Beta-algebra: Towards a Relational Algebra for Graph Analysis

Luiz Gomes-Jr[*][†]
gomesjr@ic.unicamp.br

Bernd Amann[†]
bernd.amann@lip6.fr

André Santanchè[*]
santanche@ic.unicamp.br

[*]Institute of Computing
State University of Campinas (UNICAMP)
Campinas – SP – Brazil

[†]LIP6 (UMR 7606/CNRS)
Université Pierre et Marie Curie (UPMC)
Paris – France

ABSTRACT

Graph analysis is an essential tool to understand natural and man-made networks, such as social networks, food webs, transportation infrastructures, etc. Although graph analysis has fomented the development of algorithms, visual tools, and distributed processing frameworks, there is still little support for analysis at the query language level. Current graph query languages are mostly concerned with flexible matching of subgraphs, while graph processing frameworks are mostly concerned with fast parallel execution of instructions.

Our goal is to provide analysis capabilities at the language level, allowing more interactive and explorative query-based analysis. In this paper, we present our ongoing efforts towards a relational algebra extension that offers an operator for graph-based data aggregation. The beta (β) operator is composed of four suboperators, which are used to control the path-based aggregations. The β-algebra allows seamless composition of queries that mix relational and graph-based aspects.

Here we introduce our current algebra and provide examples of its use. We also show how we are using the analysis strategy in query scenarios. Since the algebra-based query scenario allows for execution plan rewritings, we also discuss our first efforts on equivalence rules for query optimization.

Keywords

Graph algebra, relational algebra, Complex Networks, graph data models, graph query languages

1. INTRODUCTION

Graph analysis has become an important tool in diverse fields. Social, transportation, communication, and biological networks are examples of information often organized as graphs, which require specific tools and algorithms for proper data analysis.

The area of Complex Network Analysis [6] has advanced in the last decades and has produced several models, algorithms and techniques to study natural and human-made networks. In terms of database support, there has been a strong acceleration in the usage of graph databases and query languages, as well as in the development of the underlying algorithms and mechanisms. There is, however, still a big gap between graph query languages and the analysis techniques. Current graph query languages offer little support for the type of analysis required for complex networks. Such a gap is not present, for example, in traditional relational databases, which support query languages that offer aggregation operations that are the basis of more sophisticated multidimensional analysis.

Our goal is contributing towards bridging this gap. We aim at developing data management and querying mechanisms that offer a better support for network analysis. In this paper, we present our first steps towards creating an algebra that offers a graph-based aggregation operation. We expect this algebra to be the basis of more expressive queries, supporting declarative and interactive graph data exploration.

Our proposed algebra is based on Codd's relational algebra [4]. This has several advantages: (i) it provides a well established theoretical basis; (ii) it allows the combination of traditional relational operations alongside our proposed graph operation; (iii) it is a de facto standard in database research. Having an underlying algebra allows a better understanding of the semantics of the query language and, most importantly, allows the definition of rewriting rules for execution plan optimization. As an extra benefit, relational algebra compatibility also simplifies implementation in current relational database systems.

In simple terms, our goal with the algebra is to provide graph-based aggregation of values. The core of our proposal is the beta (β) operator, which encapsulates the graph traversal procedure and allows parametric control over the aggregation of values. We see graph-based aggregation as a generalization of relational aggregation over sets. Considering that most useful relational aggregations perform joins before applying an aggregation operation, we adopt this pattern of first deriving relationships between the data (joins or graph traversals) and then aggregating the values as the basis for our new constructor.

Combining the advantages of relational query languages and graph analysis, the proposed algebra allows the construction of queries involving subgoals such as: obtain a subgraph based on given nodes properties and edge labels; calculate the reputation of the nodes in the subgraph; combine the reputation and the average distance to a given reference node in the general graph; order the resulting nodes based on the final score. In our envisioned scenario, such queries would be starting points for deeper explorative analysis, with goals such as: analyze the average node degree for the top-k nodes returned in the query; obtain the average value for a certain attribute for the bottom-k scoring nodes, etc.

This paper is organized as follows: Section 2 describes related work and fundamental concepts. Section 3 presents the definition of our algebra alongside with query and execution examples. Section 4 briefly describes our current approach for querying and initial query rewriting rules for execution plan optimization. Finally, Section 5 concludes the paper.

