z}})^{\mathcal{I},\mu}\} & \text{for } p_{\boldsymbol{z}}^{-} \in \Sigma_2 \\
(U?)^{\mathcal{I},\mu} = \{(e, e) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid e \in U^{\mathcal{I},\mu}\} & \text{for } U? \in \Sigma_2
\end{array}$$

where $\boldsymbol{z}^{\mathcal{I}}$ denotes the result of replacing every individual c in $\boldsymbol{z}$ by $c^{\mathcal{I}}$ and $\mu(\boldsymbol{z})$ denotes the result of replacing every $v \in \mathsf{N}_{\mathsf{V}}^{p}$ in $\boldsymbol{z}^{\mathcal{I}}$ by $\mu(v)$. For a general regular expression E built over Σ_2, the binary relation $E^{\mathcal{I},\mu}$ is obtained using the composition, union and transitive closure of the base relations above.

Next suppose that in addition to $\mathcal{I}$ and μ, we have a function $\pi : \mathsf{N}_{\mathsf{V}}^{o} \to \Delta^{\mathcal{I}}$. We define the following satisfaction and entailment relations:

$$\begin{array}{rll}
\mathcal{I}, \mu, \pi \models \alpha(\boldsymbol{t}) & \text{iff} & \pi(\mu(\boldsymbol{t}^{\mathcal{I}})) \in \alpha^{\mathcal{I},\mu} \\
\mathcal{I}, \mu, \pi \models h \leftarrow b_1, \dots, b_n & \text{iff} & \mathcal{I}, \mu, \pi \models h \text{ or } \mathcal{I}, \mu, \pi \not\models b_i \text{ for some } i \\
\mathcal{I}, \mu \models \rho & \text{iff} & \mathcal{I}, \mu, \pi \models \rho \text{ for all } \pi : \mathsf{N}_{\mathsf{V}}^{o} \to \Delta^{\mathcal{I}} \\
\mathcal{I} \models \rho & \text{iff} & \mathcal{I}, \mu \models \rho \text{ for all } \mu : \mathsf{N}_{\mathsf{V}}^{p} \to \Delta^{\mathcal{I}} \\
\mathcal{I} \models Q & \text{iff} & \mathcal{I} \models \rho \text{ for all rules } \rho \in Q
\end{array}$$

Now consider a k-ary query Q, and let Σ_Q be the set of predicate names that occur in Q. We call a Σ-interpretation $\mathcal{I}$ *relevant for* Q if $\Sigma \cap \mathsf{int}(Q) = \emptyset$ and $\mathsf{ext}(Q) \cap \Sigma_Q \subseteq \Sigma$. If $\mathcal{I}$ is a Σ-interpretation that is relevant for Q, then we denote by $Q(\mathcal{I})$ the set of all k-tuples $\boldsymbol{c}$ such that $\mathcal{I}' \models \mathsf{ans}(\boldsymbol{c})$ for every $\Sigma \cup \mathsf{int}(Q)$-interpretation $\mathcal{I}'$ with $\mathcal{I}' \models Q$ which extends $\mathcal{I}$.

Examples We now illustrate the expressiveness of our query language with some examples. Consider a graph database with the following relationships between researchers: *was PhD advisor of* (ad), *is a coauthor of* (ca), *is a collaborator of* (cl), and *shares research topic* (srt). The following query Q_1 finds pairs connected by an arbitrarily long chain of parallel ad and ca relations.

$$\mathsf{ans}(x, y) \leftarrow ac^{*}(x, y) \qquad ac\,(x, y) \leftarrow ad \cup ad^{-}(x, y), ca(x, y)$$

This simple query cannot be expressed as a CN2RPQ; this is proven for an analogous query in [5]. The next query Q_2 finds all pairs that are connected by a chain of potential collaborators, where *potential collaborators* are people that either are collaborators, or they share a research topic and are connected by a chain of coauthors.

$$\mathsf{ans}(x,y) \leftarrow \; pcl^*(x,y) \qquad pcl(x,y) \leftarrow \; cl(x,y)$$
$$pcl(x,y) \leftarrow \; srt(x,y), ca^*(x,y)$$

This query is very similar to the regular query given in Example 2 of [13]. It can also be shown along the lines of [5] that this query is not expressible as a plain CN2RPQ. The next query Q_3 finds pairs of individuals who are connected by a chain of srt who are all connected to some z by a coauthorship chain.

$$\mathsf{ans}(x,y) \leftarrow \; cca_z^*(x,y) \qquad cca_z(x,y) \leftarrow \; srt(x,y), ca^*(x,z), ca^*(y,z)$$

This query is not expressible as a regular query; again, this can be shown as in [5].

Finally, we give an example query Q_4 that nests over guarded recursion. It builds a *preferred collaborator relation* that contains any pair of coauthors that have coauthored papers with a shared advisor, as well as pairs of x and y that can respectively reach researchers x' and y' via a chain of both srt and cl, where x' and y' are themselves preferred collaborators:

$$prefcl(x,y) \leftarrow \; ca(x,y), ca(x,z), ca(y,z), ad(x,z), ad(y,z)$$
$$prefcl(x,y) \leftarrow \; ca(x,y), srt^*(x,x'), cl^*(x,x'), srt^*(y,y'), cl^*(y,y'), prefcl(x',y')$$

Note that Q_4 does not conform to the syntax of regular queries, and we conjecture that it is expressible neither as a regular query, nor as a nested flag-and-check query.

3 Related Query Languages

In this section, we compare in more detail guarded regular queries with related languages that also extend the navigational capabilities of C2RPQs. First, it is easy to see that any nested CN2RPQ can be easily rewritten as a guarded regular query, by simply replacing exhaustively each nested expression $\langle E\rangle$ by $p_E?$ for a fresh intensional predicate p_E with $\mathsf{arity}(p_E) = (1,0)$, and adding a rule $p_E(x) \leftarrow E(x,y)$.

Now we compare guarded regular queries with other extensions of CN2RPQs that are closer in terms of expressiveness.

Regular queries were introduced in [13] as non-recursive binary Datalog programs that additionally allow for *transitive closure* rules of the form $P(x,y) \leftarrow R^+(x,y)$. An alternative definition, that is equivalent but exponentially less succinct, is given by taking non-recursive binary Datalog rules and allowing in the body regular expressions over extensional predicates and expressions of the form $R^+(x,y)$ (for arbitrary predicates), but restricting intensional predicates to occur only once in rule bodies. The query containment problem for regular queries is complete for 2EXPSPACE, but only EXPSPACE-complete (like for plain C2RPQs) if the alternative syntax is considered. The query evaluation problem is also not harder than for C2RPQs: NL-complete in data complexity and NP-complete complete in combined complexity.

Regular queries and guarded regular queries are closely related, but there are some significant differences. We have already shown above that their expressive power is orthogonal. Regular queries have no parameters and nesting is not recursive. On the other hand, their nesting is not subject to a guardedness restriction.
Nested positive 2RPQs [6] allow for positive Boolean combinations of 2RPQs, where the regular expressions may use queries of strictly lower nesting degree as binary predicates. This language is in fact equivalent to the regular queries from [13].

PFO+TC$_1$ is another extension of CN2RPQs studied by Bourhis et. al. that extends positive first-order logic with a parameterized unary transitive closure operator. This language is strictly more expressive than regular queries, and it can fully support the 'parameters' (restricted higher arities) that we allow in guarded regular queries. However, guarded regular queries and PFO+TC$_1$ are incomparable. On the one hand, PFO+TC$_1$ has NL-data complexity, while guarded regular queries are complete for P, as we show in Section 4. This implies that some guarded regular queries cannot be expressed in PFO+TC$_1$. On the other hand, PFO+TC$_1$ has been shown to have non-elementary time complexity for query containment [6] and for query answering in the presence of some DL ontologies [5]. We argue below that the latter problem is elementary for guarded regular queries,[3], which implies that not all PFO+TC$_1$ queries are expressible as guarded regular queries without a non-elementary blow-up in the formula size.

Expressive Datalog fragments having a decidable containment problem have been recently studied by Bourhis et al. [6]. They consider well-known languages like linear, monadic, and frontier-guarded Datalog and extend them with special 'flag' constants that behave analogously to the parameter positions of guarded regular queries (this had already been done for monadic Datalog [15]); they also consider *nested* versions of these languages. These *nested flag-and-check queries* are in general more expressive than regular queries and even generalize PFO+TC$_1$ [5]. However, similarly as for PFO+TC$_1$, this has a high computational cost and the containment problem becomes non-elementary [6]. Also, in contrast to guarded regular queries, all these nested queries allow only for acyclic nesting, while we allow for cyclic but guarded nesting.

One of the considered languages, GQ$^+$, is obtained by taking frontier guarded flag-and-check programs (that is, frontier guarded Datalog programs with special parameter constants), and allowing to nest into their bodies analogous programs of lower nesting level. Note that, in contrast, we have a (linear) Datalog fragment (that captures C2RPQs) at the inner query level, and use frontier-guardedness to do cyclic nesting of queries.

4 Evaluating Guarded Regular Queries

In this section, we study the complexity of query evaluation over graph databases and then outline an algorithm for query answering in the presence of DL ontologies.

Query Evaluation over Graph Databases The following proposition summarizes what we know about the complexity of evaluating guarded regular queries.

Proposition 1. *Evaluation of guarded regular queries over graph databases is:*

[3] We have not yet studied containment of guarded regular queries, but we believe it may be elementary as well.

– P-*complete for data complexity;*
– PSPACE-*hard and in* EXPTIME *for combined complexity;*
– NP-*complete for combined complexity, when the parameter arity of predicates is bounded by a fixed constant.*

Proof. For Statement 1, observe that guarded regular queries fall between binary monadic Datalog and full Datalog (cf. Section 2), and for both of these languages, the query evaluation problem is P-complete in data complexity.

Membership in EXPTIME for combined complexity is a direct consequence of the EXPTIME-completeness of query evaluation in Datalog. To obtain the PSPACE lower bound, we modify the Datalog program that was used in [10] to show PSPACE-hardness of evaluating linear sirups, in order to ensure that the resulting query satisfies the 'stickiness' requirement (Condition 2).

For Statement 3, the lower bound comes from the NP-hardness of CQ evaluation, and the upper bound is obtained by guessing a sequence of polynomial number of rule applications (along with the required homomorphisms) that leads to deriving $\mathsf{ans}(\boldsymbol{c})$.

Query Answering with DL KBs We sketch an algorithm for answering guarded regular queries over DL KBs. We do not give the details of particular DLs, but instead present the *forest model property* that many DLs have and on which our method relies.

Each DL KB $\mathcal{K}$ is defined over some set Σ of unary and binary relations with parameter arity 0. The semantics of $\mathcal{K}$ is given as a set $\mathcal{M}(\mathcal{K})$ of Σ-interpretations. Given a k-ary query Q, we let $Q(\mathcal{K})$ be the set of k-tuples $\boldsymbol{c}$ of individuals such that $\boldsymbol{c} \in Q(\mathcal{I})$ for all $\mathcal{I} \in \mathcal{M}(\mathcal{K})$. The set $Q(\mathcal{K})$ is called the *certain answer* to Q over $\mathcal{K}$.

Let $\mathbf{N}$ be the set of natural numbers and denote by $w \in \mathbf{N}^+$ the set of all non-empty words over $\mathbf{N}$. A Σ- interpretation $\mathcal{I}$ is *forest-shaped* if the following conditions hold:

(i) $\Delta^{\mathcal{I}} \subseteq \mathbf{N}^+$ is *prefix-closed*, i.e., if $w \cdot n \in \Delta^{\mathcal{I}}$ and $w \neq \epsilon$, then $w \in \Delta^{\mathcal{I}}$;
(ii) if $(e, e', \boldsymbol{d}) \in p^{\mathcal{I}}$ for some $p \in \Sigma$, then one of the following holds: (a) $e = e'$, (b) $|e| = 1$ and $|e'| = 1$, or (c) $e' = e \cdot n$ or $e = e' \cdot n$ for some $n \in \mathbf{N}$.

A KB $\mathcal{K}$ is said to possess the *forest model property* if for any $\mathcal{I} \in \mathcal{M}(\mathcal{K})$ there exists a forest-shaped interpretation $\mathcal{I}' \in \mathcal{M}(\mathcal{K})$ such that:

(i) there exists a homomorphism from $\mathcal{I}'$ to $\mathcal{I}$, and
(ii) $\Delta^{\mathcal{I}} \subseteq \{0, \ldots, r(|\mathcal{K}|)\}^*$, where r is a polynomial and $|\mathcal{K}|$ denotes the size of $\mathcal{K}$.

The forest-shaped $\mathcal{I} \in \mathcal{M}(\mathcal{K})$ that satisfy the condition $\Delta^{\mathcal{I}} \subseteq \{0, \ldots, r(|\mathcal{K}|)\}^*$ are the *forest models* of $\mathcal{K}$. It is well known that KBs written in many popular DLs have this property (cf. [8]).

Consider a KB $\mathcal{K}$ with the forest model property, a query Q of arity k, and a k-tuple $\boldsymbol{c}$ of individuals. Since our query language is monotone, it follows that for such a KB $\mathcal{K}$, $\boldsymbol{c} \notin Q(\mathcal{K})$ implies $\boldsymbol{c} \notin Q(\mathcal{I})$ for some forest model $\mathcal{I} \in \mathcal{M}(\mathcal{K})$. If $\mathcal{K}$ is in a standard DL like $\mathcal{ALCHIQ}$, one can define a tree automaton that recognizes (an encoding of) forest-shaped $\mathcal{I} \in \mathcal{M}(\mathcal{K})$. However, due to the semantics of the query language, deciding $\boldsymbol{c} \notin Q(\mathcal{I})$ involves checking the existence of an extension $\mathcal{I}'$ of $\mathcal{I}$ such that $\mathcal{I}' \models Q$ and $\mathcal{I}' \not\models \mathsf{ans}(\boldsymbol{c})$, and such $\mathcal{I}'$ need not be forest-shaped. In fact, unrestricted queries are undecidable and thus can enforce interpretations that are not tree-shaped and which cannot be coded into forest-shaped interpretations in a meaningful way.

To obtain decidability for guarded regular queries, we show that it is sufficient to consider only certain forest-shaped extensions. Given a Σ-interpretation $\mathcal{I}$, we let $\mathsf{pdom}(\mathcal{I})$ denote the domain elements that appear as parameter values in $\mathcal{I}$, i.e. those $e \in \Delta^{\mathcal{I}}$ for which there exists a predicate name $p \in \Sigma$ and a tuple $\langle e_1, \ldots, e_{k+n} \rangle \in p^{\mathcal{I}}$ with $e = e_i$ and $i > k$, where $\mathsf{arity}(p) = (k, n)$. We can prove the following:

Proposition 2. *Consider a KB $\mathcal{K}$ with the forest model property, a guarded regular query Q of arity k, and a k-tuple $\mathbf{c}$ of individuals. Let m be the maximal number of parameter variables over all rules of Q. Then $\mathbf{c} \notin Q(\mathcal{K})$ iff (†) there exists a forest-shaped $\mathcal{I} \in \mathcal{M}(\mathcal{K})$ such that for any set $g \subseteq \Delta^{\mathcal{I}}$ with $|g| \leq m$ there exists a $\Sigma \cup \mathsf{int}(Q)$-interpretation $\mathcal{I}'$ such that: (i) $\mathcal{I}'$ is forest-shaped, (ii) $\mathcal{I}'$ extends $\mathcal{I}$, (iii) $\mathcal{I}', \mu \models \rho$ for every $\rho \in Q$ and every μ with $\mathsf{ran}(\mu) \subseteq g$, (iv) $\mathcal{I}' \not\models \mathsf{ans}(\mathbf{c})$, and (v) $\mathsf{pdom}(\mathcal{I}') \subseteq g$.*

We next argue that condition (†) from Proposition 2 can be decided by employing tree automata. It is standard to represent forest-shaped interpretations of a DL KB as labeled trees upon which a tree automaton can operate (see e.g. [9]). In such an encoding, 'roots' of the original forest-shaped interpretation correspond to children of a dummy root in the tree representation. It is a bit more involved to obtain a tree representation of forest-shaped Σ-interpretations when Σ contains predicates with non-zero parameter arities. However, if the input interpretation $\mathcal{I}$ is such that $|\mathsf{pdom}(\mathcal{I})| \leq k$ for a finite k, then we can essentially employ the standard representation, except that we may need to increase the alphabet exponentially in the parameter arity of original relations.

The automata algorithm to check (†) can be briefly described as follows:

1. If $\mathcal{K}$ is in a standard DL like $\mathcal{ALCHIQ}$, we can build a nondeterministic tree automaton (NTA) A_1 that recognizes (the tree encoding of) tuples $(\mathcal{I}, g, \mathcal{I}')$ such that $\mathcal{I}$ is a forest-shaped model of $\mathcal{K}$, $g \subseteq \Delta^{\mathcal{I}}$ is such that $|g| \leq m$, and $\mathcal{I}'$ is a $\Sigma \cup \mathsf{int}(Q)$-interpretation that satisfies conditions (i)-(v) of Proposition 2. The naïve construction of A_1 requires a triple-exponential number of states and can be done by adapting the construction in [9].
2. We build an NTA A_2 that recognizes tuples $(\mathcal{I}, g)$ such that some triple of the form $(\mathcal{I}, g, \mathcal{I}')$ is recognized by A_1. Technically, this is done by performing a projection operation on A_1, that projects away the third component of recognized triples.
3. We complement A_2 to obtain A_3. Intuitively, A_3 accepts tuples $(\mathcal{I}, g)$ that A_2 rejects, i.e. for which there is no triple $(\mathcal{I}, g, \mathcal{I}')$ that satisfies conditions (i)-(v). This step causes an exponential blowup in the number of states.
4. By projecting away the second component, we can obtain from A_3 an NTA A_4 that accepts forest interpretations $\mathcal{I}$ for which there exists g such that we cannot find an $\mathcal{I}'$ that satisfies conditions (i)-(v).
5. By complementing A_4, we obtain an NTA that checks condition (†) in Proposition 2. This step also causes an exponential blowup in the number of states.

Due to the complexity results on testing nonemptiness of NTAs in [12], the above construction leads to an 5ExpTime upper bound. For standard Horn DLs like Horn-$\mathcal{SHIQ}$, $\mathcal{EL}$ or *DL-Lite*, we can improve the upper bound to 4ExpTime; these DLs have the so-called *canonical model property* and thus the last automata complementation step in the above construction is not necessary.

5 Future Work

In this paper, we proposed guarded regular queries as a new navigational query language and provided some first results regarding its relation to related proposals and the complexity and decidability of query evaluation, in particular in the presence of DL ontologies. There are number of questions that we plan to tackle in future work. First, we will complete our formal comparison of the expressive power of guarded regular queries, with the aim of showing that they are indeed incomparable to related existing formalisms. Second, we will pursue our study of the computational properties of the language by pinpointing the precise complexity of query answering in the presence of DL ontologies and by investigating the query containment problem. Finally, extending guarded regular queries by relaxing the use of parameters and frontier-guardedness is another challenging but interesting problem.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM TODS*, 37(4):31, 2012.
3. P. Barceló, J. Pérez, and J. L. Reutter. Relative expressiveness of nested regular expressions. In *Proc. of AMW'12*, CEUR Workshop Proceedings 866, pages 180–195, 2012.
4. M. Bienvenu, D. Calvanese, M. Ortiz, and M. Šimkus. Nested regular path queries in description logics. In *Proc. of KR 2014*. AAAI Press, 2014.
5. P. Bourhis, M. Krötzsch, and S. Rudolph. How to best nest regular path queries. In *Proc. of DL'14*, volume 1193, pages 404–415. CEUR-WS.org, 2014.
6. P. Bourhis, M. Krötzsch, and S. Rudolph. Query containment for highly expressive datalog fragments. *CoRR*, abs/1406.7801, 2014.
7. A. Calì, G. Gottlob, and A. Pieris. Advanced processing for ontological queries. *Proc. VLDB Endow.*, 3(1-2):554–565, Sept. 2010.
8. D. Calvanese, T. Eiter, and M. Ortiz. Regular path queries in expressive description logics with nominals. In *Proc. of IJCAI 2009*, pages 714–720, 2009.
9. D. Calvanese, T. Eiter, and M. Ortiz. Answering regular path queries in expressive description logics via alternating tree-automata. *Inf. Comput.*, 237:12–55, 2014.
10. G. Gottlob and C. Papadimitriou. On the complexity of single-rule datalog queries. *Information and Computation*, 183(1):104 – 122, 2003.
11. L. Libkin and D. Vrgoc. Regular path queries on graphs with data. In A. Deutsch, editor, *Proc. of ICDT'12*, pages 74–85. ACM, 2012.
12. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(12):69 – 107, 1995.
13. J. Reutter, M. Romero, and M. Y. Vardi. Regular queries on graph databases. In M. Arenas and M. Ugarte, editors, *Proc. of ICDT'15*, 2015. To appear.
14. J. L. Reutter. Containment of nested regular expressions. CoRR Technical Report arXiv:1304.2637, arXiv.org e-Print archive, 2013.
15. S. Rudolph and M. Krötzsch. Flag & check: Data access with monadically defined queries. In *Proc. of PODS'13*, pages 151–162. ACM, 2013.

LDQL: A Language for Linked Data Queries

Olaf Hartig

http://olafhartig.de

Abstract In this paper, we propose LDQL, that is, a language to query Linked Data on the Web. The novelty of LDQL is that it enables a user to express separately (i) patterns that describe the expected query result, and (ii) Web navigation paths that select the data sources to be used for computing the result. As a downside of this expressiveness, we find that for some LDQL queries, a complete execution is not possible in practice. To address this issue, we show a syntactic property based on which systems can identify queries that do not have this limitation.

1 Introduction

In recent years an increasing amount of structured data has been published and interlinked on the World Wide Web (WWW) in adherence to the Linked Data principles [2,3]. These principles are based on standard Web technologies. In particular, (i) the Hypertext Transfer Protocol (HTTP) is used to access data, (ii) HTTP-based Uniform Resource Identifiers (URIs) are used as identifiers for entities described in the data, and (iii) the Resource Description Framework (RDF) is used as data model. Then, any HTTP URI in an RDF triple presents a *data link* that enables software clients to retrieve more data by looking up the URI with an HTTP request. The adoption of these principles has lead to the creation of a globally distributed dataspace: the *Web of Linked Data*.

From a graph database perspective the Web of Linked Data can be conceived of as two graphs of different types: First, the (virtual) union of all RDF triples in the Web of Linked Data represents an *RDF graph* [6]; this graph is distributed over the documents in the Web of Linked Data. Second, these documents constitute the nodes of a *link graph* whose edges represent the aforementioned data links that connect the documents.

The emergence of the Web of Linked Data makes possible an *online execution* of declarative queries over up-to-date data from a virtually unbounded set of data sources, each of which is readily accessible without any need for implementing source-specific APIs or wrappers. This possibility has spawned research interest in approaches to query Linked Data on the WWW as if it was a single (distributed) database. For an overview on query execution techniques proposed in this context refer to [12].

While there does not exist a standard language for expressing such queries, a few options have been proposed. In particular, a first strand of research focuses on extending the scope of SPARQL—which is the standard query language for centralized collections of RDF data [9]—such that an evaluation of SPARQL queries over Linked Data on the WWW has a well-defined semantics [4,10,11,14,16]. A second strand of research focuses on navigational languages (e.g., NautiLOD [8]). A commonality of all these proposals is that querying the aforementioned two graphs that comprise the Web of Linked Data (i.e., the link graph and the RDF data graph) is inherently entangled in each of the proposed query formalisms. That is, for any query expressed in such a formalism, the definition of query-relevant regions of the link graph and the definition of query-relevant data from the RDF graph within the specified regions depend on one another.

In this paper, we study the alternative approach of avoiding such an inherent dependency. To this end, we propose a novel query language for Linked Data which we call LDQL. In contrast to the aforementioned query formalisms, LDQL separates query components for selecting regions of the link graph from components for specifying the query result that has to be constructed from the data in the selected regions. For the former, LDQL introduces the notion of a link path expression, which is a form of nested regular expression, and for the latter we use SPARQL. More precisely, the most basic type of LDQL queries consists of a link path expression and a SPARQL graph pattern. Additionally, such queries can be combined using conjunctions and disjunctions, where some subqueries may provide the starting points of navigation for other subqueries.

The downside of the expressiveness provided by LDQL are queries for which a complete execution is not possible in practice (because of the lack of a complete, up-to-date data catalog that is inherent to the WWW). To capture this issue formally, we define a notion of Web-safeness for LDQL queries. Then, the obvious question that arises is how to identify those LDQL queries that are Web-safe. Our contribution to answer this question is to show a sufficient syntactic condition for Web-safeness.

The paper is structured as follows: Section 2 introduces a data model that provides the basis for defining the semantics of LDQL formally. In Section 3 we define LDQL and show simple algebraic properties. Thereafter, in Section 4 we focus on Web-safeness. Section 5 concludes the paper and sketches future work. For proofs of the formal results in this paper we refer to the extended version of this paper [13].

2 Data Model

In this section we introduce a structural data model that captures the concept of a Web of Linked Data formally. As usual [4,8,10,11,14,16], for the definitions and analysis in this paper, we assume that the Web is fixed during the execution of any single query.

We use the RDF data model [6] as a basis for our model of a Web of Linked Data. That is, we assume three pairwise disjoint, infinite sets $\mathcal{U}$ (URIs), $\mathcal{B}$ (blank nodes), and $\mathcal{L}$ (literals). Then, an *RDF triple* is a tuple $\langle s, p, o \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$. For any RDF triple $t = \langle s, p, o \rangle$ we write $\mathrm{uris}(t)$ to denote the set of all URIs in t.

Additionally, we assume another infinite set $\mathcal{D}$ that is disjoint from $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{L}$, respectively. We refer to elements in this set as *Linked Data documents*, or *documents* for short, and use them to represent the concept of Web documents from which Linked Data can be extracted. Hence, we assume a function, say data, that maps each document $d \in \mathcal{D}$ to a finite set of RDF triples $\mathrm{data}(d) \subseteq (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ such that the data of each document uses a unique set of blank nodes; i.e., for any pair of distinct documents $d, d' \in \mathcal{D}$, and any two RDF triples $t = \langle s, p, o \rangle$ and $t' = \langle s', p', o' \rangle$ with $t \in \mathrm{data}(d)$ and $t' \in \mathrm{data}(d')$, it holds that $\{s, p, o\} \cap \{s', p', o'\} \cap \mathcal{B} = \emptyset$.

Given these preliminaries, we are ready to define a *Web of Linked Data*.

Definition 1. A **Web of Linked Data** is a tuple $W = \langle D, adoc \rangle$ that consists of a set of documents $D \subset \mathcal{D}$ and a partial function $adoc \colon \mathcal{U} \to D$ that is surjective.

Function $adoc$ of a Web of Linked Data $W = \langle D, adoc \rangle$ captures the relationship between the URIs that can be looked up in this Web and the documents that can be retrieved by such lookups. Since not every URI can be looked up, the function is partial. For any URI $u \in \mathcal{U}$ with $u \in \mathrm{dom}(adoc)$ (i.e., any URI that can be looked up in W), document $d = adoc(u)$ can be considered the authoritative source of data for u in W (hence, the name $adoc$). To accommodate for documents that are authoritative for multiple URIs, we do not require injectivity for function $adoc$. However, we require surjectivity because we conceive documents as irrelevant for a Web of Linked Data if they cannot be retrieved by any URI lookup in this Web.

In this paper, we assume that the set of documents D in any Web of Linked Data $W = \langle D, adoc \rangle$ is finite, in which case we say W is *finite* [11]. Moreover, given a Web of Linked Data $W = \langle D, adoc \rangle$, we say that a URI $u \in \mathrm{uris}(t)$ in an RDF triple $t = \langle s, p, o \rangle$ establishes a *data link* to a document $d \in D$ if $adoc(u) = d$.

As a final concept, our data model formalizes the aforementioned notion of a link graph in which directed edges represent data links between documents. In the following formal definition of this graph, each edge is associated with a label that identifies both the particular RDF triple and the URI in this triple that establishes the corresponding data link. These labels shall provide the basis for defining the navigational component of our query language (cf. Section 3.1).

Definition 2. The **link graph** of a Web of Linked Data $W = \langle D, adoc \rangle$ is a directed, edge-labeled multigraph $\langle D, E \rangle$ whose vertices are all documents in W, and which has a directed edge $\langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}} \rangle \in E$ from document $d_{\mathsf{src}} \in D$ to document $d_{\mathsf{tgt}} \in D$ if the data of d_{src} contains an RDF triple t with a URI $u \in \mathrm{uris}(t)$ that establishes a data link to d_{tgt}; the edge is labeled with both the triple t and the URI u. Hence, the set of edges $E \subseteq D \times \mathcal{T} \times \mathcal{U} \times D$ is defined as follows:

$$E = \big\{ \langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}} \rangle \,\big|\, t \in \mathrm{data}(d_{\mathsf{src}}) \text{ such that } u \in \mathrm{uris}(t) \text{ and } adoc(u) = d_{\mathsf{tgt}} \big\}.$$

Example 1. As a running example for this paper assume a simple Web of Linked Data $W_{\mathsf{ex}} = \langle D_{\mathsf{ex}}, adoc_{\mathsf{ex}} \rangle$ with three documents, d_{A}, d_{B} and d_{C} (i.e., $D_{\mathsf{ex}} = \{d_{\mathsf{A}}, d_{\mathsf{B}}, d_{\mathsf{C}}\}$). The data in these documents are the following sets of RDF triples:

$$\begin{aligned} \mathrm{data}(d_{\mathsf{A}}) = \{\langle u_{\mathsf{A}}, p_1, u_{\mathsf{B}} \rangle,& \qquad & \mathrm{data}(d_{\mathsf{B}}) = \{\langle u_{\mathsf{B}}, p_1, u_{\mathsf{C}} \rangle\};\\ \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}} \rangle\};& \qquad & \mathrm{data}(d_{\mathsf{C}}) = \{\langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}} \rangle\}; \end{aligned}$$

and function $adoc_{\mathsf{ex}}$ is given as follows: $adoc_{\mathsf{ex}}(u_{\mathsf{A}}) = d_{\mathsf{A}}$, $adoc_{\mathsf{ex}}(u_{\mathsf{B}}) = d_{\mathsf{B}}$, and $adoc_{\mathsf{ex}}(u_{\mathsf{C}}) = d_{\mathsf{C}}$ (i.e., $\mathrm{dom}(adoc_{\mathsf{ex}}) = \{u_{\mathsf{A}}, u_{\mathsf{B}}, u_{\mathsf{C}}\}$). This Web contains 8 data links. For instance, URI u_{A} in the RDF triple in $\mathrm{data}(d_{\mathsf{C}})$ establishes a data link to document d_{A}. Hence, the corresponding edge in the link graph of W_{ex} is $\big\langle d_{\mathsf{C}}, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{A}}, d_{\mathsf{A}} \big\rangle$. Figure 1 illustrates this link graph with all 8 edges.

3 Definition of LDQL

This section defines our Linked Data query language, LDQL. LDQL queries are meant to be evaluated over a Web of Linked Data and each such query is built from two types

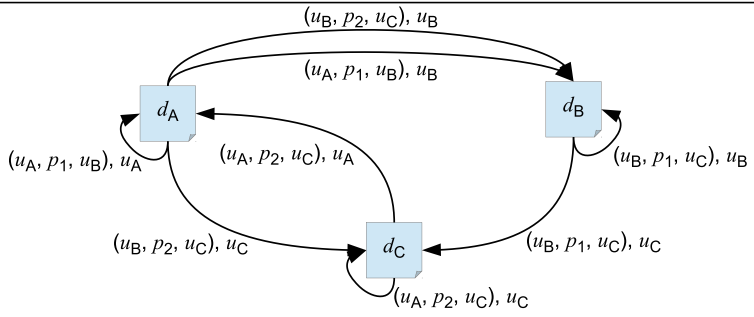


Figure 1. The link graph of our example Web of Linked Data W_{ex}.

of components: First, there are link path expressions for selecting query-relevant documents of the queried Web of Linked Data; second, there are SPARQL graph patterns for specifying the query result that has to be constructed from the data in the selected documents. For this paper, we assume that the reader is familiar with the definition of SPARQL [9], including the algebraic formalization introduced by Pérez et al. [15].

In the following, we first formalize our notion of link path expressions. Thereafter, we define a syntax and semantics of LDQL queries and show simple algebraic properties of such queries. For brevity, we introduce LDQL also based on an algebraic syntax.

3.1 Link Path Expressions

Link Path Expressions (LPEs) are a form of nested regular expressions for navigation over the link graph of a Web of Linked Data. The base case for LPEs is to select single link graph edges in the context of a designated URI. To this end, we introduce *link patterns*, that is, tuples $\langle s,p,o\rangle \in \big(\mathcal{U}\cup\{\_,+\}\big)\times\big(\mathcal{U}\cup\{\_,+\}\big)\times\big(\mathcal{U}\cup\mathcal{L}\cup\{\_,+\}\big)$, where $\_$ is a special symbol that denotes a wildcard and $+$ is another special symbol that denotes a placeholder for the context URI. Then, a link graph edge $\langle d_{\text{src}}, t, u, d_{\text{tgt}}\rangle$ with $t = \langle x_1, x_2, x_3\rangle$ *matches* a link pattern $lp = \langle y_1, y_2, y_3\rangle$ in the context of a URI u_{ctx} if for each $i \in \{1,2,3\}$, any of the following three properties holds:

1. $y_i = \_$, or
2. $y_i = +$ and $x_i = u_{\text{ctx}}$, or
3. $y_i = x_i$.

Example 2. Consider the link pattern $lp_{\text{ex}} = \langle \_, p_2, \_\rangle$. The link graph of our example Web W_{ex} (cf. Example 1) contains two edges that match lp_{ex} in the context of URI u_A, namely, the edges $\big\langle d_A, \langle u_B, p_2, u_C\rangle, u_B, d_B\big\rangle$ and $\big\langle d_A, \langle u_B, p_2, u_C\rangle, u_C, d_C\big\rangle$.

The rationale for adopting the notion of a designated context URI in our definition of link patterns is to support cases in which link graph navigation has to be focused solely on data links that are *authoritative*, where we call a data link authoritative if it is established by an RDF triple in a document d_{src} such that d_{src} is the authoritative document for some URI in the triple. Formally, in terms of link graph edges, an edge

$\langle d_{\mathsf{src}}, t, u, d_{\mathsf{tgt}} \rangle \in E$ in the link graph $\langle D, E \rangle$ of a Web of Linked Data $W = \langle D, adoc \rangle$ represents an authoritative data link if $adoc(u') = d_{\mathsf{src}}$ for some URI $u' \in \mathrm{uris}(t)$.

Example 3. Consider the link pattern $lp'_{\mathsf{ex}} = \langle \_, p_2, + \rangle$, which is a more restrictive variation of the link pattern discussed in the previous example. In contrast to lp_{ex}, there does not exist any edge in the link graph of W_{ex} that matches lp'_{ex} in the context of URI u_{A}. Any such edge would have to represent an authoritative data link; more precisely, the RDF triple of such a data link must have the context URI (i.e., u_{A}) in the object position, which is not the case for the two edges that match the less restrictive link pattern lp_{ex}. Notice also that, if we use URI u_{C} as context instead, there exists an edge that matches link pattern lp'_{ex} in the context of u_{C}, namely $\big\langle d_{\mathsf{C}}, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{A}}, d_{\mathsf{A}} \big\rangle$.

Given the notion of a link pattern, we define LPEs as nested regular expressions over the alphabet that consists of all possible link patterns. Hence, an LPE is an expression defined by the following grammar (where lp is an arbitrary link pattern):

$$lpe = \varepsilon \mid lp \mid lpe/lpe \mid lpe|lpe \mid lpe^* \mid [lpe]$$

Note that our notion of LPEs does not provide an operator for navigating paths in their inverse direction. The reason for omitting such an operator is that traversing arbitrary data links backwards is impossible on the WWW.

The semantics of LPEs is based on a designated context URI (similar to our notion of matching edges for link patterns). In particular, the semantics requires that each path of link graph edges that satisfies an LPE starts from the document that is authoritative for the given context URI. Then, the result of evaluating an LPE represents the end nodes of all such paths. More precisely, the result is a set of URIs where, informally, the lookup of each such URI returns the document that constitutes the end node of the corresponding path. The following definition captures the semantics of LPEs formally.

Definition 3. Let *lpe* be an LPE, let $W = \langle D, adoc \rangle$ be a Web of Linked Data with link graph $\langle D, E \rangle$, and let u_{ctx} be a URI. The **u_{ctx}-based evaluation** of *lpe* over W, denoted by $[\![lpe]\!]_W^{u_{\mathsf{ctx}}}$, is a set of URIs that is empty if $u_{\mathsf{ctx}} \notin \mathrm{dom}(adoc)$; if $u_{\mathsf{ctx}} \in \mathrm{dom}(adoc)$, then the set is defined recursively as follows (where $d_{\mathsf{ctx}} = adoc(u_{\mathsf{ctx}})$, lp is a link pattern, and lpe, lpe_1, lpe_2 are arbitrary LPEs):

$$\begin{aligned}
[\![\varepsilon]\!]_W^{u_{\mathsf{ctx}}} &= \{u_{\mathsf{ctx}}\} \\
[\![lp]\!]_W^{u_{\mathsf{ctx}}} &= \{u \mid \langle d_{\mathsf{ctx}}, t, u, d \rangle \in E \text{ that matches } lp \text{ in the context of } u_{\mathsf{ctx}}\} \\
[\![lpe_1/lpe_2]\!]_W^{u_{\mathsf{ctx}}} &= \{u \in [\![lpe_2]\!]_W^{u'} \mid u' \in [\![lpe_1]\!]_W^{u_{\mathsf{ctx}}}\} \\
[\![lpe_1|lpe_2]\!]_W^{u_{\mathsf{ctx}}} &= [\![lpe_1]\!]_W^{u_{\mathsf{ctx}}} \cup [\![lpe_2]\!]_W^{u_{\mathsf{ctx}}} \\
[\![lpe^*]\!]_W^{u_{\mathsf{ctx}}} &= \{u_{\mathsf{ctx}}\} \cup [\![lpe]\!]_W^{u_{\mathsf{ctx}}} \cup [\![lpe/lpe]\!]_W^{u_{\mathsf{ctx}}} \cup [\![lpe/lpe/lpe]\!]_W^{u_{\mathsf{ctx}}} \cup \ldots \\
[\![\,[lpe]\,]\!]_W^{u_{\mathsf{ctx}}} &= \{u_{\mathsf{ctx}} \mid [\![lpe]\!]_W^{u_{\mathsf{ctx}}} \neq \emptyset\}.
\end{aligned}$$

Example 4. Let lpe_{ex} be the LPE $\langle \_, p_2, \_ \rangle$ (i.e., the link pattern discussed in Example 2). For our example Web W_{ex} and context URI u_{A}, we know by Example 2 that the link graph edges $\big\langle d_{\mathsf{A}}, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{B}}, d_{\mathsf{B}} \big\rangle$ and $\big\langle d_{\mathsf{A}}, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}} \rangle, u_{\mathsf{C}}, d_{\mathsf{C}} \big\rangle$ match the pattern in the context of URI u_{A}. Hence, the LPE selects documents $d_{\mathsf{B}} = adoc_{\mathsf{ex}}(u_{\mathsf{B}})$ and $d_{\mathsf{C}} = adoc_{\mathsf{ex}}(u_{\mathsf{C}})$. More precisely, we have $[\![lpe_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}^{u_{\mathsf{A}}} = \{u_{\mathsf{B}}, u_{\mathsf{C}}\}$.

Example 5. As another example consider the slightly more complex LPE lpe'_{ex} which is of the form $\langle\_, p_1, \_\rangle^{*}/[\langle\_, p_2, \_\rangle]$. This LPE selects documents that can be reached via arbitrarily long paths of data links with predicate p_1 and, additionally, have some outgoing data link with predicate p_2. For our example Web W_{ex} and context URI u_{A}, all three documents, d_{A}, d_{B} and d_{C}, can be reached via a p_1-path from URI u_{A}, but only d_{A} and d_{C} pass the p_2-related test. Hence, we have $[\![lpe'_{\mathsf{ex}}]\!]^{u_{\mathsf{A}}}_{W_{\mathsf{ex}}} = \{u_{\mathsf{A}}, u_{\mathsf{C}}\}$.

While LPEs allows users to select documents from the queried Web of Linked Data, in the context of LDQL these documents are used to form an RDF dataset for evaluating a given SPARQL graph pattern. Formally, we specify this dataset as follows.

Definition 4. Let u_{ctx} be a URI, let *lpe* be an LPE, and let $W = \langle D, adoc\rangle$ be a Web of Linked Data. The u_{ctx}**-***lpe***-selected dataset** over W is an RDF dataset $\mathfrak{D} = \langle G_0, \mathcal{N}\rangle$ (as per [9,1]) whose default graph G_0 is the union of all Named Graphs of the dataset, and that contains a Named Graph $\langle u, G\rangle \in \mathcal{N}$ for every URI $u \in [\![lpe]\!]^{u_{\mathsf{ctx}}}_{W}$ whose lookup (in W) results in retrieving a document; hence,

$$G_0 = \textstyle\bigcup_{\langle u,G\rangle\in\mathcal{N}} G, \text{ and}$$
$$\mathcal{N} = \{\langle u, G\rangle \mid u \in [\![lpe]\!]^{u_{\mathsf{ctx}}}_{W} \text{ and } u \in \mathrm{dom}(adoc) \text{ and } G = \mathrm{data}(adoc(u))\}.$$

Example 6. Consider context URI u_{A} and the aforementioned example LPE lpe'_{ex} for which we have $[\![lpe'_{\mathsf{ex}}]\!]^{u_{\mathsf{A}}}_{W_{\mathsf{ex}}} = \{u_{\mathsf{A}}, u_{\mathsf{C}}\}$ (cf. Example 5). Then, the u_{A}-lpe'_{ex}-selected dataset over W_{ex} is $\mathfrak{D}_{\mathsf{ex}} = \langle G_{\mathsf{ex}}, \mathcal{N}_{\mathsf{ex}}\rangle$ with $\mathcal{N}_{\mathsf{ex}} = \{\langle u_{\mathsf{A}}, \mathrm{data}(d_{\mathsf{A}})\rangle, \langle u_{\mathsf{C}}, \mathrm{data}(d_{\mathsf{C}})\rangle\}$ and, thus, $G_{\mathsf{ex}} = \{\langle u_{\mathsf{A}}, p_1, u_{\mathsf{B}}\rangle, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}}\rangle, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}}\rangle\}$ (cf. Example 1).

3.2 LDQL Queries

We are now ready to define LDQL queries formally.

Definition 5. An **LDQL query** is defined recursively as follows:

1. Any tuple $q = \langle u, lpe, P\rangle$ consisting of a URI u, an LPE *lpe*, and a SPARQL graph pattern P is an LDQL query—hereafter, referred to as a *basic LDQL query*;
2. Any tuple $q = \langle ?v, lpe, P\rangle$ consisting of a variable $?v$, an LPE *lpe*, and a SPARQL graph pattern P is an LDQL query;
3. If q_1 and q_2 are LDQL queries, then $(q_1 \,\mathsf{AND}\, q_2)$ and $(q_1 \,\mathsf{UNION}\, q_2)$ are LDQL queries.

Before introducing the formal semantics of LDQL queries, we provide some intuition about it. As per Definition 5, the most basic type of an LDQL query consists of a URI, an LPE, and a SPARQL graph pattern. Informally, the result of such a query $q = \langle u, lpe, P\rangle$ is the set of SPARQL solution mappings that can be obtained by evaluating the SPARQL pattern P over the u-lpe-selected dataset.

Example 7. Consider a basic LDQL query $\langle u_{\mathsf{A}}, lpe'_{\mathsf{ex}}, B_{\mathsf{ex}}\rangle$ whose LPE is the aforementioned example LPE lpe'_{ex} (cf. Example 5) and whose SPARQL graph pattern is a basic graph pattern that contains two triple patterns, $B_{\mathsf{ex}} = \{\langle ?x, p_1, ?y\rangle, \langle ?x, p_2, ?z\rangle\}$. According to the semantics outlined above (and defined formally below), the result of this query over our example Web W_{ex} consists of a single solution mapping, namely $\mu = \{?x \mapsto u_{\mathsf{A}}, ?y \mapsto u_{\mathsf{B}}, ?z \mapsto u_{\mathsf{C}}\}$. To see why we obtain this result, recall from Example 6 that the default graph of the u_{A}-lpe'_{ex}-selected dataset over W_{ex} is $G_{\mathsf{ex}} = \{\langle u_{\mathsf{A}}, p_1, u_{\mathsf{B}}\rangle, \langle u_{\mathsf{B}}, p_2, u_{\mathsf{C}}\rangle, \langle u_{\mathsf{A}}, p_2, u_{\mathsf{C}}\rangle\}$. Then, by the standard (set) semantics of SPARQL [1], the result of evaluating basic graph pattern B_{ex} over this graph G_{ex} is a singleton set that contains solution mapping μ.

The other types of LDQL queries also have a set of solution mappings as their result: Operators AND and UNION represent conjunctions and disjunctions, respectively. The second base type of LDQL queries, tuples with a variable instead of a fixed (context) URI, is similar to basic LDQL queries with the difference that the variable can range over alternative context URIs; this type of query is meant to be used in conjunctions in which the other conjunct is used to select context URIs at query execution time (e.g., see query q''_{ex} in Example 8 below). The following definition formalizes the query semantics.

Definition 6. The **evaluation** of an LDQL query q over a Web of Linked Data W, denoted by $[\![q]\!]_W$, is defined recursively as follows (where u is a URI, $?v$ is a variable, lpe is an LPE, P is a SPARQL graph pattern, $[\![P]\!]^{\mathfrak{D}}_{G_0}$ denotes the evaluation of P over an RDF graph G_0 in an RDF dataset $\mathfrak{D} = \langle G_0, \mathcal{N}\rangle$ [1, Definition 13.3], and q_1 and q_2 are arbitrary LDQL queries):

$$\begin{aligned}
[\![\langle u, lpe, P\rangle]\!]_W &= [\![P]\!]^{\mathfrak{D}}_{G_0} \qquad (\,\mathfrak{D} = \langle G_0, \mathcal{N}\rangle \text{ is the } u\text{-}lpe\text{-selected dataset over } W),\\
[\![\langle ?v, lpe, P\rangle]\!]_W &= \textstyle\bigcup_{u\in\mathcal{U}}\big([\![\langle u, lpe, P\rangle]\!]_W \bowtie \{\mu_u\}\big) \qquad (\text{ where } \mu_u = \{?v \mapsto u\}\ ),\\
[\![(q_1 \text{ AND } q_2)]\!]_W &= [\![q_1]\!]_W \bowtie [\![q_2]\!]_W,\\
[\![(q_1 \text{ UNION } q_2)]\!]_W &= [\![q_1]\!]_W \cup [\![q_2]\!]_W.
\end{aligned}$$

Example 8. Consider an LDQL query $q_{\mathsf{ex}} = \big\langle ?x, \varepsilon, \langle ?x, p_1, ?z\rangle\big\rangle$, which is of the second base type in Definition 5 (with the SPARQL graph pattern being a single triple pattern). Additionally, let $q'_{\mathsf{ex}} = \big\langle u_{\mathsf{A}}, lpe'_{\mathsf{ex}}, \{\langle ?x, p_1, ?y\rangle, \langle ?x, p_2, ?z\rangle\}\big\rangle$ be the basic LDQL query introduced in Example 7, and let q''_{ex} be the conjunction of these two queries; i.e., $q''_{\mathsf{ex}} = (q_{\mathsf{ex}} \text{ AND } q'_{\mathsf{ex}})$. By Example 7 we know that $[\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}} = \{\mu\}$ (with solution mapping μ as given in Example 7). Furthermore, based on the data given in Example 1, it is easy to see that $[\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}} = \{\mu_1, \mu_2\}$ with $\mu_1 = \{?x \mapsto u_{\mathsf{A}}, ?z \mapsto u_{\mathsf{B}}\}$ and $\mu_2 = \{?x \mapsto u_{\mathsf{B}}, ?z \mapsto u_{\mathsf{C}}\}$. For the evaluation of query q''_{ex} over W_{ex}, the result sets $[\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$ and $[\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$ have to be joined. While for $\mu_1 \in [\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$, there does not exist a compatible join candidate in $[\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$, solution mapping $\mu_2 \in [\![q_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$ is compatible with $\mu \in [\![q'_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}}$. Consequently, we have $[\![q''_{\mathsf{ex}}]\!]_{W_{\mathsf{ex}}} = \{\mu'\}$ with $\mu' = \mu \cup \mu_2$.