2. RELATED WORK

There is great diversity of graph query languages, which have pushed the boundaries for more expressive constructors [16]. Graph query languages are often based on conjunctive regular path queries (CRPQs). CRPQs are the basis for several graph languages, such as GraphLog [5] and SPARQL[1]. Recent developments have extended CRPQs in order to allow constraints over path properties. These types of queries have been described as extended conjunctive regular path queries (ECRPQs) [2]. ECRPQs also allow paths to be returned as query results. These queries are all focused on data selection and support only the simplest cases of analysis.

Query languages such as in GID [15] allow ranking based on pre-calculated metrics of importance which capture the dynamics of a snapshot of the network. Our goal is to enable a higher level of on-demand analysis in graph query languages. To that extent, we are working towards an algebra that can handle graphs and aggregations over paths. The goal is to use this algebra to build more flexible query languages.

Several algebras that support graphs have been proposed. To our knowledge, the algebra that is closer to our goals is the alpha-algebra [1], which also serves as inspiration for the name of our operator. The alpha operator derives the transitive closure for tables that express self-relationships – e.g. CONNECTS(from, to, distance). The algebra supports aggregation and filtering over paths through the delta (Δ) attribute, which is an internal relation containing each path history in the result set. Conceptually, the alpha operator has two main characteristics that make it unsuited for our needs: the operator always processes until reaching fix point, and there is a single point for value aggregation. In our algebra, we add more flexible stop conditions and split aggregation in four suboperations (Section 3). We also change the underlying model and add several elements for querying convenience. The changes allow more analysis algorithms to be represented in the algebra as well as providing more opportunities for optimization based on query rewritings (Section 4).

[1]http://www.w3.org/TR/sparql11-query

Frasincar et al. [8] propose an algebra for RDF, with some operators inspired by relational algebra. The algebra enables both querying and construction of RDF models. Although the algebra shares many of the goals in this paper, the focus is on the complete RDF-S model, which incurs considerable complexity when compared to our simpler graph model. Most importantly, the proposed algebra does not support graph-based aggregation, our main focus.

In a more recent and simpler proposal, Cyganiak [7] defines a relational algebra for the SPARQL language over the RDF model. The authors rely on a global reference table containing RDF triples (subject, property, object) as basis for the operations. We use a similar strategy for our underlying model as we employ global tables for nodes and relationships to represent the graph in the database. Their proposal, however, does not deal with aggregation or analysis issues.

The demand for graph analysis in large scale has motivated the development of several frameworks for distributed computation: Google's Pregel, the first NoSQL implementation in that scale, was followed by diverse proposals including GraphLab [13] and GraphX[2]. These models focus on parallelization of the API operations and do not provide declarative languages as means for data querying. GraphX shares some of our motivation since it aims at simplifying mixed analysis that include graphs and relations. The focus is, however, on general parallel computation and not on querying. Our goal is to provide a higher level, declarative language for graph querying and analysis, allowing a more interactive and explorative interface with the user. Although we are not concerned with distributed computation at the moment, we believe that would be a natural evolution for our framework.

The Complex Networks [6] field is a prominent area that would benefit from query-base graph analysis. Complex network formation is based on localized phenomena, which in a global scale determines emergent behavior that cannot be assessed based merely on the analysis of parts of the system. The researchers employ a variety of models and algorithms to derive knowledge from the structures. Among the frequently used algorithms are the well known PageRank [3] and HITS. Most of the analysis in the field is done using *ad hoc* applications with no database support.

The algebra that we propose in this paper has been developed in the context of our Complex Data Management System (CDMS) [11], which aims at providing a database-like framework for complex network analysis and management. Appropriate query languages (and underlying algebras) would allow a more fine-grained, exploratory and interactive interaction with the networks.

3. THE BETA-ALGEBRA

In this section we describe the requirements for our algebra and present the beta operator alongside example queries.

3.1 Requirements

The basic requirements for the proposed algebra are:

[2]https://spark.apache.org/graphx/

Allow traditional relational aggregation: given the widespread use and familiarity with relational database queries, it is important to build on and leverage this foundation. Moreover, a graph analysis workflow often contains routines that are typically relational (counting, ranking, statistics, etc).

Enable aggregation over path traversals: Most graph analysis tasks involve aggregating values along graph traversals. It is important to allow flexible and case-specific aggregation functions that are applied as the graph is traversed.

Preserve the closure of relational algebra: It is important that the aggregation operates over and produces relations. This allows compatibility with relational algebra properties studied for decades, as well as making the language more flexible and composable.