3.3 Algebraic Properties of LDQL Queries

As a basis for the discussion in the next section, we show simple algebraic properties.

Lemma 1. *The operators* AND *and* UNION *are associative and commutative, respectively, and the operator* AND *distributes over* UNION*. That is, if q_1, q_2, and q_3 are LDQL queries, then the following semantic equivalences hold:*

- $(q_1 \text{ AND } q_2) \equiv (q_2 \text{ AND } q_1)$;

– $(q_1 \text{ UNION } q_2) \equiv (q_2 \text{ UNION } q_1)$;
– $\big(q_1 \text{ AND } (q_2 \text{ AND } q_3)\big) \equiv \big((q_1 \text{ AND } q_2) \text{ AND } q_3\big)$;
– $\big(q_1 \text{ UNION } (q_2 \text{ UNION } q_3)\big) \equiv \big((q_1 \text{ UNION } q_2) \text{ UNION } q_3\big)$;
– $\big(q_1 \text{ AND } (q_2 \text{ UNION } q_3)\big) \equiv \big((q_1 \text{ AND } q_2) \text{ UNION } (q_1 \text{ AND } q_3)\big)$.

Proof. Since the definition of LDQL operators AND and UNION is equivalent to their SPARQL counterparts, the semantic equivalences in Lemma 1 follow from corresponding equivalences for SPARQL graph patterns as shown by Pérez et al. [15, Lemma 2.5].

Lemma 1 allows us to write sequences of either AND or UNION without parentheses. Hereafter, we assume that every UNION-*free LDQL query* is represented as an LDQL query of the form $(q_1 \text{ AND } q_2 \text{ AND } \dots \text{ AND } q_m)$ where each subquery q_i $(1 \leq i \leq m)$ is either of the form $\langle u, lpe, P\rangle$ (i.e., a basic LDQL query) or of the form $\langle ?v, lpe, P\rangle$. Moreover, we say that an LDQL query is in UNION *normal form* if it is of the form $(q_1 \text{ UNION } q_2 \text{ UNION } \dots \text{ UNION } q_n)$ such that each subquery q_i $(1 \leq i \leq n)$ is a UNION-free LDQL query. The following result is an immediate consequence of Lemma 1.

Corollary 1. *For every LDQL query, there exists a semantically equivalent LDQL query that is in* UNION *normal form.*

4 Web-Safeness of LDQL Queries

In this section we study the “Web-safeness” of LDQL queries, where, informally, we call a query *Web-safe* if a complete execution of the query over the WWW is possible in practice (which is not the case for all LDQL queries as we shall see).

To provide a more formal definition of this notion of Web-safeness we make the following observations. While the mathematical structures introduced by our data model capture the notion of Linked Data on the WWW formally (and, thus, allow us to provide a formal semantics for LDQL queries), in practice, these structures are not available completely for the WWW. For instance, given that an infinite number of strings can be used as HTTP URIs [7], we cannot assume complete information about which URIs are in the domain of the partial function $adoc$ (i.e., can be looked up to retrieve some document) and which are not; in fact, disclosing this information would require a process that systematically tries to look up every possible HTTP URI and, thus, would never terminate because of the aforementioned infiniteness. Therefore, it is also impossible to guarantee the discovery of every document in the set D (without looking up an infinite number of URIs). Consequently, any query whose execution requires a complete enumeration of this set is not feasible in practice. Based on these observations, we define *Web-safeness* of LDQL queries as follows.

Definition 7. An LDQL query q is **Web-safe** if there exists an algorithm that, for any finite Web of Linked Data $W = \langle D, adoc\rangle$, computes $[\![q]\!]_W$ by looking up only a finite number of URIs without assuming an a priori availability of any information about the sets D and $\mathrm{dom}(adoc)$.

Example 9. Recall our example queries q_{ex}, q'_{ex}, and q''_{ex} (cf. Example 8). For query $q_{\mathsf{ex}} = \big\langle ?x, \varepsilon, \langle ?x, p_1, ?z\rangle \big\rangle$, any URI $u \in \mathcal{U}$ may be used to obtain a nonempty subset of the query result as long as a lookup of u retrieves a document whose data includes RDF triples that match $\langle u, p_1, ?z\rangle$. Therefore, without access to D or $\mathrm{dom}(adoc)$ of the queried Web $W = \langle D, adoc\rangle$, the completeness of the computed query result can be guaranteed only by checking each of the infinitely many possible HTTP URIs. Hence, query q_{ex} is *not* Web-safe. In contrast, although it contains q_{ex} as a subquery, query $q''_{\mathsf{ex}} = (q_{\mathsf{ex}}\ \mathsf{AND}\ q'_{\mathsf{ex}})$ is Web-safe, and so is $q'_{\mathsf{ex}} = \langle u_{\mathsf{A}}, lpe'_{\mathsf{ex}}, B_{\mathsf{ex}}\rangle$. A possible execution algorithm for q'_{ex} may first compute $[\![lpe'_{\mathsf{ex}}]\!]^{u_{\mathsf{A}}}_{W}$ by traversing the queried Web W based on the given LPE lpe'_{ex}. Thereafter, the algorithm retrieves documents by looking up all URIs $u \in [\![lpe'_{\mathsf{ex}}]\!]^{u_{\mathsf{A}}}_{W}$ (or simply keeps these documents after the traversal); and, finally, the algorithm evaluates pattern B_{ex} over the union of the RDF triples in the retrieved documents. If W is finite (i.e., contains a finite number of documents) the traversal process requires a finite number of URI lookups only, and so does the retrieval of documents in the second step; the final step does not look up any URI. To see that q''_{ex} is also Web-safe we note that after executing subquery q'_{ex} (e.g., by using the algorithm as outlined before), the execution of the other (non-Web-safe) subquery q_{ex} can be reduced to a finite number of URI lookups, namely the URIs bound to variable $?x$ in solution mappings obtained for subquery q'_{ex}. Although any other URI may also be used to obtain solution mappings for q_{ex}, such solution mappings cannot be joined with any of the solution mappings for q'_{ex} and, thus, are irrelevant for the result of q''_{ex}.

The example illustrates that there exists an LDQL query that is not Web-safe. In fact, it is not difficult to see that the argument for the non-Web-safeness of query q_{ex} as made in the example can be applied to any LDQL query of the form $\langle ?v, lpe, P\rangle$; that is, none of these queries is Web-safe. However, the example also shows that more complex queries that contain such non-Web-safe subqueries may still be Web-safe. Therefore, we now introduce properties to identify LDQL queries that are Web-safe (even if some of their subqueries are not). As a basis for the discussion, we first observe that the Web-safeness of an LDQL query carries over to any semantically equivalent LDQL query.

Fact 1. An LDQL query q is Web-safe if there exists an LDQL query q' such that q and q' are semantically equivalent and q' is Web-safe.

In conjunction with Corollary 1, Fact 1 allows us to focus our discussion on LDQL queries in UNION normal form without losing generality. It is easy to show that such queries are Web-safe if all of their (top-level) subqueries are Web-safe:

Proposition 1. *An LDQL query of the form* $(q_1\ \mathsf{UNION}\ q_2\ \mathsf{UNION}\ \dots\ \mathsf{UNION}\ q_n)$ *is Web-safe if each subquery* q_i $(1 \leq i \leq n)$ *is Web-safe.*

Proof. Assume each subquery q_i is Web-safe. Hence, for each q_i, there exists an algorithm that satisfies the conditions in Definition 7. Then, the Web-safeness of LDQL query $(q_1\ \mathsf{UNION}\ q_2\ \mathsf{UNION}\ \dots\ \mathsf{UNION}\ q_n)$ is easily shown by specifying another algorithm that calls the algorithms of the subqueries sequentially and unions their results.

By Proposition 1, we can show that an LDQL query in UNION normal form is Web-safe by showing that each of its UNION-free subqueries is Web-safe. Therefore, in the remainder of this section we focus the Web-safeness of UNION-free LDQL queries.

Our discussion of the (UNION-free) query $q''_{\mathsf{ex}} = (q_{\mathsf{ex}}\ \mathsf{AND}\ q'_{\mathsf{ex}})$ in Example 9 suggests that UNION -free LDQL queries can be shown to be Web-safe if it is possible to execute any non-Web-safe subquery by using variable bindings obtained from other subqueries. A necessary condition for such an execution strategy is that the variable in question (i.e., variable $?x$ in the case of non-Web-safe subquery $q_{\mathsf{ex}} = \langle ?x, \varepsilon, \langle ?x, p_1, ?z\rangle\rangle$) is guaranteed to be bound in every possible solution mapping obtained from the other subqueries.

To allow for an automated verification of this condition we adopt Buil-Aranda et al.'s notion of strongly bound variables [5].[1] To this end, for any SPARQL graph pattern P, let $\mathrm{sbvars}(P)$ denote the set of strongly bound variables in P as defined by Buil-Aranda et al. [5]. For the sake of space, we do not repeat the definition here. However, we emphasize that $\mathrm{sbvars}(P)$ can be constructed recursively, and each variable in $\mathrm{sbvars}(P)$ is guaranteed to be bound in every possible solution for P [5, Proposition 1]. To carry over these properties to LDQL queries, we use the notion of strongly bound variables in SPARQL patterns to define the following notion of strongly bound variables in LDQL queries; thereafter, in Lemma 2, we show the desired boundedness guarantee.

Definition 8. The set of **strongly bound variables** in a LDQL query q, denoted by $\mathrm{sbvars}(q)$, is defined recursively as follows:

1. If q is of the form $\langle u, \mathit{lpe}, P\rangle$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(P)$.
2. If q is of the form $\langle ?v, \mathit{lpe}, P\rangle$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(P) \cup \{?v\}$.
3. If q is of the form $(q_1\ \mathsf{AND}\ q_2)$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(q_1) \cup \mathrm{sbvars}(q_2)$.
4. If q is of the form $(q_1\ \mathsf{UNION}\ q_2)$, then $\mathrm{sbvars}(q) = \mathrm{sbvars}(q_1) \cap \mathrm{sbvars}(q_2)$.

Lemma 2. *Let q be an LDQL query. For every Web of Linked Data W and every solution mapping $\mu \in [\![q]\!]_W$, it holds that $\mathrm{sbvars}(q) \subseteq \mathrm{dom}(\mu)$.*

Proof. Lemma 2 follows trivially from Definition 8 and [5, Proposition 1].

We are now ready to show the following result.

Theorem 1. *A UNION-free LDQL query $(q_1\ \mathsf{AND}\ q_2\ \mathsf{AND}\ \dots\ \mathsf{AND}\ q_m)$ is Web-safe if there exists a total order $\prec$ over the set of subqueries $\{q_1, q_2, \dots, q_m\}$ such that for each subquery q_i ($1 \leq i \leq m$), it holds that either (i) q_i is a basic LDQL query or (ii) q_i is of the form $\langle ?v, \mathit{lpe}, P\rangle$ and $?v \in \bigcup_{q_j \prec q_i} \mathrm{sbvars}(q_j)$.*

Proof (Sketch). We prove Theorem 1 based on an iterative algorithm that generalizes the execution strategy outlined for query q''_{ex} in Example 9. That is, the algorithm executes the subqueries $q_1, q_2, \dots, q_m$ sequentially in the order $\prec$ such that each iteration step executes one of the subqueries by using the solution mappings computed during the previous step. By the conditions in Theorem 1, the first subquery (according to $\prec$)

[1] While we may also adopt Buil-Aranda et al.'s notion of bound variables (not to be confused with their notion of strongly bound variables), a definition of bound variables in LDQL queries would be based directly on the boundedness of variables in SPARQL patterns. Then, it is not difficult to see that the undecidability of verifying whether a given variable is bound in a given SPARQL pattern [5] would also carry over to LDQL queries. Due to space limitations, we omit discussing boundedness and use directly the decidable alternative (i.e., strong boundedness).

cannot be of the form $\langle ?v, lpe, P\rangle$; hence, the first step of the iteration is guaranteed to execute a basic LDQL query. This execution resembles the execution strategy as outlined for the (basic) query q'_{ex} in Example 9. For any subsequent iteration step, if the subquery executed by that step is of the form $\langle ?v, lpe, P\rangle$, then the corresponding condition in Theorem 1 (in conjunction with Lemma 2) guarantees that $?v \in \mathrm{dom}(\mu)$ for every solution mapping μ obtained from the previous step. These mappings are finitely many (for a finite Web of Linked Data). Then, the algorithm uses the URIs bound to variable $?v$ in these mappings as the only possible context URIs for the execution of the subquery (instead of using all URIs), which is sufficient because solution mappings computed for the subquery by using some other context URI would not be join compatible with any of the solution mappings obtained from the previous step. For a full formal proof and the complete algorithm we refer to the extended version of this paper [13].

To recapitulate, we summarize our proposed procedure to test a given LDQL query q for Web-safeness based on our results in this paper: First, by using the algebraic properties in Lemma 1, the query has to be rewritten into a semantically equivalent LDQL query $q_{\mathsf{nf}} = (q_1 \ \mathsf{UNION}\ q_2 \ \mathsf{UNION}\ \ldots\ \mathsf{UNION}\ q_n)$ that is in UNION normal form, which is possible for any LDQL query (cf. Corollary 1). Thereafter, the following Web-safeness test has to repeated for every subquery q_i $(1 \leq i \leq n)$; recall that each of these subqueries is a UNION-free LDQL query $q_i = (q^i_1 \ \mathsf{AND}\ q^i_2 \ \mathsf{AND}\ \ldots\ \mathsf{AND}\ q^i_{m_i})$. The test is to find an order for their subqueries $q^i_1, q^i_2, \ldots, q^i_{m_i}$ that satisfies the conditions in Theorem 1. Every top-level subquery q_i $(1 \leq i \leq n)$ for which such an order exists, is Web-safe (cf. Theorem 1). If all top-level subqueries are identified to be Web-safe by this test, then q_{nf} is Web-safe (cf. Proposition 1), and so is q (cf. Fact 1).

We conclude the section by pointing out the following limitation of our results: Even if the given conditions are sufficient to show that an LDQL query is Web-safe, they are not sufficient for showing the opposite. It remains an open question whether there exists a (decidable) property of all Web-safe LDQL queries that is sufficient *and* necessary.

5 Concluding Remarks and Future Work

LDQL, the query language that we introduce in this paper, allows users to express queries over Linked Data on the WWW. We defined LDQL such that navigational features for selecting the query-relevant documents on the Web are separate from patterns that are meant to be evaluated over the data in the selected documents. This separation distinguishes LDQL from other approaches to express queries over Linked Data. For instance, neither any of the existing SPARQL-based approaches [4,10,11,14,16] nor NautiLOD [8] can be used to express queries such as our example queries q'_{ex} and q''_{ex}.

Given that our analysis of LDQL in this paper focuses primarily on Web-safeness, in future work the language may be studied with respect to the complexity of query evaluation and the problem of query containment. A formal study of the expressive power of LDQL would also be interesting. A more practical direction for future research on LDQL is the development of approaches to execute LDQL queries efficiently.

References

1. M. Arenas, C. Gutierrez, and J. Pérez. On the Semantics of SPARQL. In *Semantic Web Information Management - A Model-Based Perspective*, chapter 13. Springer, 2009.
2. T. Berners-Lee. Linked Data. Online at http://www.w3.org/DesignIssues/LinkedData.html, 2006.
3. C. Bizer, T. Heath, and T. Berners-Lee. Linked Data – The Story So Far. *Semantic Web and Information Systems*, 5(3):1–22, 2009.
4. P. Bouquet, C. Ghidini, and L. Serafini. Querying The Web Of Data: A Formal Approach. In *Proceedings of the 4th Asian Semantic Web Conference*, 2009.
5. C. Buil-Aranda, M. Arenas, and O. Corcho. Semantics and Optimization of the SPARQL 1.1 Federation Extension. In *Proceedings of the 8th Extended Semantic Web Conference*, 2011.
6. R. Cyganiak, D. Wood, M. Lanthaler, G. Klyne, J. J. Carroll, and B. McBride. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, Online at http://www.w3.org/TR/rdf11-concepts/, Feb. 2014.
7. R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, Online at http://tools.ietf.org/html/rfc2616, June 1999.
8. V. Fionda, G. Pirrò, and C. Gutierrez. NautiLOD: A Formal Language for the Web of Data Graph. *ACM Transactions on the Web*, 9(1):1–43, 2015.
9. S. Harris, A. Seaborne, and E. Prud'hommeaux. SPARQL 1.1 Query Language. W3C Recommendation, Online at http://www.w3.org/TR/sparql11-query/, Mar. 2013.
10. A. Harth and S. Speiser. On Completeness Classes for Query Evaluation on Linked Data. In *Proceedings of the 26th AAAI Conference*, 2012.
11. O. Hartig. SPARQL for a Web of Linked Data: Semantics and Computability. In *Proceedings of the 9th Extended Semantic Web Conference*, 2012.
12. O. Hartig. An Overview on Execution Strategies for Linked Data Queries. *Datenbank-Spektrum*, 13(2), 2013.
13. O. Hartig. LDQL: A Language for Linked Data Queries (Extended Version). Online at http://olafhartig.de/files/LDQL-ext.pdf, 2015.
14. O. Hartig and G. Pirrò. A Context-Based Semantics for SPARQL Property Paths over the Web. In *Proceedings of the 12th Extended Semantic Web Conference*, 2015.
15. J. Pérez, M. Arenas, and C. Gutierrez. Semantics and Complexity of SPARQL. *ACM Transactions on Database Systems*, 34, 2009.
16. J. Umbrich, A. Hogan, A. Polleres, and S. Decker. Link Traversal Querying for a Diverse Web of Data. *Semantic Web Journal*, 2014.

Intuitionistic Data Exchange

Gösta Grahne, Ali Moallemi, and Adrian Onet

Concordia University, Montreal, Canada
grahne@cs.concordia.ca,moa_ali@encs.concordia.ca,adrian_onet@yahoo.com

Abstract. The field of Data Exchange has overcome many obstacles, but when it comes to negation in rule bodies combined with existentially quantified variables in rule heads, to unequalities $\neq$, as well as to inconsistency management, the same intractability barriers that plague the area of incomplete information arise.
In this paper we develop an intuitionistic relevance-logic based semantics that allows us to extend the polynomial time "naive evaluation" technique to Data Exchange dependencies (TGDs and EGDs) with negated atoms and unequalities, and to Data Exchange Target Queries consisting of unions of conjunctive queries with negation and unequalities. The semantics is also paraconsistent, and avoids the intractability barriers encountered in inconsistency management as well.

1 Introduction

The problem of *data exchange* poses one of the major challenges in distributed information processing environments. A connection in such an environment can be viewed as a labeled directed edge from a node representing a *source database* to a node representing a *target database*. The edge-label denotes a *schema mapping* that guides the middle-ware in restructuring the data from the source database to fit the requirements of the target database. Since its inception in 2003 by Fagin et al. in [8], the field of data exchange has been intensely investigated, and many functionalities are mature for technology transfer. In this paper we focus on the problems of

- computing and materializing target instances when the schema mapping is expressed as a set of embedded dependencies (TGDs and EGDs), and
- computing certain answers to unions of conjunctive queries expressed on the target schema (target UCQs).

These problems have been shown to admit efficient implementation, based on a property colloquially called the *naive evaluation property*. The property roughly says that the incompleteness of some domain values can be ignored, as long as these values are distinguished from each other, and from the "ordinary" domain values that denote named and known objects. The price to pay is that we have to restrict the schema-mappings and target queries to be monotone. Most attempts to include non-monotone features, such as negation ($\neg$) and unequality

($\neq$), soon run into intractability barriers, due to the underlying issues of incomplete information. For a comparison between different closed world semantics the reader should consult [22].

We recall (see [1]) that a *tuple generating dependency* (TGD) is a first order formula of the form $\forall \mathbf{XY} \left(\Phi(\mathbf{XY}) \to \exists \mathbf{Z}\, \Psi(\mathbf{XZ}) \right)$, where Φ and Ψ are conjunctions of relational atoms, and $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$ are sequences of variables. We refer to Φ as the *body* of the dependency, and to Ψ as the *head*. In an *equality generating dependency* (EGD), there is no existential quantification, and Ψ is an equality $X_1 = X_2$, where X_1 and X_2 are variables from $\mathbf{XY}$.

When it comes to dependencies with negation in the body, the main proposals are the *stratified model* the *stable model* and the *well-founded model* (for definitions, see [1]). We argue that these semantics are not well suited for data exchange, mainly because they "favor" negative information. Sometimes this is desirable, as for example in the classical *Tweety*-example:

$$\forall X \left(BIRD(X), \neg PENGUIN(X) \to FLY(X) \right).$$

If the source database is $\{BIRD(\texttt{tweety})\}$ it might be desirable to "defeasibly" conclude $FLY(\texttt{tweety})$. However, if we use the same semantics for

$$\forall X \left(COUNTRY(X), \neg NUKES(X) \to FRIEND(X) \right),$$

and the source database is $\{COUNTRY\ (\texttt{sylvania})\}$, it might not be prudent to conclude $FRIEND(\texttt{sylvania})$.

Intuitionistic logic rejects the law of the excluded middle and non-constructive existence proofs. The motivation is usually epistemological, but it can also be computational, as in Kleene's intuitionistic logic based on recursive predicates [20]. In an intuitionistic approach we would assign both $FLY(\texttt{tweety})$ and $FRIEND(\texttt{sylvania})$ the value *unknown* or *undetermined*. This does not preclude the truth-value to later change, as was for example the case with (the knowledge of) the truth-value of the statement expressing Fermat's Last Theorem, that changed from *unknown* to *true* in 1994.

If non-constructive existence proofs are epistemologically susceptible, what can one say about defeasible reasoning? We contend that negative facts should be explicitly derived, just as the positive ones are. In other words, only if we have explicitly decided the fact $\neg PENGUIN(\texttt{tweety})$ can we draw the conclusion $FLY(\texttt{tweety})$. In this spirit we propose to use Nuel Belnap's four-valued relevance logic $\mathbf{R}$ [5] as foundation for negation in data exchange. This allows us to derive efficient algorithms for:

– computing and materializing target instances when the schema mapping is expressed as a set of TGDs with negated atoms in the body and in the head, and EGDs with negation in the body, and
– computing certain answers to target unions of conjunctive queries with negation and unequality ($\neq$).

Belnap's logic also is *paraconsistent*, meaning that "if we have inconsistent information about ducks, it is possible that our information about decimals can still be trusted" (see Fitting [10], page 10). This feature allows efficient inconsistency management, which is not in general possible in the *repair*-approach of [3].

The rest of this paper is organized as follows. The next section introduces the notions used throughout the paper. This is followed by the section dedicated to Belnap's Logic $\mathcal{FOUR}$ that plays a central role in this paper. Section 4 describes the "naive evaluation" of conjunctive queries with negation, and Section 5 adds negation to the bodies of tuple generating dependencies, in a way that allows us to extend the "naive chase" (i.e. the standard chase [8]). In Section 6 we study how our techniques affect the termination of the chase procedure. We end with conclusions in Section 7.

2 Preliminaries

For basic definitions and concepts we refer to [1]. Let $\mathbf{a} = a_1, a_2, \ldots, a_n$ and $\mathbf{b} = b_1, b_2, \ldots, b_m$ be *sequences* of elements from a semigroup (a set with a binary associative operation $\cdot$). Then $\mathbf{ab}$ denotes the *concatenation sequence* $a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m$, and if $n = m$ we can form the *product sequence* $\mathbf{a} \cdot \mathbf{b} = a_1 \cdot b_1, a_2 \cdot b_2, \ldots, a_n \cdot b_n$. The *length* of a sequence $\mathbf{a} = a_1, a_2, \ldots, a_n$, denoted $|\mathbf{a}|$, is n. The *empty sequence* is denoted ϵ. Let $\mathbf{X} = X_1, X_2, \ldots, X_n$ be a sequence and A a set. Then a mapping $f : \mathbf{X} \to A^n$, where $f(X_i) = a_i$, is listed as $f = \{X_1/a_1, X_2/a_2, \ldots, X_n/a_n\}$.

Signatures, schemas, languages etc. A *signature* σ consists of a finite set $\mathsf{R} = \{R, S, \ldots\}$ of *relation symbols* (called the *schema*), a countably infinite set $\mathsf{Vars} = \{X, Y, Z, \ldots\}$ of *variables*, a countably infinite set $\mathsf{Nulls} = \{x, y, z, \ldots\}$ of *nulls*, and a countably infinite set $\mathsf{Cons} = \{a, b, c, \ldots\}$ of *constants.* We assume that Nulls and Cons are semigroups under the $\cdot$ operation. Each *relational symbol* $R \in \mathsf{R}$ has an associated natural number $ar(R)$, called the *arity* of R.

Let σ be a signature. The set of *well-formed formulas (wff's)* of the language $\mathcal{L}_\sigma$ is defined inductively by stating that each atom $R(\mathbf{X})$, where $R \in \mathsf{R}$ and $\mathbf{X} \in (\mathsf{Vars} \cup \mathsf{Cons})^{ar(R)}$, is a wff, and then closing the set under $\vee, \wedge, \neg, \exists$, and $\forall$. If $\Phi(\mathbf{X})$ is a $\mathcal{L}_\sigma$-formula, then $vars(\Phi)$ denotes its variables, the sequence $\mathbf{X}$ denotes the free variables, and $cons(\Phi)$ the constants in Φ. If $\Phi(\mathbf{X})$ is a formula and v is the mapping $\mathbf{X} \mapsto \mathbf{a}$, then $\Phi(\mathbf{a})$ denotes the sentence $\Phi(v(\mathbf{X}))$. If a wff Φ does not have any free variables, it is called a *sentence.*

Boolean valued instances. In algebraic semantics (see e.g. [23]) an interpretation of sentences in the object language is obtained through a homomorphism h into a structure $(\mathbf{A}, \varphi)$, where $\mathbf{A}$ is an algebra, and φ is a subset of the carrier set A of $\mathbf{A}$. The elements in φ represent "truth." The homomorphism maps atomic sentences to elements in A, and the connectives and quantifiers are mapped into operators in the algebra. Under interpretation h, a sentence Φ is then "true" in $\mathbf{A}$, if $h(\Phi) \in \varphi$.

The well known *Boolean valued* models [19] are obtained by taking the algebra to be a Boolean algebra $\mathbf{B} = (B, \vee, \wedge, \neg, 0, 1)$, and φ to be an ultrafilter of $\mathbf{B}$. Recall that an ultrafilter is a subset φ of B, such that $1 \in \varphi$; for all $b_1, b_2 \in B$, $b_1 \wedge b_2 \in \varphi$ iff $b_1 \in \varphi$ and $b_2 \in \varphi$; and for all $b \in B$ either $b \in \varphi$ or $\neg b \in \varphi$, but $\{b, \neg b\} \notin \varphi$. Adapting this approach to database instances (models), we get the following definitions.

Definition 1. *Let* $\sigma = (\mathsf{R}, \mathsf{Vars}, \mathsf{Nulls}, \mathsf{Cons})$ *be a database signature, and let* $\mathbf{B} = (B, \vee, \wedge, \neg, 0, 1)$ *be a Boolean algebra. A* $\mathbf{B}$-instance $\mathcal{I}$ *consists of the following parts:*

1. *A finite set of elements* $dom(\mathcal{I}) \subseteq \mathsf{Cons} \cup \mathsf{Nulls}$*, the* domain *of* $\mathcal{I}$*,*
2. $cons(\mathcal{I}) = dom(\mathcal{I}) \cap \mathsf{Cons}$*, the* constants *of* $\mathcal{I}$*,*
3. *The set* $nulls(\mathcal{I}) = dom(\mathcal{I}) \cap \mathsf{Nulls}$*, the* nulls *of* $\mathcal{I}$*, and*
4. *For each* $R \in \mathbf{R}$ *and* $\mathbf{a} \in dom(\mathcal{I})^{ar(R)}$*, an* interpretation $R(\mathbf{a})^{\mathcal{I}} \in B$.

Definition 2. *Let* Φ *be an* $\mathcal{L}$*-sentence,* $\mathbf{B}$ *a Boolean algebra,* $\varphi \subset B$ *an ultrafilter in* $\mathbf{B}$*, and* $\mathcal{I}$ *a* $\mathbf{B}$*-interpretation. Then the* satisfaction *relation* $\mathcal{I} \vDash_{\mathbf{B},\varphi} \Phi$ *is defined inductively as follows.*

1. $\mathcal{I} \vDash_{\mathbf{B},\varphi} R(\mathbf{a})$*, if* $R(\mathbf{a})^{\mathcal{I}} \in \varphi$*,*
2. $\mathcal{I} \vDash_{\mathbf{B},\varphi} \Phi \wedge \Psi$*, if* $\Phi^{\mathcal{I}} \wedge \Psi^{\mathcal{I}} \in \varphi$*,*
3. $\mathcal{I} \vDash_{\mathbf{B},\varphi} \neg\Phi$*, if* $\neg(\Phi^{\mathcal{I}}) \in \varphi$*,*
4. $\mathcal{I} \vDash_{\mathbf{B},\varphi} \forall \mathbf{X}\Phi(\mathbf{X})$*, if* $\bigwedge_{\mathbf{a} \in dom(\mathcal{I})^{ar(R)}} (\Phi(\mathbf{a})^{\mathcal{I}}) \in \varphi$*,*
5. $\mathcal{I} \vDash_{\mathbf{B},\varphi} \Phi \to \Psi$*, if* $\Phi^{\mathcal{I}} \in \varphi$ *entails that* $\Psi^{\mathcal{I}} \in \varphi$*.*

The *natural partial order* $\leq$ in $\mathbf{B}$ is defined as $\boldsymbol{\alpha} \leq \boldsymbol{\beta}$ iff $\boldsymbol{\alpha} \vee \boldsymbol{\beta} = \boldsymbol{\beta}$. The partial order can be lifted to $(\mathbf{B}, \varphi)$-instances $\mathcal{I}$ and $\mathcal{J}$ by stipulating that $\mathcal{I} \leq \mathcal{J}$ if $cons(\mathcal{I}) \subseteq cons(\mathcal{J})$, and there is a mapping $h : dom(\mathcal{I}) \to dom(\mathcal{J})$, identity on the constants, such that $R(\mathbf{a})^{\mathcal{I}} \leq R(h(\mathbf{a}))^{\mathcal{J}}$, for all $R \in \mathbf{R}$ and $\mathbf{a} \in dom(\mathcal{I})^{ar(R)}$. Such a mapping h is said to be a *homomorphism* from $\mathcal{I}$ to $\mathcal{J}$. It is easily shown that $\leq$ is a partial order on the equivalence classes generated by the relation $\mathcal{I} \simeq \mathcal{J}$, defined to hold iff $\mathcal{I} \leq \mathcal{J}$ and $\mathcal{J} \leq \mathcal{I}$.

Going back to the satisfaction relation $\vDash_{\mathbf{B},\varphi}$, note that if $B = \{0, 1\}$, then $\mathbf{B} = \mathbf{2}$ (the two-element Boolean algebra), and the unique ultrafilter is $\{1\}$. Thus we can write simply $\vDash_{\mathbf{2}}$, and $\mathcal{I} \vDash_{\mathbf{2}} \Phi$ becomes the standard two-valued semantics in which $\to$ behaves as material implication. Thus $\mathcal{I} \vDash_{\mathbf{2}} \Phi \to \Psi$ if and only if $\Phi^{\mathcal{I}} \leq \Psi^{\mathcal{I}}$. Here $\leq$ is the natural order, i.e. $0 \leq 1$, and $\mathcal{I} \leq \mathcal{J}$ means that there is a classical database homomorphism [1] from $\mathcal{I}$ to $\mathcal{J}$, and $\mathcal{I} \simeq \mathcal{J}$ means that $\mathcal{I}$ and $\mathcal{J}$ are homomorphically equivalent (in the classical database sense).

Conjunctive queries and certain answers. A *conjunctive query (CQ)* is a first order formula of the form $q = \exists \mathbf{Y}\Phi(\mathbf{XY})$, where Φ is a conjunction of atoms of $\mathcal{L}_\sigma$. For an atom R of arity k in Φ, for notational convenience we write $R(\mathbf{XY})$ instead of $R(x_{i_1}, \ldots, x_{i_k})$, where $x_{i_1}, \ldots, x_{i_k}$ is a subsequence of $\mathbf{XY}$. The free variables of Φ are those in $\mathbf{X}$. If $\mathbf{X} = \epsilon$, the expression denotes a *Boolean* CQ. A query q is called a *union of conjunctive queries (UCQ)* if it is of the form $q = \exists \mathbf{Y}(\Phi_1(\mathbf{XY}) \vee \ldots \vee \Phi_m(\mathbf{XY}))$, where each Φ_i is a conjunction of atoms. It is a well known fact that UCQs are monotonic in the $0 \leq 1$ partial order, meaning that if $\mathcal{I} \leq \mathcal{J}$, then $q(\mathcal{I}) \leq q(\mathcal{J})$, for all UCQs q. It has further been shown in [14] that $\mathcal{I} \simeq \mathcal{J}$ if and only if $q(\mathcal{I}) \simeq q(\mathcal{J})$, for all UCQs q. Thus, if a user only has UCQs as query language, then the user cannot distinguish between $\simeq$-equivalent (homomorphically equivalent) instances. This leads to the important notion of *incomplete instances* and *certain answers*, instrumental in data exchange [8]. Conceptually, an *incomplete database instance* is a set $\mathcal{X}$ of ordinary instances $\mathcal{I}$. The information that is certain is the information that holds in *all* possible instances (possible worlds). This leads to the definition of the *certain answer* to a UCQ q on an incomplete instance $\mathcal{X}$ as $\bigwedge q(\mathcal{X}) = \bigwedge\{q(\mathcal{I}) : \mathcal{I} \in \mathcal{X}\}$. Here $\bigwedge$ denotes the $\leq$-greatest lower-bound (modulo $\simeq$) of a set of ordinary instances. Likewise, $\bigvee$ denotes the $\leq$-least upper-bound (modulo $\simeq$). The least upper-bound is clearly given by the disjoint (rename nulls apart) union. The lower bound $\bigwedge$ has been shown to exist by Hell and Nesetril [16], whose results were generalized by Libkin in [21]. In particular, Libkin showed that if $\mathcal{X}$ is a finite set of instances, then $\bigwedge(\mathcal{X})$ can always be represented as a finite instance (up to $\simeq$-equivalence).

Usually the certain answer is further restricted to only the named domain values (the constants). For an instance $\mathcal{I}$, define $\mathcal{I}\!\upharpoonright$ as function $\mathcal{I}$ restricted to sequences of objects in $cons(\mathcal{I})$. Thus, the usual certain answer is

$$Cert(q, \mathcal{X}) = \left(\bigwedge\{q(\mathcal{I}) : \mathcal{I} \in \mathcal{X}\}\right)\!\upharpoonright. \tag{1}$$

Note that $q(\mathcal{X})$ is what Libkin [21] calls *answers as knowledge*, while (1) is his *answers as objects*. Since an instance $\mathcal{I}$ can contain nulls, it can be viewed as a "naive table" [18], that represents the set of complete (ground) instances $\mathcal{X}_{\mathcal{I}} = \{\mathcal{J} : \mathcal{I} \leq \mathcal{J} \text{ and } dom(\mathcal{J}) \subseteq \mathsf{Cons}\}$. Consider now an instance $\mathcal{I}$ as a naive table representing the set $\mathcal{X}_{\mathcal{I}}$ of possible worlds. Then the "naive evaluation theorem" says that $\bigwedge q(\mathcal{X}_{\mathcal{I}}) \simeq q(\mathcal{I})$. Consequently for a UCQ q, we have [18, 8]:

$$Cert(q, \mathcal{X}_{\mathcal{I}}) = \Big(q(\mathcal{I})\Big)\!\upharpoonright. \tag{2}$$

3 Belnap's logic $\mathcal{FOUR}$

We are interested in Belnap's intuitionistic logic $\mathcal{FOUR}$. This logic has a sound and complete axiomatization in terms of a system **R** of relevance logic, and $\Phi \rightarrow \Psi$ becomes relevant entailment of first degree (see [2, 5]).

The logic $\mathcal{FOUR}$ is a generalization of Kleene's strong three-valued logic. $\mathcal{FOUR}$ has four truth-values $\varnothing, \{\mathbf{f}\}, \{\mathbf{t}\}$, and $\{\mathbf{f}, \mathbf{t}\}$. Intuitively, assigning a sentence Φ the value $\{\mathbf{t}\}$ means that "the computer has been told that Φ is true," (see [5], page 11) and assigning the value $\{\mathbf{f}\}$ means that "the computer has been told that Φ is false." Furthermore, the value $\{\mathbf{f}, \mathbf{t}\}$ means that "the computer has been told that Φ is false, and the computer has been told that Φ is true." In other words, Φ is *inconsistent*. Finally, assigning Φ the value $\varnothing$ means that "the computer has not been told anything about the truth-value of Φ," that is, the truth-value of Φ is *unknown*.

Since truth-values are sets, they are naturally ordered by set-inclusion. The more elements the set contains, the more information the computer has about the truth. This gives rise to the *information ordering*, which we in the sequel will denote $\leqq$. We can still retain the $\leq$-order, which is called the *truth order*, by stipulating that $\{\mathbf{f}\} \leq \{\mathbf{t}\}$, and that $\varnothing$ and $\{\mathbf{f}, \mathbf{t}\}$ are incomparable and both lie strictly between $\{\mathbf{f}\}$ and $\{\mathbf{t}\}$. We shall use the following symbols: $\mathtt{0}$ for $\{\mathbf{f}\}$, $\mathtt{1}$ for $\{\mathbf{t}\}$, $\bot$ for $\varnothing$, and $\top$ for $\{\mathbf{f}, \mathbf{t}\}$. Furthermore, $\top$ and $\bot$ are incomparable wrt $\leq$, and $\mathtt{0}$ and $\mathtt{1}$ are incomparable in $\leqq$. All of this can be put together into a structure $\mathbf{4} = (4, \vee, \wedge, \curlyvee, \curlywedge, \neg, \mathtt{0}, \mathtt{1}, \bot, \top)$, where $4 = \{\bot, \top, \mathtt{0}, \mathtt{1}\}$. Here $\vee$ is the least upper bound in the $\leq$-order, and $\wedge$ the greatest lower bound (i.e. $\top \vee \bot = \mathtt{1}$, and $\top \wedge \bot = \mathtt{0}$, etc). The least upper bound in the $\leqq$-order is $\curlyvee$, and the greatest lower bound is $\curlywedge$ (e.g. $\mathtt{0} \curlyvee \mathtt{1} = \top$, and $\mathtt{0} \curlywedge \mathtt{1} = \bot$, etc). For negation, $\neg\,\mathtt{1} = \mathtt{0}, \neg\,\mathtt{0} = \mathtt{1}, \neg\,\top = \top$, and $\neg\,\bot = \bot$. Consequently, $(4, \vee, \wedge, \neg, \mathtt{0}, \mathtt{1})$ is a Boolean algebra.

Belnap's structure $\mathbf{4}$ has subsequently been generalized by Ginsberg and Fitting to so called *bilattices* [12, 9]. The simplest non-trivial bilattice is $\mathbf{4}$, just as $\mathbf{2}$ is the simplest non-trivial Boolean algebra. Bilattices have been thoroughly investigated by Fitting and Avron, among others (see e.g. [9, **?**,**?**,**?**]). The algebra $\mathbf{4}$ has been shown to provide a principled and uniform account of various non-monotonic semantics of logic programming with negation, including the stable models and the well-founded model.

Since $\mathbf{4}$ is an algebra, it can be used to give meaning to $\mathcal{L}_\sigma$ sentences, by choosing the designated values φ. Arieli and Avron [4] have argued that φ should be a *ultrabifilter*[1] and they have shown that $\varphi = \{\top, \mathtt{1}\}$ is the unique ultrabifilter in $\mathbf{4}$. We shall thus work with the algebraic semantics $(\mathbf{4}, \{\top, \mathtt{1}\})$.

A $\mathbf{4}$-instance is then a $\mathbf{B}$-interpretation of Definition 1, where $\mathbf{B} = \mathbf{4}$, and by noticing that $(4, \vee, \wedge, \neg, \mathtt{0}, \mathtt{1})$ is a Boolean algebra, $\mathcal{I} \vDash_{\mathbf{4}} \Psi$ can be read from Definition 2 of $\vDash_{(\mathbf{B},\varphi)}$, by choosing $\mathbf{B} = \mathbf{4}$, and $\varphi = \{\top, \mathtt{1}\}$.

Let $\mathcal{I}$ and $\mathcal{J}$ be $\mathbf{4}$-instances, such that $cons(\mathcal{I}) \subseteq cons(\mathcal{J})$ and let h be a mapping from $dom(\mathcal{I})$ to $dom(\mathcal{J})$, such that h is identity on the constants. The mapping h is said to be a *homomorphism* from $\mathcal{I}$ to $\mathcal{J}$, provided that $R(\mathbf{a})^{\mathcal{I}} \leqq R(h(\mathbf{a}))^{\mathcal{J}}$, for all $R \in \mathsf{R}$ and $\mathbf{a} \in dom(\mathcal{I})^{ar(R)}$. The partial order $\leqq$ of $\mathbf{4}$ is lifted to $\mathbf{4}$-instances in the following definition.

Definition 3. *Let $\mathcal{I}$ and $\mathcal{J}$ be $\mathbf{4}$-instances such that $cons(\mathcal{I}) \subseteq cons(\mathcal{J})$. We say that $\mathcal{J}$ is* more informative *than $\mathcal{I}$, denoted $\mathcal{I} \leqq \mathcal{J}$, if there is a homomorphism from $\mathcal{I}$ to $\mathcal{J}$. Furthermore, if both $\mathcal{I} \leqq \mathcal{J}$ and $\mathcal{J} \leqq \mathcal{I}$, we say that $\mathcal{I}$ and $\mathcal{J}$ are* information equivalent*, and denote it $\mathcal{I} \doteq \mathcal{J}$. Let $\mathscr{I}$ denote the set of all instances over σ. Then $\mathscr{I}/_{\doteq}$ denotes the set of equivalence classes of instances induced by $\doteq$.*

Clearly $\leqq$ is a partial order on $\mathscr{I}/_{\doteq}$. Due to space limitation, the definitions for relations $\curlywedge$ and $\curlyvee$ are omitted, they can be found in the full version [13]. We need to verify that $\curlywedge$ and $\curlyvee$ indeed are the meet and join in the lattice induced by $\leqq$.

[1] For a technical definition, see [4]

Lemma 1. *Let $\mathcal{I}$ and $\mathcal{J}$ be* **4***-instances. Then $\mathcal{I} \leq \mathcal{J}$ if and only if $\mathcal{I} \curlywedge \mathcal{J} \doteq \mathcal{I}$. and $\mathcal{I} \leqq \mathcal{J}$ if and only if $\mathcal{I} \curlyvee \mathcal{J} \doteq \mathcal{J}$.*

Reducing $\mathcal{FOUR}$ to $\mathcal{TWO}$. We now losslessy represent a **4**-instance as a two-valued instance as follows.

Definition 4. *1. Let $\mathcal{I}$ be a* **4***-instance over schema* R*. The* **2** -projection *of $\mathcal{I}$, denoted $\mathcal{I}^{\downarrow\mathbf{2}}$, is a two-valued Boolean instance over schema $\bigcup_{R\in\mathsf{R}}\{R^+, R^-\}$, where*

(a) $cons(\mathcal{I}^{\downarrow\mathbf{2}}) = cons(\mathcal{I})$, $nulls(\mathcal{I}^{\downarrow\mathbf{2}}) = nulls(\mathcal{I})$, and $dom(\mathcal{I}^{\downarrow\mathbf{2}}) = dom(\mathcal{I})$.

(b) For all relations $R \in \mathsf{R}$ and $\mathbf{a} = a_1, \ldots, a_{\alpha(R)}$, where $a_i \in dom(\mathcal{I})$ for all $i \in \{1, \ldots, \alpha(R)\}$, $R^+(\mathbf{a})^{\mathcal{I}^{\downarrow\mathbf{2}}} = 1$ if $R(\mathbf{a})^{\mathcal{I}} \geq 1$, and 0 otherwise; and $R^-(\mathbf{a})^{\mathcal{I}^{\downarrow\mathbf{2}}} = 1$ if $R(\mathbf{a})^{\mathcal{I}} \geq 0$, and 0 otherwise.

2. Let $\mathcal{I}$ be a **2***-instance. Then the* **4***-instance corresponding to $\mathcal{I}$ is $\mathcal{I}^{\uparrow\mathbf{4}}$, where*

(a) $cons(\mathcal{I}^{\uparrow\mathbf{4}}) = cons(\mathcal{I})$, $nulls(\mathcal{I}^{\uparrow\mathbf{4}}) = nulls(\mathcal{I})$, and $dom(\mathcal{I}^{\uparrow\mathbf{4}}) = dom(\mathcal{I})$.

(b) For all $R \in \mathsf{R}$ and $\mathbf{a} = a_1, \ldots, a_{\alpha(R)}$, for all $i \in \{1, \ldots, \alpha(R)\}$,

$$R(\mathbf{a})^{\mathcal{I}^{\uparrow\mathbf{4}}} = \begin{cases} \top \text{ if } R^+(\mathbf{a})^{\mathcal{I}} = 1 \text{ and } R^-(\mathbf{a})^{\mathcal{I}} = 1 \\ \bot \text{ if } R^+(\mathbf{a})^{\mathcal{I}} = 0 \text{ and } R^-(\mathbf{a})^{\mathcal{I}} = 0 \\ 1 \text{ if } R^+(\mathbf{a})^{\mathcal{I}} = 1 \text{ and } R^-(\mathbf{a})^{\mathcal{I}} = 0 \\ 0 \text{ if } R^+(\mathbf{a})^{\mathcal{I}} = 0 \text{ and } R^-(\mathbf{a})^{\mathcal{I}} = 1 \end{cases}$$

4 UCQs with negation

The intuitionistic semantics $(\mathbf{4}, \{\top, 1\})$ allows us to extend the (polynomial time) "naive evaluation" from unions of conjunctive queries to unions of conjunctive queries with negation. We next define CQs with negation. UCQs with negation is defined similarly.

A *literal* is an expression of the form $R(\mathbf{X})$ or $\neg R(\mathbf{X})$, where $R(\mathbf{X})$ is an atom in $\mathcal{L}_\sigma$. A *conjunctive query with negation (NCQ)* is a first order formula of the form $q = \exists \mathbf{Y} \Phi(\mathbf{XY})$, where Φ is a conjunction of literals. Consider now an NCQ

$$q = \exists \mathbf{Y} \Big(\bigwedge_{i=1}^{n} R_i(\mathbf{XY}) \wedge \bigwedge_{i=n+1}^{n+m} \neg R_i(\mathbf{XY}) \Big). \tag{3}$$

The NCQ q, when applied on an instance $\mathcal{I}$, extends the interpretation $\mathcal{I}$ to also include the relation Q, defined as follows.

Definition 5. *Let $\mathcal{I}$ be a* **4***-instance and $q = \exists \mathbf{Y} \Phi(\mathbf{XY})$ an NCQ of the form (3). Let v range over all mappings from $\mathbf{XY}$ to $dom(\mathcal{I})$. Then*

$$Q(\mathbf{a})^{\mathcal{I}} = \bigvee_{\{v\,:\,v(\mathbf{X})=\mathbf{a}\}} \Big(\bigwedge\nolimits_{i=1}^{n} R_i(v(\mathbf{XY}))^{\mathcal{I}} \wedge \bigwedge\nolimits_{i=n+1}^{n+m} \neg \big(R_i(v(\mathbf{XY}))^{\mathcal{I}} \big) \Big),$$

for each $\mathbf{a} \in dom(\mathcal{I})^{|\mathbf{X}|}$.

We next show that NCQs are monotonic in the information order $\leq$. By a recent result of Gheerbrant et al. [11], such monotonicity is a necessary and sufficient condition for a class of queries to admit naive evaluation. Indeed, the following holds.