Support flexible selection of nodes of interest: To enable explorative analysis over graph data it is important to have efficient means to filter nodes and relationships that are to be part of the analysis. A declarative language would naturally allow this type of flexibility.

Offer means to express subgraphs to constrain the analysis: Currently, most complex networks analysis is done over graphs as a whole. We believe this is highly associated with a lack of convenient means to select a subgraph of interest and apply the analysis over the selection. An appropriate algebra needs to enable graph-based constraints over the execution of the algorithms.

Rewriting rules for cost-based optimization: One of the main advantages of building query languages over an underlying algebra is the possibility of rewriting the queries for faster execution. Including graph-specific operations allows the specification of semantically sound rewriting rules for the graph setting.

Other requirements, not covered in this paper but that we want to address soon are:

Include convergence criteria for operation termination: This would allow the implementation of complete versions of popular graph analysis algorithms (e.g. PageRank).

Provenance/Path information for query results: Like in the alpha algebra with its delta attribute, adding information about paths traversed by the operations allows more flexibility for query composition and allows optimizations such as avoiding loops in graphs.

3.2 Data model

In this paper we use a simple interpretation of graphs in the relational model. In this model, a labeled property multigraph[3] [14] G is represented in two relational tables, V (vertices or nodes) and E (edges or links). The tables contain attributes representing the properties for all nodes and edges (typically highly sparse tables). The V table is in the form $\langle node, a_1, a_2 ... a_n \rangle$, where $node$ represents the node id in the database, and $a_1, a_2 ... a_n$ are node properties.

[3]a graph with labeled links and properties for nodes and links

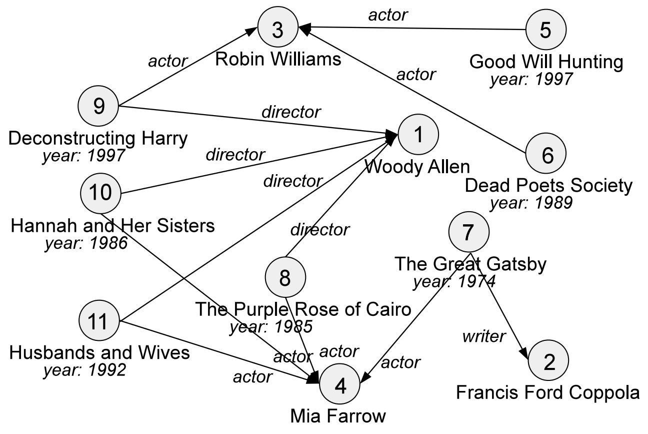


Figure 1: Subgraph from a movies dataset

The E table is in the form $\langle source, dest, label, a_1, a_2 ... a_n \rangle$. $source$ and $dest$ are ids representing the connected nodes for a given edge (the order implies the direction), $label$ is the label according to the labeled property graph model, and the properties are as in the V table.

This data model is the reference for the proposed algebra. However, as with any other model, it does not impose implementation constrains (e.g. it can be implemented over a pure graph database and use efficient structures to represent the sparse tables). Our own implementation stack does not include any traditional relational component.

In the following paragraphs, the beta operator will be presented informally, in increasingly complexity as parameters and suboperations are introduced. Since the operator can be seen from either a graph or relational perspective, we will use equivalent terms interchangeably (e.g. join and traversal step). The semantics, however, is always relational.

3.3 The beta operator

Much like the alpha operator, the beta operator assesses recursive relations in the database. However, the main goal is not to derive transitive closure. Instead, the focus is on data aggregation along the traversal of the relations. In its simplest form, the beta operator performs a single join between a single column source table and the table E. An union operation is then applied to aggregate the original and new nodes. For the sake of readability, we omit extra join, projection, and renaming operations required to maintain the original schema after the execution of the operator. In a graph interpretation, the beta operator augments an initial set of nodes with all of their neighbors. For example, based on the graph in Figure 1, the beta operator applied to a source table containing one column with node ids $\{9, 10\}$ would produce $\{9, 10, 3, 1, 4\}$. This operation is represented as $\beta(\sigma_{id \in \{9,10\}}(V))$.

In general, we represent the beta operator as $n\beta_p(R)$, with $p = \langle s, dir, set, map, reduce, update, C \rangle$. Several parameters are used to control the behavior of the beta operator. s is the join condition, which accepts Boolean expressions just like its relational counterpart. n determines the number of recursive calls to the operator (i.e. consecutive joins).