Theorem 1. *Let q be an NCQ, and let $\mathcal{I}$ and $\mathcal{J}$ be* **4***-instances, such that $\mathcal{I} \leq \mathcal{J}$ by homomorphism h. Then $Q(\mathbf{a})^{\mathcal{I}} \leq Q(h(\mathbf{a}))^{\mathcal{J}}$, for all $\mathbf{a} \in dom(\mathcal{I})^{ar(Q)}$.*

Furthermore, the naive evaluation property of CQs on **2**-instances can now be extended to NCQs on **4**-instances. It turns out that an NCQ q can be evaluated naively on $\mathcal{I}^{\downarrow\mathbf{2}}$ by turning q into a UCQs, denoted $q^{\downarrow\mathbf{2}}$. Assume q is of the form (3). Then

$$q^{\downarrow\mathbf{2}} = \exists\mathbf{Y}\Big(\Big(\bigwedge_{i=1}^{n} R_i^+(\mathbf{XY}) \wedge \bigwedge_{i=n+1}^{n+m} R_i^-(\mathbf{XY})\Big) \vee \bigvee_{i=1}^{n} R_i^-(\mathbf{XY}) \vee \bigvee_{j=n+1}^{m} R_j^+(\mathbf{XY})\Big). \quad (4)$$

In the next theorem we view an NCQ query q as mapping an instance $\mathcal{I}$ to the instance $q(\mathcal{I})$, where $q(\mathcal{I})$ is a **4**-instance over schema $\{Q\}$, with $Q(\mathbf{a})^{q(\mathcal{I})} = Q(\mathbf{a})^{\mathcal{I}}$.

Theorem 2. *Let q be an NCQ and $\mathcal{I}$ a* **4***-instance. Then $q(\mathcal{I}) \doteq (\mathbf{q}^{\downarrow\mathbf{2}}(\mathcal{I}^{\downarrow\mathbf{2}}))^{\uparrow\mathbf{4}}$.*

Similarly to the two-valued case, we can regard a **4**-instance $\mathcal{I}$ as a naive table defining the set $\mathcal{X}_{\mathcal{I}} = \{\mathcal{J} : \mathcal{I} \leq \mathcal{J}$ and $dom(\mathcal{J}) \subseteq \mathsf{Cons}\}$ of possible worlds. We can now extended the naive evaluation property to NCQs on **4**-instances, by using the result (proved in the full version) that $\bigwedge q(\mathcal{X}_{\mathcal{I}}) \doteq q(\mathcal{I})$. In analogue with (1) we define the *certain answer* as $Cert_{\mathbf{4}}(q, \mathcal{X}) = (\bigwedge\{q(\mathcal{I}) : \mathcal{I} \in \mathcal{X}\})\!\upharpoonright$. To wrap this section up formally, we conclude

Corollary 1. *Let q be an NCQ and $\mathcal{I}$ a* **4***-instance. Then $Cert_{\mathbf{4}}(q, \mathcal{X}_{\mathcal{I}}) = \Big((\mathbf{q}^{\downarrow\mathbf{2}}(\mathcal{I}^{\downarrow\mathbf{2}})^{\uparrow\mathbf{4}}\Big)\!\upharpoonright$.*

Note that all the previous results hold as well for the larger class NUCQ of unions of conjunctive queries with negation. We have only presented the transformation of NCQs q into UCQs $q^{\downarrow\mathbf{2}}$. In a very similar fashion we can transform any NUCQ query into a UCQ, while preserving all theorems of this section, and in particular Corollary 1.

5 Data Exchange in $\mathcal{TWO}$ and $\mathcal{FOUR}$

A *tuple generating dependency (TGD)* is a first order formula having the form $\forall\mathbf{XY}(\Phi(\mathbf{XY}) \rightarrow \exists\mathbf{Z}\ R(\mathbf{XZ}))$. An *equality generating dependency (EGD)* is a first order formula of the form $\forall\mathbf{X}(\Phi(\mathbf{X}) \rightarrow X_1 = X_2)$, where X_1 and X_2 are variables in $\mathbf{X}$. A *TGD with negation and unequality (NTGD$^{\not\approx}$)* is a TDG that can have atomic negations in the body and in the head, and that also includes a special equality symbol $\approx$, possibly negated (denoted $\not\approx$). Note that NTGD$^{\approx}$s include the EGDs. Also, the notion of a **4**-instance has to be extended to account for the explicit equality. Details appear in the full paper [13]. The salient point is that an atom $a \approx b$ can have truth-values in $\mathbf{4}$, meaning that $(a \approx b)^{\mathcal{I}}$ can be *inconsistent* or *unknown*, in addition to *true* or *false*. Let $\mathbf{n} \in \{\mathbf{2}, \mathbf{4}\}$. A dependency ξ is said to be *satisfied* in an $\mathbf{n}$-instance $\mathcal{I}$ if $\mathcal{I} \vDash_{\mathbf{n}} \xi$. A set of dependencies is denoted Σ, and $\mathcal{I} \vDash_{\mathbf{n}} \Sigma$, if $\mathcal{I} \vDash_{\mathbf{n}} \xi$ for all $\xi \in \Sigma$. The following is a central concept in data exchange.

Definition 6. *Let $\mathcal{I}$ be a $\mathbf{2}$-instance (a $\mathbf{4}$-instance) and Σ a set of TGDs and EGDs (a set of NTGD$^{\approx}$s). A* $\mathbf{2}$-universal model *(a* $\mathbf{4}$-universal model*) of $(\mathcal{I}, \Sigma)$ is a $\mathbf{2}$-instance $\mathcal{J}$ (a $\mathbf{4}$-instance $\mathcal{J}$), such that*

1. *$\mathcal{I} \leq \mathcal{J}$ $(\mathcal{I} \lessapprox \mathcal{J})$,*
2. *$\mathcal{J} \vDash_{\mathbf{2}} \Sigma$ $(\mathcal{J} \vDash_{\mathbf{4}} \Sigma)$, and*
3. *for all instances $\mathcal{K}$, if $\mathcal{I} \leq \mathcal{K}$ and $\mathcal{K} \vDash_{\mathbf{2}} \Sigma$, then $J \leq \mathcal{K}$
(if $\mathcal{I} \lessapprox \mathcal{K}$ and $\mathcal{K} \vDash_{\mathbf{4}} \Sigma$, then $J \lessapprox \mathcal{K}$).*

Let $\mathcal{I}$ be a $\mathbf{2}$-instance and Σ a set of dependencies. The definition of the procedure $Chase(\mathcal{I}, \Sigma)$ can be found in [8, 7]. The chase procedure is said to *successfully terminate* (or simply terminate) [8], if it does not fail due to some EGD and it stops after a finite number of steps. In this case $Chase(\mathcal{I}, \Sigma)$ denotes the instance returned by the chase procedure. It is known (see [8, 7]) that $Chase(\mathcal{I}, \Sigma)$ is a $\mathbf{2}$-universal model of $(\mathcal{I}, \Sigma)$.

Data Exchange. A *data exchange schema* is a schema of the form $\mathbf{S} \cup \mathbf{T}$, where $\mathbf{S} \cap \mathbf{T} = \varnothing$. The schema $\mathbf{S}$ is called the *source schema*, and $\mathbf{T}$ is called the *target schema*. Let $\mathbf{S} \cup \mathbf{T}$ be a data exchange schema. A *source-to-target TGD (st-TGD)* is a TGD where the relation symbols in the body come from $\mathbf{S}$ and the relation symbols in the head come from $\mathbf{T}$. Source-to-target NTGD$^{\approx}$s (s-t NTGD$^{\approx}$s) are defined similarly. A *target EGD (t-EGD)* is an EGD where all relation symbols come from $\mathbf{T}$. A *data exchange setting* is a pair $(\mathbf{S} \cup \mathbf{T}, \Sigma)$, where $\mathbf{S} \cup \mathbf{T}$ is a data exchange schema, and Σ is a set of s-t TGDs (or s-t NTGD$^{\approx}$s) and t-EGDs.

A $\mathbf{2}$-*instance* of a data exchange setting is a pair $(\mathcal{I}, \Sigma)$, such that $nulls(\mathcal{I}) = \varnothing$, $R(\mathbf{a})^{\mathcal{I}} = 0$, for all $R \in \mathbf{T}$ and $\mathbf{a} \in dom(\mathcal{I})^{ar(R)}$, and $\Sigma = \Sigma_{st} \cup \Sigma_t$, where Σ_{st} is a set of s-t TGDs and Σ_t is a set of t-EGDs.

A $\mathbf{4}$-*instance* of a data exchange setting is like a $\mathbf{2}$-instance, except that Σ_{st} can contain NTGD$^{\approx}$'s, and that $R(\mathbf{a})^{\mathcal{I}} = \bot$, for all $R \in \mathbf{T}$ and $\mathbf{a} \in dom(\mathcal{I})^{ar(R)}$.

Let $\mathbf{n} \in \{\mathbf{2}, \mathbf{4}\}$ and $\preccurlyeq \in \{\leq, \lessapprox\}$. An $\mathbf{n}$-*solution* for an $\mathbf{n}$-instance $(\mathcal{I}, \Sigma)$ of a data exchange setting $(\mathbf{S} \cup \mathbf{T}, \Sigma)$, is an $\mathbf{n}$-instance $\mathcal{J}$, such that $\mathcal{I} \preccurlyeq \mathcal{J}$, $\mathcal{J} \vDash_{\mathbf{n}} \Sigma$, and $\mathcal{J}\restriction_{\mathbf{S}} = \mathcal{I}\restriction_{\mathbf{S}}$. The notation $\mathcal{J}\restriction_{\mathbf{S}}$ stands for the instance $\mathcal{J}$ restricted to the relation symbols in $\mathbf{S}$, and similarly for $\mathcal{I}\restriction_{\mathbf{S}}$. The set of all $\mathbf{n}$-solutions to an $\mathbf{n}$-instance $\mathcal{I}$ in setting $(\mathbf{S} \cup \mathbf{T}, \Sigma)$ is denoted $Sol_{\mathbf{n}}(\mathcal{I}, \Sigma)$. A $\mathbf{n}$-solution $\mathcal{J} \in Sol_{\mathbf{n}}(\mathcal{I}, \Sigma)$ is said to be an $\mathbf{n}$-*universal solution* if $\mathcal{J} \preccurlyeq \mathcal{K}$, for all $\mathcal{K} \in Sol_{\mathbf{n}}(\mathcal{I}, \Sigma)$.

Let $\mathcal{I}$ be a $\mathbf{2}$ instance of a data exchange setting $(\mathbf{S} \cup \mathbf{T}, \Sigma)$, and let $q = \exists \mathbf{Y} \Phi(\mathbf{XY})$ be a *target* UCQ, i.e. such that all relation symbols in Φ come from $\mathbf{T}$. Then the *certain answer* [8] to q on $(\mathcal{I}, \Sigma)$ is defined as

$$Cert_{\mathbf{2}}(q, \mathcal{I}, \Sigma) = \Big(\bigwedge\{q(\mathcal{J}) : \mathcal{J} \in Sol_{\mathbf{2}}(\mathcal{I}, \Sigma)\}\Big)\restriction . \tag{5}$$

In other words, $Cert_{\mathbf{2}}(q, \mathcal{I}, \Sigma)$ is the greatest lower bound in the partial order $\leq$ of the set $\{q(\mathcal{J}) : \mathcal{J} \in Sol_{\mathbf{2}}(\mathcal{I}, \Sigma)\}$, restricted to the constants. The following result is at the foundation of data exchange.

Theorem 3. [8] *Let $(\mathcal{I}, \Sigma)$ be a data exchange setting, such that the chase with Σ of $\mathcal{I}$ terminates, and let q be a target UCQ. Then*

1. *$\mathcal{J} = Chase(\mathcal{I}, \Sigma)$ is a $\mathbf{2}$-universal solution for $(\mathcal{I}, \Sigma)$.*
2. *$Cert(q, \mathcal{I}, \Sigma) = (q(\mathcal{J}))\!\upharpoonright$.*

In the full paper [13] we show that by a suitable reduction, any set Σ of NTGD$^{\approx}$s can be transformed into a set $\Sigma^{\downarrow \mathbf{2}}$ of TGDs, such that for all $\mathbf{4}$-instances $\mathcal{I}$, $\mathcal{I} \models_{\mathbf{4}} \Sigma$ if and only if $\mathcal{I}^{\downarrow \mathbf{2}} \models_{\mathbf{2}} \Sigma^{\downarrow \mathbf{2}}$. The following theorem makes the connection between the data exchange solutions, as defined in [8], and solutions for the intuitionistic data exchange problem.

Theorem 4. *Let $(\mathbf{S} \cup \mathbf{T}, \Sigma)$ be a data exchange setting where Σ consists of st-NTGD$^{\approx}$s and t-EGDs, and let $\mathcal{I}$ be a $\mathbf{4}$-instance. Then*

$$Sol_{\mathbf{4}}(\mathcal{I}, \Sigma) = Sol_{\mathbf{2}}(I^{\downarrow \mathbf{2}}, \Sigma^{\downarrow \mathbf{2}})^{\uparrow \mathbf{4}}.$$

Armed with this result we now show that we can use the chase process defined in [8] to find a $\mathbf{4}$-universal model. As the next theorem shows, a $\mathbf{4}$-universal solution is a good candidate to be materialized and used to compute the certain answer for any NUCQ-query.

Theorem 5. *Let $(\mathcal{I}, \Sigma)$ be a $\mathbf{4}$-instance of a data exchange setting, where Σ consists of st-NTGD$^{\approx}$s and t-EGDs, Then $(Chase(\mathcal{I}^{\downarrow \mathbf{2}}, \Sigma^{\downarrow \mathbf{2}}))^{\uparrow \mathbf{4}}$ is a $\mathbf{4}$-universal solution for $(\mathcal{I}, \Sigma)$. Moreover, for every NUCQ query q we have*

$$Cert_{\mathbf{4}}(q, \mathcal{I}, \Sigma) = (q^{\downarrow \mathbf{2}}((Chase(\mathcal{I}^{\downarrow \mathbf{2}}, \Sigma^{\downarrow \mathbf{2}}))))\!\upharpoonright^{\uparrow \mathbf{4}}.$$

6 On Universal Solution Existence

In the previous sections we have shown that for a given $\mathbf{4}$-instance $\mathcal{I}$ and a set of NTGDs Σ, (or NTGD$^{\neq}$s), the $\mathbf{4}$-universal solution is the best candidate for target materialization in data exchange, and that as such it can be used to efficiently compute certain answers to target UNCQ queries (or UNCQ$^{\approx}$ queries). The question then arises whether a $\mathbf{4}$-universal solution exists. Since our results include the lossless decomposition $Sol_{\mathbf{4}}(\mathcal{I}, \Sigma) = Sol_{\mathbf{2}}(\mathcal{I}^{\downarrow \mathbf{2}}, \Sigma^{\downarrow \mathbf{2}})^{\uparrow \mathbf{4}}$, the answer follows directly from from [7] and [15]. Formally, we have

Theorem 6. *1. The problem of testing if there exists a $\mathbf{4}$-universal solution for a given $\mathbf{4}$-instance and a set of NTGD$^{\approx}$s is* RE*-complete.*
2. The problem of testing if a set of NTGD$^{\approx}$s has a $\mathbf{4}$-universal solution for every $\mathbf{4}$-instance is coRE*-complete.*

The definition of the *core-chase* procedure can be found in [7], from which the next theorem follows.

Theorem 7. *The core chase procedure is complete for finding* $\mathbf{4}$*-universal solutions for* $\mathbf{4}$*-instances* $\mathcal{I}$ *and NTGD*$^{\approx}$*s*

One of the important classes for which universal solutions are guaranteed to exist, is the *guarded dependencies* of [6]. Applying the definition from [6], we say that an NTGD$^{\approx}$ is guarded if the body has an atom that contains all the universally quantified variables. It can easily be shown that if Σ is a set of guarded NTGD$^{\approx}$s, then $\Sigma^{\downarrow\mathbf{2}}$ is a set of guarded TGDs as well. From this and from the elegant result of [17] it follows that

Theorem 8. *A* $\mathbf{4}$*-universal solution always exists for a set of guarded NTGD*$^{\approx}$*s.*

In the literature one can find many classes of TGDs that are known to ensure uniform chase termination, and thus guaranteeing the existence of universal solutions for all instances (for an overview, see e.g. [22]). Let $\mathcal{C}$ be such a class. We then have

Corollary 2. *Let* Σ *be a set of NTGD*$^{\approx}$*s. If* $\Sigma^{\downarrow\mathbf{2}} \in \mathcal{C}$*, then a* $\mathbf{4}$*-universal solution for* $(\mathcal{I}, \Sigma)$ *exists, for all* $\mathbf{4}$*-instances* $\mathcal{I}$*.*

We note that most of the known classes guaranteeing uniform termination have a tractable membership problem.

7 Conclusions

In this paper we have put forth a new approach to Data Exchange based on Belnap's logic $\mathcal{FOUR}$. We showed that moving to a four-valued logic comes with the benefit of naturally extending the mapping language to NTGD$^{\approx}$s and the query language to NUCQ$^{\approx}$s, thus allowing negation over atoms, including equality atoms $a \approx b$. We also showed that well-known techniques from Data exchange (chase, universal solution, naive evaluation, etc) can be used in $\mathcal{FOUR}$ as well. Furthermore, the *core* of a $\mathbf{4}$-instance $\mathcal{I}$ (minimal instance in size that is $\simeq$ equivalent with $\mathcal{I}$) is the same as $core(I^{\downarrow\mathbf{2}})^{\uparrow\mathbf{4}}$. Thus the core chase [7] is complete in finding $\mathbf{4}$-universal solutions as well. In the special case where the instances are two-valued, the mappings are TGDs and EGDs, and the target queries are UCQs, the semantics reduces to the classical one [8].

There is still more work to be done as we haven't touched on for instance meta-data management problems, such as composition and inverse. Another aspect that needs to be further investigated is the chase termination problem in $\mathcal{TWO}$ for a set of TGDs corresponding to a set of NTGD$^{\approx}$s in $\mathcal{FOUR}$. Note that the set of TGDs in this case is special in the sense that it always contain the set of full TGDs expressed by Σ.

We believe that $\models_{\mathbf{4}}$ is a semantics with natural appeal, and one that can be practically implemented over existing DBMSs.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. Alan Anderson, Belnap R., D. Nuel, and J. Michael Dunn. *Entailment: The Logic of Relevance and Necessity, Vol. I.* Princeton University Press, 1992.
3. Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In Victor Vianu and Christos H. Papadimitriou, editors, *PODS*, pages 68–79. ACM Press, 1999.
4. Ofer Arieli and Arnon Avron. The value of the four values. *Artif. Intell.*, 102(1):97–141, 1998.
5. Jr. Belnap, Nuel D. A useful four-valued logic. In J.M̃ichael Dunn and George Epstein, editors, *Modern Uses of Multiple-Valued Logic*, volume 2 of *Episteme*, pages 5–37. Springer Netherlands, 1977.
6. Andrea Calì, Georg Gottlob, and Michael Kifer. Taming the infinite chase: Query answering under expressive relational constraints. In *KR*, pages 70–80, 2008.
7. Alin Deutsch, Alan Nash, and Jeffrey B. Remmel. The chase revisited. In *PODS*, pages 149–158, 2008.
8. Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *ICDT*, pages 207–224, 2003.
9. Melvin Fitting. Bilattices and the semantics of logic programming. *J. Log. Program.*, 11(1&2):91–116, 1991.
10. Melvin Fitting. The family of stable models. *J. Log. Program.*, 17(2/3&4):197–225, 1993.
11. Amélie Gheerbrant, Leonid Libkin, and Cristina Sirangelo. When is naive evaluation possible? In *PODS*, pages 75–86, 2013.
12. Matthew L. Ginsberg. Bilattices and modal operators. In Rohit Parikh, editor, *TARK*, pages 273–287. Morgan Kaufmann, 1990.
13. Gösta Grahne, Ali Moallemi, and Adrian Onet. Intuitionistic data exchange. Technical report, `http://arxiv.org/`.
14. Gösta Grahne and Adrian Onet. Representation systems for data exchange. In *ICDT*, pages 208–221, 2012.
15. Gösta Grahne and Adrian Onet. The data-exchange chase under the microscope. *CoRR*, abs/1407.2279, 2014.
16. Pavol Hell and Jaroslav Neŝetr̆il. *Graphs And Homomorphisms.* Oxford University Press, 2004.
17. André Hernich. Computing universal models under guarded tgds. In *ICDT*, pages 222–235, 2012.
18. Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *J. ACM*, 31(4):761–791, 1984.
19. Thomas Jech. Boolean valued models. In *Handbook of Boolean Algebras*, pages 1197–1211, 1989.
20. Stephen Cole Kleene. *Introduction to metamathematics.* D. Van Norstrand, 1952.
21. Leonid Libkin. Incomplete data: what went wrong, and how to fix it. In *PODS*, pages 1–13, 2014.
22. Adrian Onet. *The chase procedure and its applications.* PhD thesis, Concordia University, 2012.
23. Alasdair Urquhart. Basic many-valued logic. In D.M. Gabbay and F. Guenthner, editors, *Handbook of philosophical logic*, volume 2, pages 249–295. Springer Netherlands, 2001.

A preliminary investigation into SPARQL query complexity and federation in Bio2RDF

Carlos Buil-Aranda[1], Martín Ugarte[1], Marcelo Arenas[1], and Michel Dumontier[2]

[1] Department of Computer Science, Pontificia Universidad Católica, Chile
{cbuil, marenas}@ing.puc.cl
martinugarte@puc.cl
[2] Stanford Center for Biomedical Informatics Research
Stanford University, Stanford, CA, USA
michel.dumontier@stanford.edu

Abstract When users query a SPARQL endpoint, they normally face an empty text box in which they have to write the desired queries. This obstructs the process of obtaining the data they want, since users rarely have any assistance in querying a (possibly huge) RDF database. In this paper we report a deep analysis of the server log files that record the queries that users send to the SPARQL endpoints, focusing in the Bio2RDF cluster. This log analysis reveals the large number of repeated queries that users submit, and how they pursue a trial and error process by adding and removing operations from the submitted queries to obtain the desired results. We also show how users try to connect to other RDF datasets in the Linked Open Data cloud. Our results offer insight into the interaction between users and a schema-light RDF dataset, and secondly, suggest improvements to SPARQL server optimizations in terms of optimization and results caching.

1 Introduction

Querying Semantic Web data is a difficult task. Normally the databases are publicly available and they can be accessed via a web service called SPARQL endpoint. This service is made available through a web application with a single text box, allowing users to enter arbitrary SPARQL queries. That text box generally contains an example query, which is the only assistance that users have for accessing the data stored in the RDF database. That default query may point them to some useful data but most probably the results obtained will be meaningless for specific tasks. This situation is more problematic if instead of a single RDF database, users wants to access a cluster of databases like in the Bio2RDF [9,3] project. Bio2RDF is an open source project that provides over 30 biomedical datasets as Linked Data. Each dataset is made available for download and is available for querying in a dataset-specific SPARQL endpoint. In this case users not only face the difficulty of accessing the data in every dataset separately, but they also face the difficulty of combining results from several databases.

In this paper we propose to analyze the server log files from the datasets in the Bio2RDF project. This log analysis shows that there is a large amount of repeated queries, and that users follow a trial and error process, varying the complexity of the queries for obtaining the results they want. We also show how users try to connect to external RDF datasets in the Linked Open Data cloud, and we try to understand and explain the users' intentions when they query a SPARQL endpoint. The log analysis we show in this paper is driven by two goals: first to help users in obtaining useful results from a semi-unknown RDF database and second, improve the performance of SPARQL servers by looking in detail how users access them.

Related Work There have been several attempts to obtain useful research results from SPARQL endpoints query logs. Most of them have been published in the workshop series Usage Analysis and the Web of Data (USEWOD) [5,4,6,7], which is the leading initiative for encouraging research in SPARQL endpoints log analysis. These research works vary from analyzing the usage frequency of the main SPARQL operators [2], characterizing machine agents [15] or identifying browsing and query patterns by using Description Logics ontologies [10]. Further research works include a more in-detail analysis of the FILTER operator usage [1], a log analysis towards caching and pre-fetching SPARQL query results for improving performance [11], and a work that used the USEDOWD dataset to differentiate queries generated by software applications from those generated by users [16]. Other works outside the USEWOD workshop include an analysis of the SPARQL queries submitted to DBpedia [14], statistics about the access to RDF datasets in the Linked Data Cloud in 2010 [13] and a method to detect errors or weaknesses within ontologies used for Linked Data population based on statistics and network visualizations [12].

2 Log Processing

We analyzed the log files generated by the Bio2RDF servers maintained by the Dumontier Lab over an 18 month period (from May 12th 2013 until September 28th 2014). These logs included every valid HTTP GET/POST request that was received by each Bio2RDF endpoint. The total amount of requests received was 115,119,540. We first parsed these log files generating, for each request, a tuple containing the user's IP address (used as user ID), the time and date in which the HTTP request was received, the string in the HTTP request unquoted, the user agent which submitted the request, the Bio2RDF server targeted, the HTTP response code, and the size of that response. Out of the 115,119,540 valid HTTP requests received, 90,938,804 of them corresponded to SPARQL queries. This is natural since the studied servers also serve websites and further services. The queries were characterized as SELECT, ASK, CONSTRUCT and DESCRIBE queries. The next step was to remove duplicate queries. We parsed the tuples generated in the previous file and we obtained that (surprisingly) only 6,538,280 queries were unique, having thus a total of 84,400,524 repetitions. As our log study is intended to analyze the users' behavior, repetitions were only counted under the same user, meaning that the same query issued by two different users is not considered as a repetition. Next, we transformed the 6,538,280 queries into an algebraic representation, using the SPARQL Syntax Expressions[1] format from Apache Jena. This transformation facilitated a detailed analysis of the queries. For the generation of the SSE expressions

[1] https://jena.apache.org/documentation/notes/sse.html

we used the Ruby library Ruby-RDF[2], which was unable to parse 174,011 of the queries (possibly due to syntax errors). For 1,289,134 of the remaining 6,364,269, we were unable to generate the corresponding algebraic expression, in some cases due to syntax errors that were not captured by the SSE parser (e.g. not using <> for URIs). Finally this process generated a total of 4,901,124 unique queries for analysis.

3 Complexity Analysis

To gain insight into what users formulate against Bio2RDF SPARQL endpoints, we tabulated combinations of operators mentioned in the queries. We first decomposed each query into its operators, number of triple patterns, and expressions used in FILTER clauses.

Our results, summarized in Table 1, show that the most submitted query pattern is a SELECT query with a single triple pattern, scoring more than 820,000 submissions out of the 4,901,124 unique queries. The second most submitted query pattern is a CONSTRUCT query with a single triple pattern and a FILTER expression (representing more than 540,000 unique queries) followed by a DESCRIBE query (which represent about 440,000 unique queries). The three patterns mentioned above characterize a 36.8% of the analyzed queries. This shows that the general usage of SPARQL is very basic and the patterns in the body of the queries are rather simple, but users know about the language given they use distinct query forms (SELECT, CONSTRUCT and DESCRIBE). We can conclude that there is a fair understanding of the query language, but the datasets are not known by the users. Hence, they issue basic queries to gain some insight on how data is structured.

Figure 1 represents the previous situation in more detail, showing how the decreasing amount of queries submitted is directly related with the increasing amount of operators in the queries. The less common query patterns contained several operators (e.g. 3 OPTs, 3 joins and 7 triples), which is by no means surprising. We can also see how only the first 20 query patterns represent a 90% of the queries submitted, which can be an interesting fact to consider when optimizing a SPARQL endpoint. Figure 1 contains the first 50 query patterns, with labels for some of them. In total there are almost 1,000 different query pattern types. The data used to generate Figure 1 and a more detailed figure is available at https://plot.ly/%7Ecbuil/31.

4 Iterative Analysis

Next, we examined the behavior of users in terms of how they increased/decreased the amount of operators and triple patterns in SPARQL queries over time. We hypothesized that if an initial query returned a large set of results, users might then refine the query with additional SPARQL operators to reduce the result size. In contrast, if a user obtains too few results, she might generalize the query by removing some operations in order to increase the number of results. To evaluate this we defined a query complexity measure which assigns a weight of 1 to each operator and each triple pattern. For

[2] https://github.com/ruby-rdf/sparql

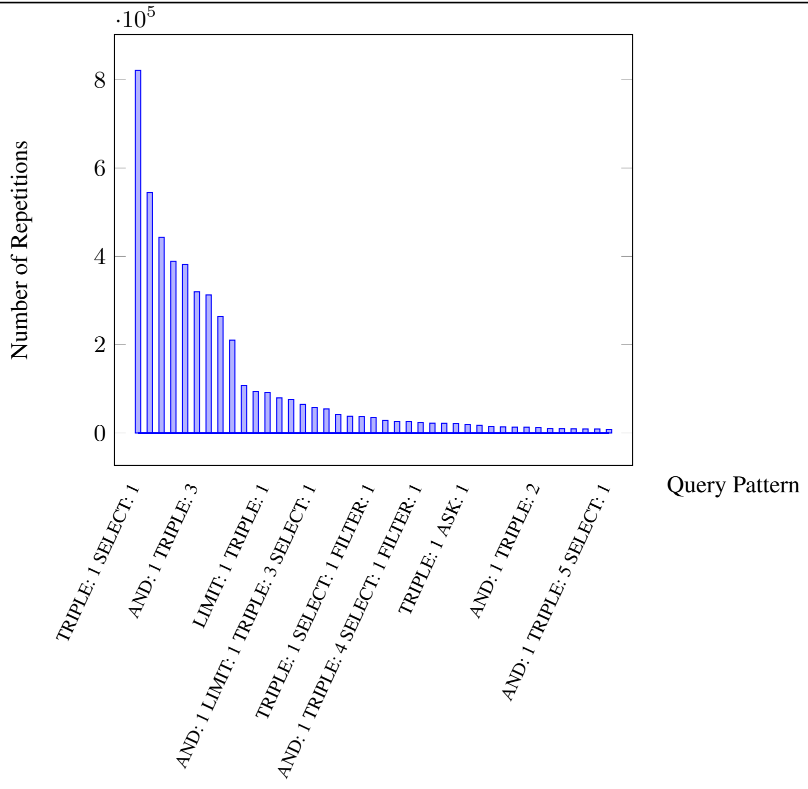


Figure 1. Query patterns ordered by number of repetitions. The most common pattern is SELECT v WHERE t, being v a set of variables (or the symbol $*$) and t a triple pattern.

instance, a query of the form DESCRIBE u (where u is a URI) has complexity 1 beacause of the DESCRIBE operator, while a SELECT query joining 3 triple patterns has a complexity of 5; 1 for the SELECT operator, 1 for the join (bgp) and 1 for each triple pattern.

We measured the HTTP request response size as a proxy for the size of the result set returned to the user. Then, we computed the number of consecutive complexity increases/decreases (referred to as a streak) for each user. Table 4 shows the amount of complexity-increasing streaks we found in the Bio2RDF log files. The first column shows the length of each streak. For example length 2 means that a user issued three queries, being the second more complex than the first and the third more complex than de second. The second column shows the times we found streaks of that length. The third column show how many of the streaks stopped when the result size of the last query was larger than the result size of the second last query. Conversely, the fourth column shows the same but when the result size of the last query was smaller than the result size of the second last query. The last two columns are intended to understand the intention of a user when he issues a streak of increasing queries: did the user stop

Table 1. Query Pattern Repetitions. The most repeated query pattern has a SELECT and a single triple, followed by a CONSTRUCT query and a DESCRIBE query.

Number of Query Pattern Repetitions	Query Pattern
821046	TRIPLE PATTERNS: 1, SELECT: 1
544341	TRIPLE PATTERNS: 1, CONSTRUCT: 1,FILTER: 1
443125	DESCRIBE 1
389011	LIMIT: 1, TRIPLE PATTERNS: 1, SELECT: 1
381351	TRIPLE PATTERNS: 1
319708	AND: 1, TRIPLE PATTERNS: 3

adding operators when he got less results? or was it when he got more results than before? Conversely, Table 4 shows the same statistics for decreasing streaks.

As opposed to what we originally hypothesis, the results show that users who add new operators in their query workflow will generally obtain more results than in their previous query, as depicted in Table 4. Similarly, users who remove operators stop removing them generally when the result size is smaller than that of the previous query (4). Our interpretation of the statistics is that users will add operators once they understand the dataset structure, and hence they will issue correct queries that will return more results. On the other side, when users issue a query with less operators they might be looking to understand new portions of the data, but in general they obtain less results due to a limited knowledge of the data structure. The only situation in which this is not the case is when users issue decreasing streaks of size 3. Here users stop removing operators once they get larger result. A preliminary interpretation of this could be that users who know the dataset are obtaining to few results, and hence they start removing restrictive parts of the query (like joins or filters) in order to obtain more information. We believe this is something worth investigating in more detail. In summary, the results show that when users add operations they generally are obtaining more results, and when users remove operations they are obtaining less results. This might have an interesting impact in terms of server optimization, as a static analysis on two consecutive queries and the answer to the first of them could already give insight on what will be the size of the result to the second query. Of course this requires a more refined definition of increasing/decreasing streaks, which is left as future work.

Table 2. Amount of *increasing* streaks, streak sizes and relation with amount of results from the previous query results.

Streak length	# of increasing streaks	Ended with larger result	Ended with smaller result
1	286,684	259,148	26,083
2	21,903	21,334	464
3	157	60	88

Table 3. Amount of *decreasing* streaks, streak sizes and relation with amount of results from the previous query results.

Streak length	# of decreasing streaks	Ended with larger result	Ended with smaller result
1	283,474	23,007	258,655
2	6,905	1,602	5,187
3	10,005	9,854	129

5 Dataset Federation Analysis

Finally, we used our analysis platform to examine which datasets were being queried both inside and outside the Bio2RDF network. To do so, we first tabulated the queries sent to specific Bio2RDF endpoints (Figure 2), as well as the queries that used the SERVICE keyword to query SPARQL endpoints that were outside of the Bio2RDF network (Figure 3). Our results show that i) the top 5 Bio2RDF datasets are (in decreasing number of queries posed): Pubmed (with more than 11,000 SERVICE calls), Gene (with almost 1,000 SERVICE calls), Pharmgkb, Drugbank and Bioportal (recently added to the Bio2RDF network); and ii) the top 5 SPARQL endpoints used to complement Bio2RDF queries are: the Gene Expression Atlas (with more than 500 SERVICE requests), Beta Uniprot (a development version of the Uniprot dataset), DBpedia, the Chemical Biology Group and Reactome: a knowledge base of biologic pathways and processes. It is important to notice that 4 of these datasets are funded by the European Bioinformatics Institute. More detailed figures and the data used to generate Figures 2 and 3 can be found at https://plot.ly/%7Ecbuil/77 and https://plot.ly/%7Ecbuil/78 respectively. The results we present in this section show that SPARQL 1.1 federation features are being used to connect to a surprisingly large number of endpoints. In total, there were 5,470 SERVICE calls in the final log files processed, 4,462 of them were directed to Bio2RDF server while 1008 were directed to other endpoints in the LOD Cloud.

6 Conclusions

In this paper we performed an analysis of the queries received at the Bio2RDF servers. This analysis showed first that the amount of repeated SPARQL queries received by these servers is huge (about a 7% of queries are unique), which of course can be used to optimize how servers are caching previously computed answers. Once the duplicate queries were removed, we found that around 50% of the queries contain just a single triple pattern, and that triple pattern is generally under of a SELECT or a CONSTRUCT. Furthermore, 20 query patterns represent 90% of the queries received by the servers, and this can have a high impact on terms of how servers should be optimized to answer queries. It is also interesting to notice that the second most used SPARQL query is DESCRIBE, which is marked as an *"Informative"* query form in the SPARQL 1.1 recommendation document (i.e. its implementation is not mandatory). This indicates that a large portion of the users are trying to understand the shape of the data before querying for information.

A more in detail analysis of the query patterns showed that a significant number of users add operators once they understand the data structure, and hence they obtain more

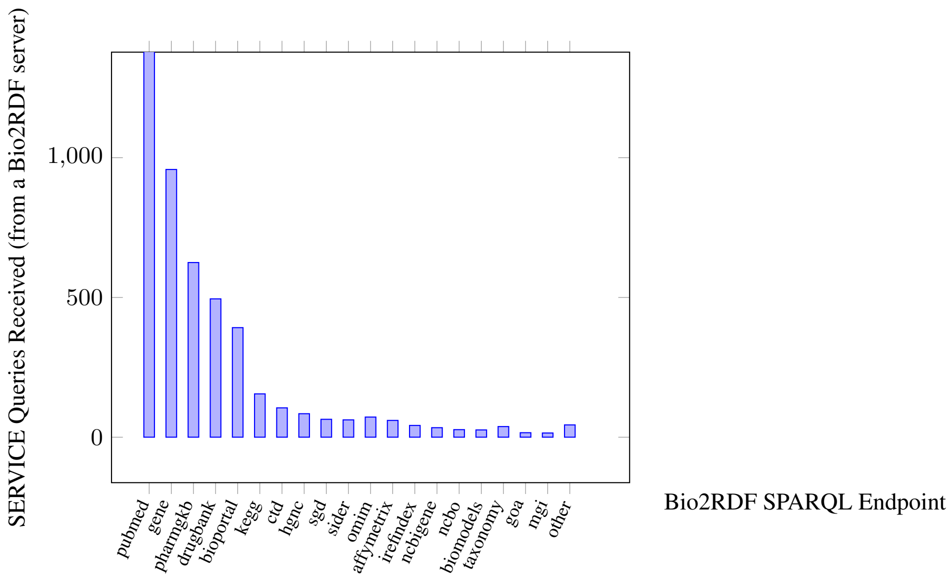


Figure 2. Bio2RDF servers queried using SPARQL Federation. The most Bio2RDF endpoint queried, by large, is Pubmed (http://pubmed.bio2rdf.org/sparql).

results. On the contrary, users remove operators looking for new portions of the data, and their lack of knowledge on the dataset leads to less results. Finally, a brief analysis of the SPARQL Federation queries showed a surprising amount of users that try to link the Bio2RDF data in one endpoint to other Bio2RDF datasets or to other datasets in the LOD cloud using the SERVICE keyword.

Limitations of the experiments. The experiments performed in this work present several limitations. First of all, we did not analyze the URIs in the SPARQL queries and the result sizes may be related to these URIs. It is important no notice that URIs in the queries may exist in the dataset or not, affecting positively or negatively to the queries result sizes and thus to our results. Similarly we did not analyze the effect of the LIMIT solution modifier not the effect of FILTERs, affecting as well to the query's result sizes. However, our results can still provide a useful insight of what Bio2RDF users want to obtain when querying the endopints.

Future work. This work is just a starting point for a more complete and detailed analysis of the Bio2RDF users and queries. A first next step is to overcome the experiments limitations to produce more accurate statistics about the use of Bio2RDF datasets. Once we overcome the limitations, our results can also provide a base line in which we can assess the capability of users to generate more complex queries with guided query tools (e.g. SPARQLED[3], YASGUI[4], etc). The statistics gathered about query patterns and

[3] http://sindice.com/sparqled/
[4] http://yasgui.org/

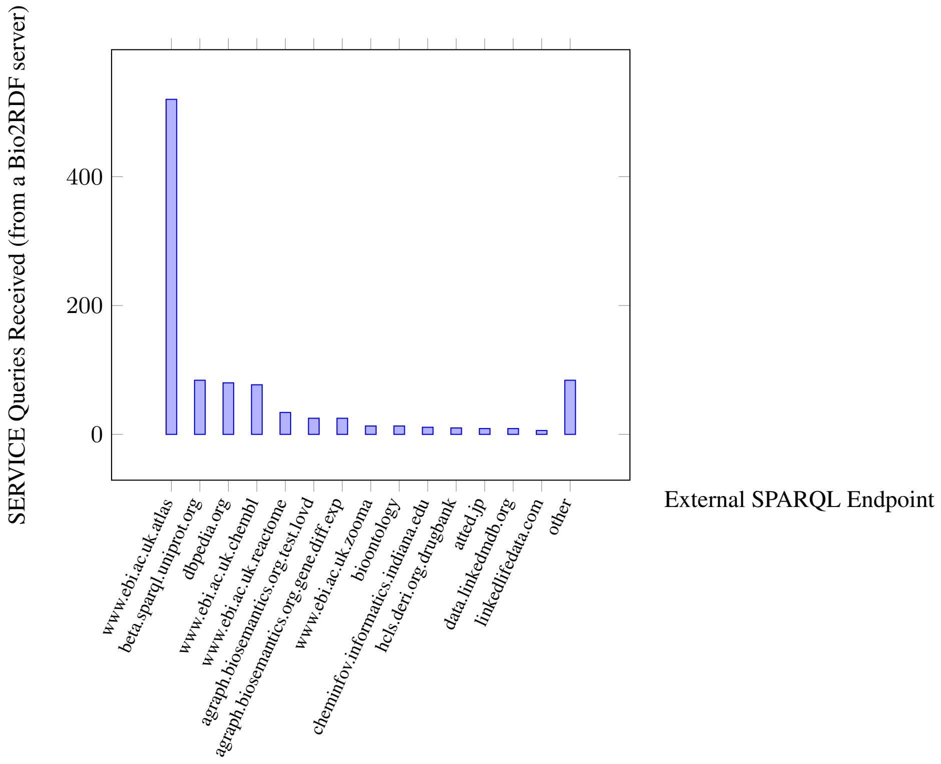


Figure 3. External servers to Bio2RDF queried, using SPARQL Federation. The most queried external endpoint are those from the European Bioinformatics Institute. That suggest that the researchers from that institute try to link the Bio2RDF data with theirs. Endpoints with less than 6 SERVICE queries are grouped in the “Other”.

complexity may be of great help for these type of applications in order to help formulate effective SPARQL queries. Another well know problem in the Linked Data community is that SPARQL endpoints suffer from a performance problem due to the many requests received [8]. As mentioned before, our results be largely used to address this problem by guiding the optimization of software and result caching. Finally, our analysis only targeted the Bio2RDF endpoints. Our final goal would be to generalize our results to more SPARQL endpoints in the LOD Cloud. It is worth noting that our framework is not particular to Bio2RDF and can be applied to arbitrary (clusters of) SPARQL endpoints.

7 Acknowledgments

Carlos Buil-Aranda has been supported by the CONICYT/FONDECYT project 3130617 and by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004. Marcelo Arenas and Martín Ugarte have been supported also by the Millennium Nucleus Center for Semantic Web Research under Grant NC120004

References

1. S. Aljaloud, M. Luczak-Rösch, T. Chown, and N. Gibbins. Get All, Filter Details - On the Use of Regular Expressions in SPARQL queries. In *Proceedings of the ESWC2014 workshop on Usage Analysis and the Web of Data (USEWOD 2014)*, 2014.
2. M. Arias, J. D. Fernández, M. A. Martínez-Prieto, and P. de la Fuente. An empirical study of real-world sparql queries. *CoRR*, abs/1103.5043, 2011.
3. F. Belleau, M.-A. Nolin, N. Tourigny, P. Rigault, and J. Morissette. Bio2rdf: Towards a mashup to build bioinformatics knowledge systems. *J. of Biomedical Informatics*, 41(5):706–716, Oct. 2008.
4. B. Berendt, V. Hollink, K. M. Markus Luczak-Rösch, and D. Vallet. 2nd international workshop on usage analysis and the web of data. in 21st eswc. In *In 21st International World Wide Web Conference (WWW2012)*.
5. B. Berendt, V. Hollink, K. M. Markus Luczak-Rösch, and D. Vallet. 1st international workshop on usage analysis and the web of data. In *In 20th International World Wide Web Conference (WWW2011)*, 2011.
6. B. Berendt, V. Hollink, K. M. Markus Luczak-Rösch, and D. Vallet. 3rd international workshop on usage analysis and the web of data. in 10th eswc. In *In 10th ESWC Semantics and Big Data, Montpellier, France.*, 2013.
7. B. Berendt, V. Hollink, K. M. Markus Luczak-Rösch, and D. Vallet. 4th international workshop on usage analysis and the web of data. in 11th eswc. In *In 11th ESWC Semantics and Big Data.*, 2014.
8. C. Buil-Aranda, A. Hogan, J. Umbrich, and P. Vandenbussche. SPARQL web-querying infrastructure: Ready for action? In *ISWC*, pages 277–293, 2013.
9. A. Callahan, J. Cruz-Toledo, P. Ansell, and M. Dumontier. Bio2rdf release 2: Improved coverage, interoperability and provenance of life science linked data. In *The Semantic Web: Semantics and Big Data, 10th International Conference, ESWC 2013, Montpellier, France, May 26-30, 2013. Proceedings*, pages 200–212, 2013.
10. J. Hoxha, M. Junghans, and S. Agarwal. Enabling semantic analysis of user browsing patterns in the web of data. *CoRR*, abs/1204.2713, 2012.
11. J. Lorey and F. Naumann. Caching and prefetching strategies for sparql queries. In *Proceedings of the 3rd International Workshop on Usage Analysis and the Web of Data (USEWOD)*, Montpellier, France, 0 2013. Best Workshop Paper.
12. M. Luczak-Rösch and M. Bischoff. Statistical analysis of web of data usage. In *Joint Workshop on Knowledge Evolution and Ontology Dynamics (EvoDyn2011), CEUR WS*, 2014.
13. K. Mller, M. Hausenblas, R. Cyganiak, and S. Handschuh. Learning from linked open data usage: Patterns and metrics. In *Web Science*, 2010.
14. F. Picalausa and S. Vansummeren. What are real sparql queries like? In *Proceedings of the International Workshop on Semantic Web Information Management*, SWIM '11, pages 7:1–7:6, New York, NY, USA, 2011. ACM.
15. M. Raghuveer. Characterizing machine agent behavior through sparql query mining. *CoRR*.
16. L. Rietveld and R. Hoekstra. Man vs. Machine Dierences in SPARQL Queries. In *Proceedings of the ESWC2014 workshop on Usage Analysis and the Web of Data (USEWOD 2014)*, 2014.

Keyword Search in the Deep Web

Andrea Calì[1,4], Davide Martinenghi[2], and Riccardo Torlone[3]

[1]Birkbeck, University of London, UK
andrea@dcs.bbk.ac.uk

[2]Politecnico di Milano, Italy
davide.martinenghi@polimi.it

[3]Università Roma Tre, Italy
torlone@dia.uniroma3.it

[4]Oxford-Man Inst. of Quantitative Finance
University of Oxford, UK

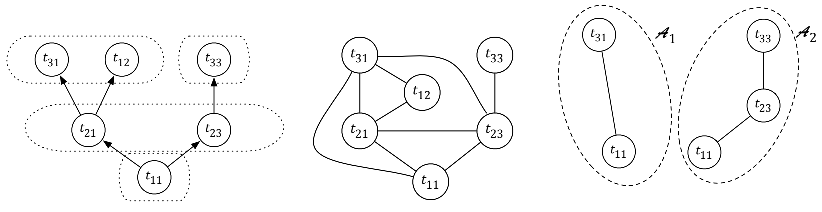
Abstract. The Deep Web is constituted by data accessible through Web pages, but not readily indexable by search engines, as they are returned in dynamic pages. In this paper we propose a framework for accessing Deep Web sources, represented as relational tables with so-called access limitations, with keyword-based queries. We formalize the notion of optimal answer and investigate methods for query processing. To our knowledge, this problem has never been studied in a systematic way.

1 Problem definition

Basics. We model data sources as relations of a relational database and we assume that, albeit autonomous, they have "compatible" attributes. For this, we assume that the attributes of relations are defined over a set of *abstract domains* $\mathbf{D} = \{D_1, \ldots, D_m\}$, which, rather than denoting concrete value types (such as string or integer), represent data types at a higher level of abstraction (for instance, *car* or *country*). The set of all values is denoted by $\mathcal{D} = \bigcup_{i=1}^{n} D_i$.

In the following, we shall denote by $R(A_1, \ldots, A_k)$ a (relation) schema, by $dom(A) \in \mathbf{D}$ the domain of an attribute A, by r a relation over R, and by $\boldsymbol{r} = \{r_1, \ldots, r_n\}$ a (database) instance of a database schema $\mathbf{R} = \{R_1, \ldots, R_n\}$.