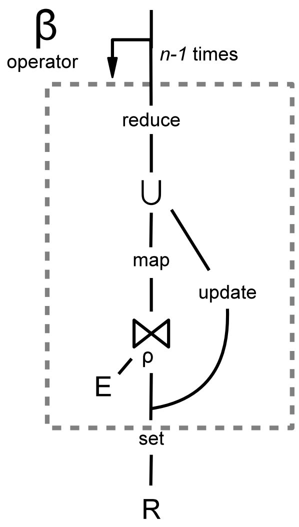


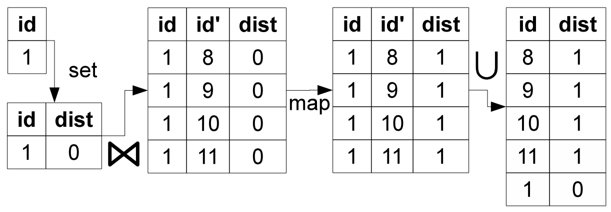
Figure 2: Simplified execution tree template

$dir \in \{inbound, outbound, both\}$ determines the order of the relations in the joins (or the direction of the graph traversal operations). The optional parameter *source* determines, for tables with more than one node column, the column over which the beta operator operates (the default is the fist column).

The operator keeps the algebra closed since (i) it always produces a table with at least the same columns as the input table, and (ii) it can be defined using standard relational algebra with aggregation. The design choices (such as encapsulating the table E inside the beta operator) are for convenience and to focus on what we think are the most important aspects of an aggregation operation. This aspect is inspired by the introduction of the relation join that despite being a redundant operation has directed the focus of the research on properties and optimizations of a central element of the model.

The most important elements of the beta operator are the aggregation suboperations. To allow full control of the computation as the graph is traversed, we define four operations: *set*, *map*, *reduce*, and *update*. *set* is a function that attributes a value to a new column before the join (traversal) operation is performed. *map* calculates a new value based on each node in the source relation. The new values are associated with the neighbors after the join operation. *reduce* is a function that aggregates over values for the same source node (equivalent to a group by). Finally, *update* redefines the aggregation values for the source nodes before the union. Figure 2 shows a simplified execution tree for the beta operator. As an example, the query:

$\sigma_{type=movie}(5\beta(\sigma_{id=1}(V)))$,
with {set: dist=0, map: dist=dist+1,dir=both}, obtains distances to movie nodes that can be reached from the initial node 1 (director Woody Allen) in up to 5 steps. Figure 3 shows the partial tables after the suboperations for the first iteration of the query. The query:



id
1

id	dist
1	0

id	id'	dist
1	8	0
1	9	0
1	10	0
1	11	0

id	id'	dist
1	8	1
1	9	1
1	10	1
1	11	1

id	dist
8	1
9	1
10	1
11	1
1	0

Figure 3: Snapshots of the resulting tables inside the beta operator for the first iteration of the example query.

$\sigma_{type=movie}(5\beta(\sigma_{id=1}(V)))$,
with {set: dist=0, map: dist=dist+1, reduce: dist=MIN(dist)} obtains the minimum distances in the same setting.

Another parameter is the optional graph constructor C. Its purpose is to specify the effective subgraph for the beta operator, constraining the search space for the traversals. The reference points for the constructor are the input nodes. C has its own parameters: *radius* is the maximum distance from the reference points; s is the edge selection expression (similar to the join condition in the beta operator). Nodes and links beyond the radius or that do not match the selection are ignored by the beta operator. The addition of the graph constructor in the algebra is also for convenience sake. The same effect could be obtained by a series of joins and selections over the E table. The query:

$5\beta(\sigma_{type=movie}(V))$,
with {set: rank=1, map: rank=rank/|e.out()|,
reduce: SUM(rank), reset: rank = 0, C: {radius:3}}, is a simplified PageRank algorithm executed for five iterations (not until convergence) over a graph of radius 3 around the source nodes in R. $|e.out()|$ represents the number outbound nodes (that can be obtained with traditional algebra aggregation). If the number of source nodes is known and its inverse is used in set function, the query obtains, for each node in the constructed graph from C, the probability that a random walker would stop at the node after five steps.

Other parameters and functionalities that we want to investigate are (i) specifying stop conditions for the beta operator, including simple test and convergence properties, (ii) recording path traversal information in a delta attribute, with functions that operate over it (similar to the alpha-algebra), and (iii) a modifier equivalent to the SQL *distinct*, that uses the delta attribute to avoid the computation of cycles.