Access limitations. An *access pattern* Π for a schema $R(A_1, \ldots, A_k)$ is a mapping sending each attribute A_i into an *access mode*, which can be either input or output; A_i is correspondingly called an *input* (resp., *output*) *attribute* for R wrt. Π. For ease of notation, we shall mark input attributes with an 'i' superscript to distinguish them from the output ones. Let $A'_1, \ldots, A'_l$ be all the input attributes for R wrt. Π; any tuple $\langle c_1, \ldots, c_l \rangle$ such that $c_i \in dom(A'_i)$ for $1 \leq i \leq l$ is called a *binding* for R wrt. Π. An *access* α consists of an access pattern Π for a schema R and a binding for R wrt. Π; the *output* of such an access α on an instance $\boldsymbol{r}$ is the set $\mathcal{T} = \sigma_{A_1=c_1,\ldots,A_l=c_l}(r)$. Intuitively, we can only access a relation if we can provide a value for every input attribute. Given an instance $\boldsymbol{r}$ for a database schema $\mathbf{R}$, a set of access patterns $\boldsymbol{\Pi}$ for the relations in $\mathbf{R}$, and a set of values $\mathcal{C} \subseteq \mathcal{D}$, an *access path* (for $\mathbf{R}$, $\boldsymbol{\Pi}$ and $\mathcal{C}$) is a sequence of accesses $\alpha_1, \ldots, \alpha_n$ on $\boldsymbol{r}$ such that each value in the binding of α_i, $1 \leq i \leq n$, either occurs in the output of an access α_j with $j < i$ or is a value in $\mathcal{C}$. A tuple t in $\boldsymbol{r}$ is said to be *reachable* if there exists an access path P such that t is in the output of some



(a) Access paths (b) Join graph (c) Answers

Fig. 1. Example 1: Reachable portion, corresponding join graph, and answers.

access in P; the *reachable portion* $reach(\boldsymbol{r}, \boldsymbol{\Pi}, \mathcal{C})$ of $\boldsymbol{r}$ is the set of all reachable tuples in $\boldsymbol{r}$ given the values in $\mathcal{C}$.

Keyword queries. A *keyword query* is a set of values in $\mathcal{D}$ called *keywords*.

Example 1 Consider a query $q = \{k_1, k_2\}$, a schema (with access patterns $\boldsymbol{\Pi}$) $\mathbf{R} = \{R_1(A_1^i, A_2), R_2(A_2^i, A_1), R_3(A_1^i, A_2, A_3)\}$, and an instance $\boldsymbol{r}$ such that

$r_1 =$

A_1^i	A_2	
k_1	c_1	t_{11}
c_2	c_3	t_{12}

$r_2 =$

A_2^i	A_1	
c_1	c_2	t_{21}
c_4	c_2	t_{22}
c_1	c_6	t_{23}

$r_3 =$

A_1^i	A_2	A_3	
c_2	c_1	k_2	t_{31}
c_5	c_4	k_2	t_{32}
c_6	c_7	k_2	t_{33}

Figure 1(a) shows the reachable portion of $\boldsymbol{r}$ given the values in q along with the access paths used to extract it, with dotted lines enclosing outputs of accesses. ■

Given a set $\mathcal{T}$ of tuples, the *join graph* of $\mathcal{T}$ is a node-labelled undirected graph $\mathcal{T} = \langle N, E \rangle$ constructed as follows: *(i)* the nodes N are labelled with tuples of $\mathcal{T}$, and *(ii)* there is an arc between two nodes n_1 and n_2 if the tuples labelling n_1 and n_2 have at least one value in common.

Example 1 (cont.) The join graph of $reach(\boldsymbol{r}, \boldsymbol{\Pi}, q)$ is shown in Figure 1(b). ■

Definition 1 (Answer). *An answer to a keyword query q against a database instance $\boldsymbol{r}$ over a schema $\boldsymbol{R}$ with access patterns $\boldsymbol{\Pi}$ is a set of tuples $\mathcal{A}$ in $reach(\boldsymbol{r}, \boldsymbol{\Pi}, q)$ such that: (1) each $c \in q$ occurs in at least one tuple t in $\mathcal{A}$; (2) the join graph of $\mathcal{A}$ is connected; (3) for every subset $\mathcal{A}' \subseteq \mathcal{A}$ such that $\mathcal{A}'$ enjoys Condition 1 above, the join graph of $\mathcal{A}'$ is not connected.*

It is straightforward to see that there could be several answers to a keyword query; below we give a widely accepted criterium for ranking such answers [4].

Definition 2. *Let $\mathcal{A}_1, \mathcal{A}_2$ be two answers of a keyword query q on an instance $\boldsymbol{r}$ of size $|\mathcal{A}_1|$ and $|\mathcal{A}_2|$ respectively; we say that $\mathcal{A}_1$* is better than *$\mathcal{A}_2$, denoted $\mathcal{A}_1 \preceq \mathcal{A}_2$, if $|\mathcal{A}_1| \leq |\mathcal{A}_2|$. The* optimal *answers are those of minimum size.*

Example 1 (cont.) The sets $\mathcal{A}_1 = \{t_{11}, t_{31}\}$ and $\mathcal{A}_2 = \{t_{11}, t_{23}, t_{33}\}$ are answers to q; $\mathcal{A}_1$ is better than $\mathcal{A}_2$ and is the optimal answer to q. ■

2 Keyword-based answering in the Deep Web

We now present a vanilla algorithm to discuss the computational complexity of answering a keyword query q in the deep Web modeled as an instance $\boldsymbol{r}$ of a schema $\mathbf{R}$ with access patterns $\boldsymbol{\Pi}$. Example 1 shows that, in the worst case, we need to extract the whole reachable portion to obtain the tuples involved in an optimal answer. In fact, $\boldsymbol{s} = \mathit{reach}(\boldsymbol{r}, \boldsymbol{\Pi}, q)$ is actually a connected join graph, since every tuple in it is in some output of some access path starting from the values in the query (see for example Figure 1.a), but further paths may exist between tuples in $\boldsymbol{s}$ (see Figure 1.b). Therefore, query answering requires in general two main steps, described in Algorithm 1: *(i)* extract the reachable portion $\boldsymbol{s}$ of $\boldsymbol{r}$; *(ii)* if possible, remove tuples from $\boldsymbol{s}$ so that the obtained set satisfies the conditions of Definition 1, while minimizing its size.

Algorithm 1: Computing an optimal answer ($\mathit{Answer}(q, \boldsymbol{\Pi}, \boldsymbol{r})$)

```
Input: Keyword query $q$, access patterns $\boldsymbol{\Pi}$, instance $\boldsymbol{r}$ over $\mathbf{R}$
Output: Answer $\mathcal{A}$
1. $\mathcal{A} := \mathit{reachablePortion}(\boldsymbol{r}, \boldsymbol{\Pi}, q)$; // see Algorithm 2
2. if $\mathcal{A}$ does not contain all values in $q$ then return nil;
3. else $\mathit{prune}(\mathcal{A}, q)$; // see Algorithm 3
4. return $\mathcal{A}$;
```

A simple way of extracting the reachable portion, inspired by the procedure described in [1], is shown in Algorithm 2. This algorithm may be allowed to terminate early if the answer is not required to be optimal (flag ω set to $false$), and thus can stop as soon as the reachable portion contains all the keywords in the query. This is coherent with the *distinct root-based semantics* of keyword search in relational databases, which provides a tradeoff between quality of the result and efficiency of the method to evaluate it [4].

Algorithm 2: Reachable portion ($\mathit{reachablePortion}(\boldsymbol{r}, \boldsymbol{\Pi}, q)$)

```
Input: Instance $\boldsymbol{r}$ over $\mathbf{R}$, access patterns $\boldsymbol{\Pi}$, initial values $q$
Flag: boolean $\omega$ // if $\omega = true$ the answer is guaranteed to be optimal
Output: Reachable portion $RP$
1. $RP := \emptyset$; $\mathcal{C} := \emptyset$;
2. while an access can be made with a new binding $b$ for some $R \in \mathbf{R}$ wrt. $\boldsymbol{\Pi}$ using values in $\mathcal{C} \cup q$
3.     $\mathcal{O}$ := output of access to $r$ over $R$ with binding $b$;
4.     $RP := RP \cup \mathcal{O}$; // cumulating all the obtained tuples into $RP$
5.     $\mathcal{C} := \mathcal{C} \cup \bigcup_{A \in R, t \in \mathcal{O}} \{t(A)\}$; // cumulating all the obtained values into $\mathcal{C}$
6.     if $\mathcal{C} \supseteq q \wedge \neg\omega$ then break;
7. return $RP$;
```

Basically, determining an optimal answer from the reachable portion corresponds to finding a Steiner tree of its join graph [4], i.e., a minimal-weight subtree of this graph involving a subset of its nodes. An efficient method for solving this problem in the context of keyword search over structured data is presented in [2], where a *q-fragment* can model our notion of answer. Yet, when optimality is not required, a simple technique (quadratic in the size of $\boldsymbol{r}$) to obtain an answer (steps 2–6 of Algorithm 3) consists in trying to remove any tuple from the set as long as it contains all the keywords and remains connected.

Algorithm 3: Pruning ($prune(\mathcal{T}, q)$)

```
Input: Set of tuples $\mathcal{T}$, keyword query $q$
Flag: boolean $\omega$ // if $\omega = true$ the answer is guaranteed to be optimal
Output: Minimal set of tuples $\mathcal{T}$
1. if $\omega$ then return a minimal subtree of the join graph of $\mathcal{T}$ that contains $q$;
2. $\mathcal{T}' := \mathcal{T}$; $\mathcal{T}'' := \emptyset$;
3. while $\mathcal{T}'' \neq \mathcal{T}'$
4.     $\mathcal{T}'' := \mathcal{T}'$;
5.     for each $t \in \mathcal{T}''$ if $\mathcal{T}' \setminus \{t\}$ is connected and $\mathcal{T}' \setminus \{t\} \supseteq q$ then $\mathcal{T}' := \mathcal{T}' \setminus \{t\}$;
6. return $\mathcal{T}'$;
```

The extraction of the reachable portion of an instance $\boldsymbol{r}$ with access limitations can be implemented by a Datalog program over $\boldsymbol{r}$ [1], which can be evaluated in polynomial time in the size of the input [3]. In addition, in [2] it is shown that the optimal q-fragments of $\boldsymbol{r}$ can be enumerated in ranked-order with polynomial delay, i.e., the time for printing the next optimal answer is again polynomial in the size of $\boldsymbol{r}$. Hence, we can state the following preliminary result.

Theorem 1. *An optimal answer to a keyword query against a database instance with access limitations can be efficiently computed under data complexity.*

3 Discussion and future work

In this paper, we have defined the problem of keyword search in the Deep Web and provided some preliminary insights on query answering in this context. As future work on the problem in question, we plan to:

- devise optimization strategies for query answering; in particular, identify conditions under which an optimal answer can be derived without extracting the whole reachable instance;
- leverage known values (besides the keywords), modeled as relations with only one (output) attribute, to speed up the search for an optimal answer;
- study the problem in a scenario in which the domains of the keywords are known in advance: in this case schema-based techniques can be used;
- consider the case in which nodes and arcs of the join graph are weighted to model source availability and proximity, respectively.

Acknowledgments. Andrea Calì acknowledges support by the EPSRC grant "Logic-based Integration and Querying of Unindexed Data" (EP/E010865/1).

References

1. A. Calì, D. Martinenghi. Querying Data under Access Limitations. In *ICDE*, pag. 50–59, 2008.
2. B. Kimelfeld and Y. Sagiv. Finding and approximating top-k answers in keyword proximity search. In *PODS*, pag. 173–182, 2006.
3. M. Vardi. The complexity of relational query languages. In *STOC*, pag. 137–146, 1982.
4. J. Xu Yu, L. Qin, and L. Chang. Search in Relational Databases: A Survey. *IEEE Data Eng. Bull.*, 33(1): 67–78, 2010.

Implementing Data-Centric Dynamic Systems over a Relational DBMS

Diego Calvanese, Marco Montali, Fabio Patrizi, Andrey Rivkin

Free University of Bozen-Bolzano, Piazza Domenicani 3, 39100 Bolzano, Italy
calvanese,montali,patrizi,rivkin@inf.unibz.it

1 Introduction

We base our work on a model called *data-centric dynamic system* (DCDS) [1], which can be seen as a framework for modeling and verification of systems where both the process controlling the dynamics and the manipulation of data are equally central. More specifically, a DCDS consists of a *data layer* and a *process layer*, interacting as follows: the data layer stores all the data of interest in a relational database, and the process layer modifies and evolves such data by executing actions under the control of a process, possibly injecting into the system external data retrieved through service calls. In this work, we propose an implementation of DCDSs in which all aspects concerning not only the data layer but also the process layer, are realized by means of functionalities provided by a relational DBMS. We present the architecture of our prototype system, describe its functionality, and discuss the next steps we intend to take towards realizing a full-fledged DCDS-based system that supports verification of rich temporal properties.

2 Preliminaries

A DCDS is a tuple $\mathcal{S} = \langle \mathcal{D}, \mathcal{P} \rangle$, where $\mathcal{D}$ is the *data layer* and $\mathcal{P}$ is the *process layer*. The data layer defines the data model of $\mathcal{S}$; it is a tuple $\mathcal{D} = \langle \mathcal{C}, \mathcal{R}, \mathcal{E}, \mathcal{I}_0 \rangle$, where: $\mathcal{C}$ is a countably infinite set of constants providing data values, $\mathcal{R}$ is a a relational schema equipped with equality and full denial constraints $\mathcal{E}$, and $\mathcal{I}_0$ is an initial database instance over $\mathcal{C}$ that conforms to the schema $\mathcal{R}$ and satisfies the constraints $\mathcal{E}$.

The *process layer* defines the progression mechanism for the DCDS; it is a tuple $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \rho \rangle$, where: $\mathcal{F}$ is a finite set of functions representing calls to external services, $\mathcal{A}$ is a finite set of (update) actions, and ρ is a process specification.

Intuitively, the process layer captures the dynamics of the domain of interest, while the data layer captures its static properties. More specifically, the process layer describes the actions ($\mathcal{A}$) that can be executed to query and/or update the current state of the system (whose structure conforms to the data layer), and how/when such actions can be executed (process ρ). Some actions need to take fresh values from the external environment; these can be obtained by performing service calls (from $\mathcal{F}$). Every action execution must guarantee that all the constraints in $\mathcal{E}$ are satisfied by the data layer, so as to prevent the exeuction of actions leading to states that are inconsistent with respect to the constraints.

An *action* $\alpha \in \mathcal{A}$ is an expression $\alpha(p_1, \ldots, p_n) : \{e_1, \ldots, e_m\}$ where $\alpha(p_1, \ldots, p_n)$ is the action *signature*, with α the *action name* and $p_1, \ldots, p_n$ the *action parameters*, and $\{e_1, \ldots, e_m\}$ is a set of *(simultaneous) effect specifications*. Each e_i has the form $q_i \rightsquigarrow$ **del** D_i, **add** A_i, where: q_i is a query over $\mathcal{R}$ whose terms are variables, action parameters, and constants from $\mathtt{ADOM}(\mathcal{I}_0)$[1]; and D_i and A_i are sets of facts from $\mathcal{R}$, whose terms include free variables of q_i (which, in turn, include action parameters) and terms from $\mathtt{ADOM}(\mathcal{I}_0)$. Each A_i may include Skolem terms obtained by applying a function $f \in \mathcal{F}$ to any of the terms above. Skolem terms represent external service calls and model the values returned by the external environment when the action is executed.

The process specification ρ, is a finite set of *condition-action (CA) rules*. Each CA rule has the form $Q \to \alpha$, where $\alpha \in \mathcal{A}$ and Q is a query over $\mathcal{R}$, constituting the precondition of the CA rule, whose free variables are the parameters of α (other terms can be quantified variables or constants in $\mathtt{ADOM}(\mathcal{I}_0)$). W.l.o.g., we can assume that there is a single CA rule for each action.

As regards the process execution, it essentially amounts to iterating over the following steps. First, an action α is chosen by the user and the corresponding CA rule in ρ is evaluated over the current database instance $\mathcal{I}$ (initially $\mathcal{I}_0$). This produces a (finite) set of complete *bindings*, for α's parameters. Then, the user is asked to pick one of such bindings, say $\vec{p}$, so as to obtain a *ground action* $\alpha\vec{p}$. The next step is the action execution, which consists of applying all the action effects simultaneously. This requires *(i)* evaluating all queries $q_i\vec{p}$ (with partial assignment to their variables) associated with all effect specifications, *(ii)* binding the values occurring in the answers with the terms in all **del** D_i and all **add** A_i, and *(iii)* first deleting all the facts obtained in all D_i's, and subsequently adding those obtained in all A_i's, from and to the current database instance $\mathcal{I}$. In case some term t in some A_i involves a service call, the corresponding service is called with the appropriate inputs, to obtain the value to be assigned to t. We stress that all deletions take place at the same time, followed by all additions. Importantly, the final update (deletions and additions) is actually performed only if the resulting instance satisfies the constraints in $\mathcal{E}$ (otherwise a new iteration starts again on the current database instance, but the current binding is no longer provided as an option to the user). When the update is performed, a new instance $\mathcal{I}'$ is obtained, over which the process can iterate again. For a detailed description of the execution semantics, we refer the reader to [1]. In the next section, we describe an actual implementation of the system based on RDBMS technology.

3 Specifying and Implementing DCDSs in a RDBMS

A DCDS specification is maintained by the RDBMS, which interacts with the *Flow Engine* implemented in Java. The Flow Engine executes calls to the RDBMS, and handles the interaction with external services through a *Service Manager*.

Specifically, the RDBMSs maintains: *(i)* a *data layer specification* consisting of relational tables, equipped with functional dependencies and additional domain-dependent constraints, and *(ii)* a *process layer specification*, consisting of action metadata in the form of a relational table containing the action names together with their parameters. Moreover, the DBMS stores sets of stored procedures. Each stored procedure represents

[1] The active domain $\mathtt{ADOM}(\mathcal{I})$ of a DB instance $\mathcal{I}$ is the subset of elements of $\mathcal{C}$ occurring in $\mathcal{I}$.

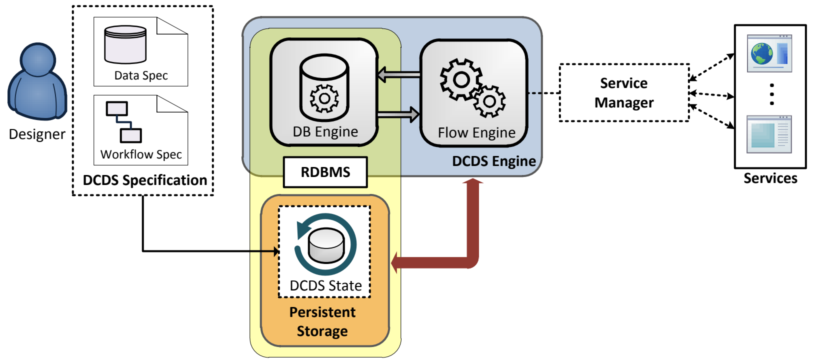


Fig. 1. Architecture of the DCDS implementation

(i) the evaluation of a query in the precondition of a CA rule or a query in an action effect, or *(ii)* an action execution (in terms of database updates).

At each moment in time, the DBMS stores the DCDS snapshot, which is subject to the data layer specification. The snapshot is initialized to the initial state of the DCDS, and then manipulated by the Flow Engine. We illustrate now the operation of the Flow Engine, which initializes the DCDS execution by querying the DBMS about the available actions, and then repeatedly calls an action by coordinating the activation of stored procedures according to the action execution cycle, while interacting with external services to acquire the service call results. Specifically, with reference to Figure 2, an *action execution cycle* is carried out as follows: *(1)* The cycle starts with the user choosing one of the available actions presented by the Flow Engine. *(2)* The Flow Engine evaluates the CA-rule associated to the chosen action α by calling the corresponding stored procedure over the current DCDS snapshot, and stores the returned possible parameter assignments for the action in a temporary table T. All parameters assignments in T are initially unmarked, meaning that they are available for the user to

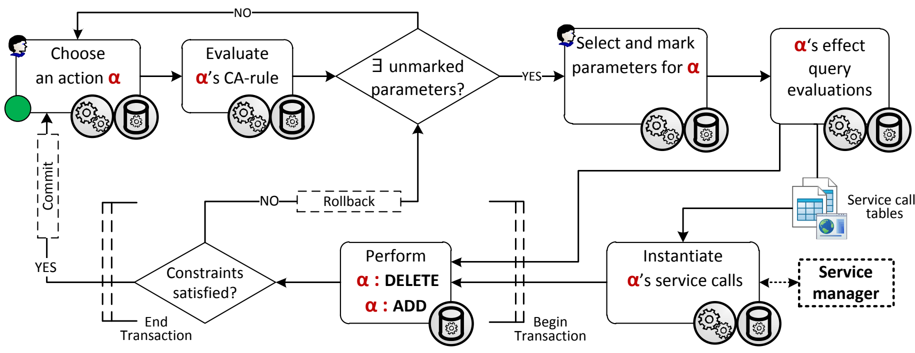


Fig. 2. The action execution cycle

choose. *(3)* If unmarked parameters are present in T, the user is asked to choose one of those, which is marked as unavailable, and the Flow Engine proceeds with evaluating α instantiated with the chosen parameters. *(4)* To do so, first the queries in all the effects of α are executed, which provides values to instantiate both the arguments of the service calls (stored in *service calls tables*, one for each service), and the facts to delete and add from the current snapshot. *(5)* The service calls are executed by calling the *service manager*, and the returned results provide the missing instantiations for the facts to add. *(6)* A transaction is started to perform the delete and add operations. *(7)* If the DCDS constraints are satisfied, the transaction is committed, and the next iteration of the action execution cycle is started. Otherwise, if the constraints are not satisfied, the transaction is aborted so that the DCDS snapshot stays unmodified, and the user is asked to choose a different parameter from the ones still available in T. *(8)* If no unmarked parameters are available, the user is asked to choose another action.

4 Conclusion and Future Work

In this paper we have presented an implementation of Data-Centric Dynamic Systems based on the use of a Relational Database Management System, exploited to handle *all* the aspects of a DCDS, including specification and execution.

The ultimate goal of this research is to build a system for model checking DCDSs. As discussed in [1], the possible evolutions of these systems can be represented by means of a transition system, that can thus be checked against temporal first-order properties, see, e.g., [1,2,4]. The main problem with this task is that the state space can be infinite, thus the standard model checking techniques cannot be applied off-the-shelf. However, under certain conditions, namely when the active domain of all states is guaranteed to be *bounded* [1,2], it is possible to build a faithful abstraction of the system that is finite-state and can thus be checked with such techniques. Our next step will consist in extending our prototype system with the capability to build the finite abstraction, maintaining it in the DBMS, and exploiting state-of-the-art model checking tools, such as NuXMV [3], to perform verification.

References

1. Bagheri Hariri, B., Calvanese, D., De Giacomo, G., Deutsch, A., Montali, M.: Verification of relational data-centric dynamic systems with external services. In: Proc. of PODS. pp. 163–174 (2013)
2. Belardinelli, F., Lomuscio, A., Patrizi, F.: Verification of agent-based artifact systems. JAIR 51, 333–376 (2014)
3. Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: The nuXmv symbolic model checker. In: Proc. of CAV. LNCS, vol. 8559, pp. 334–342. Springer (2014)
4. Deutsch, A., Hull, R., Patrizi, F., Vianu, V.: Automatic verification of data-centric business processes. In: Proc. of ICDT. pp. 252–267 (2009)

Disentangling the Notion of Dataset in SPARQL

Daniel Hernández and Claudio Gutierrez

Center for Semantic Web Research,
Department of Computer Science, Universidad de Chile

Abstract. The notion of dataset in SPARQL seems to be neither a simple nor a well defined notion. In this paper we first review the literature, current documentation and SPARQL engines to show the subleties behind this apparently simple notion and some of the ambiguities of its specification. Then we present formal specifications and algorithms to deal with them in practice.

Introduction

The concept of *dataset* in SPARQL is introduced in several different parts of the W3C documentation (for example in [2, 3, 4, 7, 9, 10, 12]). The specification is spread around, leaves open issues, contain subtleties that result in manifold interpretations, and even in some corner cases, contradict each other.

A first source of misunderstandings is triggered by the two related, but different notions of *default dataset* and *dataset description*. A dataset is defined by the standards of SPARQL and RDF as a set $\{G_0, (u_1, G_1), \ldots, (u_n, G_n)\}$ where each G_i is an RDF graph and each u_i is an IRI. G_0 is called the default graph and each pair (u_i, G_i) is called a named graph. The default dataset is the one a SPARQL endpoint uses when no explicit dataset description is provided in a query request. In the mentioned documentation there is no unique model proposed for dataset descriptions. Indeed, the standard defines two formats for dataset descriptions: (i) In the SPARQL grammar as a sequence of '`FROM` u' and '`FROM NAMED` u' clauses where u is an IRI, and (ii) in the HTTP request with the query string parameters '`default-graph-iri`' and '`named-graph-iri`'. If a dataset description is provided, a dataset must be generated from it. The IRI u in the dataset description clause '`FROM NAMED` u' is assumed to be a reference to a resource that serializes a graph G_u. Thus, the reference is indirect. On the contrary, in the resulting dataset, u will name directly the graph G_u.