4. QUERYING AND OPTIMIZATION

In this section we present our initial attempts with query language design and rule-based optimization.

4.1 Querying

We are developing the algebra presented here to support the query language that we have been developing as part of our CDMS system. The language that we are currently using is an extension of popular graph queries as shown in Figure 4. The language shows the type of queries we are envisioning, although it is less expressive than the algebra that we are

```
SELECT DISTINCT ?actor WHERE {
  ?film movie:director director:1 .
  ?film movie:actor ?actor .                                   a
  ?film movie:initial_release_date ?date .
  FILTER ( fn:starts-with(?date, "199") ) }
RANK BY RELEVANCE OF ?actor TO director:1

START diag=node(*)
   WHERE has(diag.Type) and diag.Type = "Diagnose"
   RETURN diag                                                 b
RANK BY
   %r RELEVANCE OF diag TO node( %p ) WEIGHTED,
   %c CONNECTIVITY OF diag TO node( %p ) WEIGHTED
```

Figure 4: Query examples. a) extends SPARQL and b) extends Cypher

proposing. We plan to design a more expressive language following the definition of our algebra.

In the initial language, the graph-based aggregation is expressed in a RANK BY clause. The clause accepts metrics that aggregate values over graph traversals as in the algebra presented. For example, Relevance is a generalization of the notion of relevance in Information Retrieval, attributing higher scores to elements that have multiple and more specific connections (paths). This metric can be represented in our algebra by a beta operator with aggregation functions $\{set : score = 1, map : score = (score * 0.9)/e.out(), reduce := SUM(score)\}$. Details about the metrics and queries can be found in [10].

Figure 4a shows a query that retrieves actors whose careers are strongly correlated (relevant) with the director Woody Allen (id 1). Based on the subgraph in Figure 1, Mia Farrow would have a much higher score than Robin Williams. Another interesting query, that includes traditional relational aggregation, would be to find the pair (actor, director) with the maximum mutual relevance. This type of query would be hard to express using current graph or relational queries.

Figure 4b shows a query that we used for a nursing diagnosis task. Possible diagnosis are ranked based on their connections with the symptoms identified in the patients. This query contains a combination of two different metrics (Relevance and Connectivity).

4.2 Rewriting rules

An important motivation for introducing a new algebra is to better understand the computation complexity and define rewriting rules for query optimization. This work is still ongoing and we will only show some first examples for illustration.

The first rule is about the bidirectionality of the analysis. A beta operation that starts on a group of nodes and selects another group of target nodes can be reversed (changing the directions of the allowed edges). Reversing the direction can, in certain cases, reduce the search space by avoiding dense regions of the graph. For example, a three step undirected traversal from node 1 to node 6 in Figure 1 visits 9 nodes, while the traversal from 6 to 1 activates only 4 nodes. This rule can be represented as:

$\sigma_a(\beta(\sigma_b(R))) \equiv \sigma_b(\overline{\beta(\sigma_a(R))})$, where $\overline{\beta}$ represents the β operator with inverted directions (parameter dir). We are omitting, for sake of simplicity, extra joins and renamings that would make the outmost selections equivalent. We have tested this strategy for the query in Figure 4a against a comprehensive movie database [12]. The execution was reduced to half the time of the baseline query. The initial results, using a different formalization, were published as a technical report [9]. We still have to assess the subset of aggregation functions that allow the use of this rule. In practice, this rule requires cost-based planning, which we have not implemented yet.

Another rule, regarding compositionability of operators, combines aggregation functions that are applied over the same data by different beta operators. It can be represented as:

$\sigma_a(\beta_p(\sigma_b(R))) \bowtie \sigma_a(\beta'_{p'}(\sigma_b(R))) \equiv \sigma_a(\beta_{p \bullet p'}(\sigma_b(R)))$, where p represents the tuple of aggregation functions for the operator and $p \bullet p'$ combines the respective functions. The functions in p and p' must not make conflicting operations over the same attributes. We expect this type of rewriting to be very common, as multiple metrics can be used for the same target nodes (as in the query b in Figure 4). We have not yet implemented this rule in the system.