The second source of misunderstandings comes from the use of blank nodes. In our experience the following concepts related to blank nodes are difficult to understand or have subtleties when applied to datasets: the *scoping graph*, the operation *merge* when composing the *default graph* and the scope limited to files that are specified for *blank node labels* in RDF and SPARQL. Finally, there are extensions that allow blank nodes as names for named graphs and literals as subjects of RDF triples.

Structure of this paper. The paper is structured in two main sections. In the section 1 we thoroughly analize current misunderstantings in the documentation regarding the definition and use of datasets. In section 2 we propose a formal model for dataset descriptions and give algorithms to build and use the query dataset.

1 Datasets in the literature and engines

Can blank nodes be used as names of named graphs? According to the SPARQL specification *"An RDF dataset is a set* $\{G, (u_1, G_1), (u_2, G_2), \ldots, (u_n, G_n)\}$ *where* G *and each* G_i *are graphs, and each* u_i *is an IRI. Each* u_i *is distinct.* G *is called the default graph.* (u_i, G_i)*, are called named graphs."* [4, §18.1.3]. Thus, blank nodes as names of graphs are not allowed in SPARQL. Restricting names to IRIs is consistent with the SPARQL need to use names to retrieve graphs from the Web. How we interpret the clause '`FROM _:b`'?. We cannot use a blank node to identify a resource or in the Web or a graph in the default dataset. The SPARQL grammar does not allow the clause '`GRAPH _:b {` P `}`'; if it where allowed, then '`_:b`' would not reference a specific graph in the dataset because blank nodes in the query and in the data are in different scopes.

Despite the problems that blank nodes introduce in SPARQL, the RDF specification allows blank nodes as names of graphs[1]. *"An RDF dataset is a collection of RDF graphs, and comprises: (i) Exactly one default graph, being an RDF graph. The default graph does not have a name and MAY be empty. (ii) Zero or more named graphs. Each named graph is a pair consisting of an IRI or a blank node (the graph name), and an RDF graph. Graph names are unique within an RDF dataset."* [7, §4]. There is another specification, TriG, that assumes the possibility of blank nodes as names of graphs: *"In a TriG document a graph IRI or blank node may be used as label for more than one graph statements. The graph label of a graph statement may be omitted. In this case the graph is considered the default graph of the RDF Dataset."* [2, §2.2].

Let s be and endpoint and u be the name of the graph G_u in the default dataset of s. Then, according the The SPARQL 1.1 Graph Store Protocol [9] the IRI s`?graph=`u allows to retrieve the graph G_u. What happens if u is a blank node? According to RDF a blank node must never be used as a name to access a resource, because it *"has no intrinsic name."* [7, §9]. Thus, providing an IRI for G_u based on a blank node u contradicts the RDF semantics.

Despite these these issues, in Jena 2.0 and Virtuoso 6.1 blank nodes as names of dataset graphs are supported.

Can an RDF triple have a literal as subject? According to the RDF specification *"An RDF triple consists of three components: the subject, which is an IRI or a blank node; the predicate, which is an IRI; and the object, which is an IRI, a literal or a blank node."* [7, §3.1]. Thus, an RDF triple must not have a literal as

[1] Note that the concept of dataset was not defined in the earlier RDF 1.0 [6] but was included in RDF 1.1 [7], after the inclusion of RDF datasets in SPARQL.

subject. However, the definition of *triple pattern* suggest that RDF triples may include literals as subject, as triple patterns do: *"A triple pattern is member of the set:* $(T \cup V) \times (I \cup V) \times (T \cup V)$*."* [4, §18.1.5]. Note that T denotes the set $I \cup B \cup L$, called the set of RDF terms. Indeed, the inclusion of literals as triple subjects has been accepted by the RDF core Working Group:

> *[The RDF core Working Group] noted that it is aware of no reason why literals should not be subjects and a future WG with a less restrictive charter may extend the syntaxes to allow literals as the subjects of statements.*
> —Should the subjects of RDF statements be allowed to be literals?
> `http://www.w3.org/2000/03/rdf-tracking/#rdfms-literalsubjects`

In our experience SPARQL implementations do not support literals as triple subjects. Jena 2.0 and Virtuoso 6.1 raise an error when uploading files with literals in the subject position.

The default dataset. The SPARQL specification states: *"If a query provides such a dataset description, then it is used in place of any dataset that the query service would use if no dataset description is provided in a query."* [4, §13.2]. Thus, every SPARQL endpoint may provide a *default dataset* to be used in the absence of a dataset description. Note that the dataset description could be defined not only in the query but also in parameters of the request: *"The RDF dataset may also be specified in a SPARQL protocol request, in which case the protocol description overrides any description in the query itself."* [4, §13.2]. The description can be included in three forms: A parameter in the IRI of the HTTP request, a parameter in the body of the HTTP request or as dataset clauses in the query [3, §2.1].

Angles and Gutierrez [1] have interpreted the specification in a different way. They assumed that the default dataset has no named graphs and an empty default graph, i.e., the default dataset is always $\{\emptyset\}$. This interpretation follows the principle of running queries against the Web so that the evaluation of a query does not depend on the particular SPARQL endpoint that evaluates it. On the contrary, in the specification, a query may be evaluated against a default dataset of the SPARQL endpoint where the query is submitted.

SPARQL federation allows using more than one dataset in the same query. When a query includes a '`SERVICE` s `{` P `}`' clause, then the SPARQL endpoint identified as s may evaluate the graph pattern P against the default dataset of s. Indeed, the specification [10, §3.2] states that the '`SERVICE`' generate a request to the endpoint identified as s with the query Q = '`SELECT * WHERE {` P `}`'. Thus, as the query Q has no dataset description, the dataset used to evaluate P is the default dataset of the endpoint identified by s.

Note that as subselects cannot include dataset descriptions, all dereferencing of graphs from the Web must be done by the endpoint that receives the whole query. Thus, endpoints that are used to delegate the evaluation of graph patterns can only use their own default datasets.

Jena 2.0 and Virtuoso 1.6 follow the SPARQL specification, i.e., they use the default dataset in the absence of a dataset description and in federated queries.

Can graphs in RDF datasets share blank nodes? The RDF specification is clear in allowing blank nodes to be shared across graphs: *"Blank nodes can be shared between graphs in an RDF dataset."* [7, §4]. The TriG language for serializing datasets supports sharing blank nodes across graphs: *"BlankNodes sharing the same label in differently labeled graph statements are considered to be the same BlankNode."* [2, §2.3.1]. Datasets engines such as Jena and Virtuoso preserve the identity of blank nodes when loading dataset serializations that share blank nodes across named graphs.

Despite the clarity in the standards and the assumptions made by engine developers, this question has been a source of misunderstandings. Mallea, Arenas, Hogan and Polleres [8] assumed that blank nodes cannot be shared across graphs. In a later work [5] they recognize their mistake: *"This clarification may serve as a corrigendum for our previous paper in which we stated that blank nodes cannot be shared across graphs in SPARQL [48]. This statement is misleading in that although blank nodes cannot be shared across scoping graphs, they can be shared across named graphs"*. Perez, Arenas and Gutierrez [11] assumed *"for the sake of the simplicity"* that blank nodes where not shared by graphs in a RDF dataset.

This misunderstanding comes from the principle that blank nodes are scoped to files (that is, according to the specification) and the assumption that each graph must be contained in its own file (which is wrong). *"Blank node identifiers are local identifiers that are used in some concrete RDF syntaxes or RDF store implementations. They are always locally scoped to the file or RDF store, and are not persistent or portable identifiers for blank nodes. Blank node identifiers are not part of the RDF abstract syntax, but are entirely dependent on the concrete syntax or implementation. The syntactic restrictions on blank node identifiers, if any, therefore also depend on the concrete RDF syntax or implementation. Implementations that handle blank node identifiers in concrete syntaxes need to be careful not to create the same blank node from multiple occurrences of the same blank node identifier except in situations where this is supported by the syntax."* [7, §3.2]. *"A blank node is a node that is not a URI reference or a literal. In the RDF abstract syntax, a blank node is just a unique node that can be used in one or more RDF statements, but has no intrinsic name."* [7, §3.2].

The SPARQL specification states that to evaluate a query, files referenced in the dataset description must be retrieved to build the dataset. Nothing is said about graphs that are included in the default dataset. *"If a query provides more than one* `FROM` *clause, providing more than one IRI to indicate the default graph, then the default graph is the RDF merge of the graphs obtained from representations of the resources identified by the given IRIs."* [4, §13.2.1]. *"The* `FROM NAMED` *syntax suggests that the IRI identifies the corresponding graph, but the relationship between an IRI and a graph in an RDF dataset is indirect. The IRI identifies a resource, and the resource is represented by a graph (or, more precisely: by a document that serializes a graph)."* [4, §13.2.2].

The use of the merge operation to combine graphs into the default graph suggests that blank nodes can be shared by the RDF files dereferenced when interpreting the dataset description. Moreover, merge is not applied on named graphs, a fact that suggest that blank nodes coming from different files can be shared across named graphs of the dataset resulting of the evaluation of a dataset description.

What does occur if a dataset description references only one remote file as part of the default graph? The merge operation must be applied over a single graph? How is evaluated the dataset description '`FROM` u `FROM` u'? Must the graph resulting of dereferencing u be merged with itself?

The way in which the merge operation is applied is another source of misunderstandings. *"In an RDF merge, blank nodes in the merged graph are not shared with blank nodes from the graphs being merged."* [4, §13.1]. A different definition of merge was provided by Angles and Gutierrez [1]: *"The merge of graphs, denoted $G_1 + G_2$, is the graph $G_1 \cup G'_2$ where G'_2 is the graph obtained from G_2 by renaming its blank nodes to avoid clashes with those in G_1."* According the SPARQL specification $G_1 + G_2$ does not share blank nodes with G_1 nor G_2 as in the definition provided by Angles and Gutierrez.

We identify three strategies to avoid blank node clashes: (i) Rename blank nodes when interpreting files, ensuring that no blank nodes are shared. (ii) Use the merge operator to combine the graphs that compose the default graph. (iii) Use both strategies, (i) and (ii).

The strategy (i) is not mentioned in the SPARQL specification, but in our experience many people think that it must be used as a direct consequence of the fact that blank nodes are scoped to files. The strategy (i) is sufficient to ensure that blank nodes are scoped to files. On the contrary, the strategy (ii) is not sufficient, as blank nodes in named graphs are not renamed, hence can be shared. The use of merge in strategies (ii) and (iii) may introduce spurious identities for blank nodes. For example, let u be an IRI that references a file that serializes the graph G_u. Let us consider a dataset description that contains both clauses: '`FROM` u' and '`FROM NAMED` u'. Then, a blank node that occurs in G_u may be renamed with a fresh blank node in the default graph when applying the merge operation, losing its link with its occurrence in the named graph (u, G_u).

The documentation about merge is not clear. The SPARQL 1.1 Service Description specification [12] introduces the property UnionDefaultGraph to indicate that a service uses the union of the named graphs as the default graph. A property to describe default graphs as a merge of named graphs is not introduced for service descriptions. This seems preferable as in the specifications of SPARQL and RDF 1.1 named graphs are allowed to share blank nodes. However, it seems contrary to the use of merge to construct the default graph described by several '`FROM`' clauses in the query as the SPARQL specification stated.

Neither Jena nor Virtuoso allow referencing remote graphs in the dataset description. Thus, they never dereference an IRI. Moreover, the merge operation is never performed. Indeed, if (u_1, G_1) and (u_2, G_2) are two named graph in a default graph of an endpoint identified as s, then the default graph specified in the dataset description '`FROM` u_1 `FROM` u_2' is $G_1 \cup G_2$. Thus, no blank node renaming is done by Jena and Virtuoso at the moment of evaluating queries.

The task of renaming blank nodes to ensure the file scope of them is performed by Jena and Virtuoso at the moment of loading RDF files. Both engines rename all blank nodes of the form '`_:bn`' with fresh blank nodes and set fresh blank nodes for nodes that have no label. However, Virtuoso 6.1 use identifiers as `<nodeID://b01>` as blank node identifiers. This kind of identifiers responds true when evaluating the function '`isBlank(`b`)`' but (as IRIs) are not scoped. This use of blank node identifiers in Virtuoso does not follow the standard. Indeed, a blank node identifier can be used in the query to refer a blank node in the data. On the contrary, the standard limits the scope of blank nodes to basic graph patterns and disallows the sharing of blank nodes across basic graph patterns in queries. Jena follows the standard raising the error *"blank node reuse is not allowed in this point"* if a blank occurs in more that one basic graph pattern.

How must be interpreted an IRI that occurs several times in the dataset description of a query? The SPARQL specification leaves this question open in the case where an IRI that occurs in more that one '`FROM`' or '`FROM NAMED`' clauses must be dereferenced one or more times. *"The actions required to construct the dataset are not determined by the dataset description alone. If an IRI is given twice in a dataset description, either by using two* `FROM` *clauses, or a* `FROM` *clause and a* `FROM NAMED` *clause, then it does not assume that exactly one or exactly two attempts are made to obtain an RDF graph associated with the IRI. Therefore, no assumptions can be made about blank node identity in triples obtained from the two occurrences in the dataset description. In general, no assumptions can be made about the equivalence of the graphs."* [4, §12.2.3]. Note that if an RDF file is dereferenced twice the file may change during the attempts to get it, resulting in different files. Thus, the results depend on the policy of the service to handle different versions of an RDF file.

Let us consider the dataset description '`FROM NAMED` u `FROM NAMED` u'. The specification states that names must not be repeated in the dataset. *"Each u_i [the IRI that names a graph] is distinct."* [4, §18.1.3]. *"Graph names are unique within an RDF dataset."* [7, §4]. *"Using the same IRI in two or more* `FROM NAMED` *clauses results in one named graph with that IRI appearing in the dataset."* [4, 13.2.2]. Thus the question arises: What is the graph that should be used if u is dereferenced twice (and these copies are not equal)?

Let us consider the dataset description '`FROM` u `FROM` u' and let be G_u the graph obtained of the dereferencing of u (assuming that only one request for u was issued). If the default graph is the union of G_u with itself then the default graph of the dataset will be G_u. On the contrary, if merge is applied, the default graph G_0 will be a graph that duplicate every triple containing a blank node in G_u. Thus G_u will entail G_0 but $G_u \not\subseteq G_0$.

Note that neither Jena nor Virtuoso handle this issue because they do not support dereferencing IRIs that are not names of named graphs in the default datasets.

2 The notion of dataset

The Web is constituted by a finite set of HTTP servers. A server can send HTTP requests and responses to another server. Servers can be disconnected or added to the Web. The internal configuration of a server may change, so the same request can be answered with different responses if the server configuration changes between the requests.

We will assume that servers follow the REST design principles, so a GET request does not change the internal configuration of the server that receives it. As, an SPARQL client that sent only GET requests have no control over changes in the data, then the order and repetition of requests cannot be used by the client to ensure a result.

In the Web, there is a finite set of files serializing RDF graphs. Every file is accessible sending an HTTP request GET u to an HTTP server. The request GET u may result in a response with the file referenced by u if the server s associate u to a file.

In the Web, there is a finite set of particular types of HTTP servers, called SPARQL endpoints, that provide access to datasets via requests that contain SPARQL queries. A SPARQL endpoint is identified by an IRI that can be used to send queries. Let Q be a SPARQL query, then the request GET s`?query=`Q is the SPARQL GET request that sends the query Q to the endpoint identified as s. A successful response with a file serializing the result of Q is sent back if the query is accepted and if no error occurs during the query execution. The result is interpreted as a Boolean value, a sequence of mappings or an RDF graph.

A SPARQL GET request may have the following query string parameters: '`query`' (exactly 1), '`default-graph-uri`' (0 or more) and '`named-graph-uri`' (0 or more). The SPARQL protocol includes also the possibility to send queries via URL-encoded POST or via POST directly. In this paper we will focus in SPARQL queries sent using the HTTP method GET.

The description above roughly indicates how the protocols necessary to evaluate SPARQL queries work. In what follows, we present a formal model that simplifies and make explicit such protocols.

2.1 A data model for SPARQL

Let I, B, L and V be infinite disjoint sets containing the IRIs, the blank nodes, the literals and the variables. Let IB and IBL be the sets $I \cup B$ and $I \cup B \cup L$, respectively. A triple is a tuple in $IB \times I \times IBL$. A graph is a set of triples. A dataset is set $\{G_0, (u_1, G_1), \ldots, (u_n, G_n))\}$ where $G_0, \ldots, G_n$ are graphs, $u_1, \ldots, u_n$ are different IRIs and $n \geq 0$. G_0 is called the default graph and the rest are called the named graphs. A dataset description is a pair (A, B) where A and B are sets of IRIs. A mapping is a partial function with a finite domain in V with range in IBL. There is a function δ that takes a pair (u, t) where u is an IRI and t is a positive real number representing time, and returns either null, or a graph or a dataset. Intuitively, $\delta(u, t)$ is the data that u makes accessible at the instant t. We call endpoints the IRIs that make datasets accessible during a period.

Note that most definitions presented above where taken from the standards and the literature, except the model of the dataset description. The definition of a dataset description as a pair of sets (instead of a list of dataset clauses) gives an unequivocal semantics to the order of dataset clauses and to the repetition of dataset clauses in the syntax of the dataset description. We argued previously that assuming order or considering repetitions makes no sense if the client has no control over how the data changes over time.

2.2 Algorithms for the dataset

A request is a pair of IRIs (a, b) at time t. A response is a tuple (a, b, r) where a and b are IRIs and r is null (interpreted as a not-found error message[2]), a graph, a sequence of mappings or a Boolean value. A query IRI is an IRI u that has the format $s?p$ where prefix s is called the IRI endpoint and the query string p must contain the parameter '`query`' and zero or more parameters '`default-graph-uri`' and '`named-graph-uri`'. The procedure to generate the response to a request is described by the Algorithm 1. Note that this algorithm allows retrieving the whole dataset through the endpoint IRI. This is not supported by the current standard. However, it is possible to download it with queries or with the graph store protocol [9].

Algorithm 1: Response(a, b)

```
Data: A request $(a, b)$ sent at an instant $t$.
Result: The response of the request $(a, b)$.
if $b$ is a query IRI $s?p$ and $\delta(s, t)$ is a dataset then
    return $(s, a, r)$ where $r$ is the result of evaluating the query and parameters
    indicated in the query string $p$ in the endpoint $s$, that is, using the dataset
    $\delta(s, t)$ as the default dataset.
else
    return $(b, a, \delta(b, t))$
end
```

The procedure to generate the dataset description is presented in Algorithm 2. The endpoint must first check the parameters in the request. If no dataset description is provided in such parameters it finds the dataset description in the query.

Each IRI u in a dataset description must be associated with a graph G_u. If the default daset contains a graph named u, then G_u is the graph associated with u in the dataset. Else, the graph G_u is the result of renaming the blank

[2] Every request must result in a response that occurs after the request. In the HTTP protocol, a request may result in manifold unsuccessful results (a time out, an internal error, etc.). For the sake of the simplicity, in our model we consider an unsuccessful result as a null result.

Algorithm 2: DatasetDescription($a, s?p$)

```
Data: A request $(a, s?p)$ where $s?p$ is a query IRI.
Result: The dataset description of $(a, s?p)$.
Let $Q$ be the value that occurs in $p$ as the property 'query'.
Let $A$ and $B$ be the set of IRIs that occur in $p$ associated to the properties
'default-graph-iri' and 'named-graph-iri', respectively.
if $A \cup B \neq \emptyset$ then
    return $(A, B)$
else if there is at least a 'FROM' or a 'FROM NAMED' clause in $Q$ then
    Let $A$ and $B$ be the set of IRIs that occur in $Q$ in the clauses 'FROM' and
    'FROM NAMED', respectively.
    return $(A, B)$
else
    the request has no dataset description.
end
```

nodes with fresh labels in the response of the request (s, u). This procedure is formally described in Algorithm 3. Finally, Algorithm 4 describes the algorithm to be used when answering the query.

Algorithm 3: DatasetFromDescription(A, B)

```
Data: A dataset description $(A, B)$.
Result: The dataset built from the description $(A, B)$.
Let $G \leftarrow \bigcup G_i$ for all graph $G_i$ in the default dataset (the scoping graph); let
$G_0 \leftarrow \emptyset$ (the default graph); let $D \leftarrow \{G_0\}$ (the resulting dataset); let $D_0$ be the
default dataset.
for $u \in A \cup B$ do
    if exists $(u, G') \in D_0$ then
        Let $G_u \leftarrow G'$
    else
        Send the request $(s, u)$ and let $(u, s, r)$ be its response.
        if $r$ is a graph then
            Let $G'$ be the result of replacing all blank nodes in $r$ with blank
            nodes that does not occurs in $G$;
            Let $G_u \leftarrow G'$
        end
    end
    if $u \in A$ then
        Let $G_0 \leftarrow G_u$
    end
    if $u \in B$ then
        Let $D \leftarrow D \cup \{(u, G_u)\}$
    end
end
return $D$
```

Algorithm 4: Dataset$(a, s?p)$

```
Data: A request $(a, s?p)$ where $s?p$ is a query IRI and $\delta(s, t)$ is a dataset.
Result: The dataset of the request at $t$.
if DatasetDescription$(a, s?p)$ is defined then
    $(A, B) \leftarrow$ DatasetDescription$(a, s?p)$;
    return DatasetFromDescription$(A, B)$
else
    return $\delta(s, t)$.
end
```

3 Conclusions

We have proposed a formal model that defines the dataset that is used to evaluate every graph pattern in a query. The model was designed to be as close as possible to the RDF and SPARQL specifications; to formalize current natural-language specification; and to clarify some ambiguities it has.

Acknowledgements: The authors thank funding to Millennium Nucleus Center for Semantic Web Research under Grant NC120004.

Bibliography

[1] R. Angles and C. Gutierrez. SQL Nested Queries in SPARQL. In *Procedings on the Alberto Meldenzon Workshop (AMW)*, 2010.

[2] C. Bizer and R. Cyganiak. RDF 1.1 Trig: RDF Dataset Language. Recommendation, World Wide Web Consortium, Febrary 2014.

[3] L. Feigenbaum, G. T. Willians, K. G. Clark, and E. Torres. SPARQL 1.1 Protocol. Recommendation, World Wide Web Consortium, 2013.

[4] S. Harris and A. Seaborne. SPARQL 1.1 Query Language. Recommendation, World Wide Web Consortium, 2013.

[5] A. Hogan, M. Arenas, A. Mallea, and A. Polleres. Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27–28(0):42 – 69, 2014. ISSN 1570-8268. doi: http://dx.doi.org/10.1016/j.websem.2014.06.004. URL `http://www.sciencedirect.com/science/article/pii/S1570826814000481`. Semantic Web Challenge 2013.

[6] G. Klyne and J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium, February 2004.

[7] G. Klyne, J. Carroll, and B. McBride. RDF 1.1 Concepts and Abstract Syntax. Recommendation, World Wide Web Consortium, February 2014.

[8] A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On blank nodes. In L. Aroyo, C. Welty, H. Alani, J. Taylor, A. Bernstein, L. Kagal, N. Noy, and E. Blomqvist, editors, *The Semantic Web – ISWC 2011*, volume 7031 of *Lecture Notes in Computer Science*, pages 421–437. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-25072-9. doi: 10.1007/978-3-642-25073-6_27. URL `http://dx.doi.org/10.1007/978-3-642-25073-6_27`.

[9] C. Ogbuji. SPARQL 1.1 Graph Store Protocol. Recommendation, World Wide Web Consortium, Mar. 2013.

[10] E. Prud'hommeaux and C. Buil-Aranda. SPARQL 1.1 Federated Query. Recommendation, World Wide Web Consortium, 2013.

[11] J. Pérez, M. Arenas, and C. Gutierrez. Semantics of SPARQL. Technical Report, TR/DCC-2006-17, Universidad de Chile, Octover 2006. URL `http://users.dcc.uchile.cl/~jperez/papers/sparql_semantics.pdf`.

[12] G. T. Willians. SPARQL 1.1 Service Description. Recommendation, World Wide Web Consortium, Mar. 2013.