We have also explored options for speeding up queries beyond rule-based rewritings. We have tested, with positive results, caching paths between nodes for metrics that need to traverse the paths in both directions. We also explored a few query approximation strategies for specific metrics. These experiments are also reported in [9]. Another possibility is to materialize graphs constructed from the parameter C for use in multiple beta operators in the same query (rewriting the execution tree to take advantage of the materialized graph).

5. CONCLUSION

Graph analysis has become an important requirement in a wide range of modern applications and research fields. This type of analysis is currently highly specialized, employing ad-hoc applications or complex distributed frameworks. Graph databases offer little support for graph-based aggregations that would allow for query-based analysis.

Here we presented our ongoing work on the beta-algebra, which is intended to allow graph-based aggregations for declarative query languages. The algebra extends the relational model to support graph traversals and allows the control of several aspects of the aggregations.

We have shown examples of the use of the algebra and how it fits in our broader goal of developing data management and querying mechanisms specific for graph/complex network analysis. Also, we presented initial tests and directions for query optimization based on execution plan rewriting, along with our preliminary experimental results.

Our algebra allows more expressive querying when comparing to pure relation aggregation and CRPQs. The increased expressiveness enables explorative analysis and more interactive data manipulation. It also enables seamless integration of relational and graph-based analysis, which is a common application scenario. We believe the algebra is a good basis to build expressive query languages as well as useful optimization strategies.

Ongoing and future work include the expansion of the algebra to allow more flexible stop conditions (e.g. convergence), accumulation of traversal history (such as with the delta attribute in alpha-algebra), definition and tests of rewriting rules, specification of a query language that can take full advantage of the algebra, and implementation of a distributed query processor.

Acknowledgments

This work was partially financed by the Microsoft Research FAPESP Virtual Institute (NavScales project), CNPq (MuZOO Project and PRONEX-FAPESP), INCT in Web Science (CNPq 557.128/2009-9) and CAPES-COFECUB, with individual grants from CAPES and FAPESP (Proc. 2012/15988-9 and 2014/01419-8). The opinions and conclusions expressed in this work do not necessarily reflect the funding agencies'.

6. REFERENCES

[1] R. Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *IEEE Trans. on Softw. Eng.*, 14(7):879, July 1988.

[2] P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM Trans. Database Syst*, 37(4):31, 2012.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[4] E. F. Codd. A relational model of data for large shared data banks. *Comm. ACM*, 13(6):377–387, June 1970.

[5] M. Consens and A. O. Mendelzon. Graphlog: a visual formalism for real life recursion. In *In Proceedings of the Ninth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 404–416, 1990.

[6] L. Costa, O. Oliveira Jr, G. Travieso, F. Rodrigues, P. Boas, L. Antiqueira, M. Viana, and L. Rocha. Analyzing and modeling real-world phenomena with complex networks: A survey of applications. *Advances in Physics*, 60:329–412, 2011.

[7] R. Cyganiak. A relational algebra for SPARQL. Technical Report HPL-2005-170, Hewlett Packard Laboratories, May 21 2005.

[8] F. Frasincar, G.-J. Houben, R. Vdovjak, and P. Barna. RAL: An algebra for querying RDF. *World Wide Web*, 7(1):83–109, 2004.

[9] L. Gomes-Jr, L. Costa, and A. Santanchè. Querying complex data. Technical Report IC-13-27, Institute of Computing, University of Campinas, October 2013.

[10] L. Gomes-Jr, R. Jensen, and A. Santanchè. Towards query model integration: topology-aware, ir-inspired metrics for declarative graph querying. In *GraphQ-EDBT*, 2013.

[11] L. Gomes-Jr and A. Santanchè. The Web Within: leveraging Web standards and graph analysis to enable application-level integration of institutional data. *to appear in Transactions on Large Scale Data and Knowledge Centered Systems*, 2014.

[12] O. Hassanzadeh and M. Consens. Linked movie data base. In *Proceedings of the 2nd Workshop on Linked Data on the Web (LDOW2009)*, 2009.

[13] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed graphlab: A framework for machine learning and data mining in the cloud. *Proc. VLDB Endow.*, 5:716–727, 2012.

[14] M. A. Rodriguez and P. Neubauer. The graph traversal pattern. *CoRR*, abs/1004.1001, 2010.

[15] R. Varadarajan, V. Hristidis, L. Raschid, M.-E. Vidal, L. D. Ibáñez, and H. Rodríguez-Drumond. Flexible and efficient querying and ranking on hyperlinked data sources. In *EDBT*, pages 553–564, 2009.

[16] P. T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.