Dagger: *D*igging for Interesting *Agg*regates in *R*DF Graphs

Yanlei Diao*, Ioana Manolescu*, and Shu Shang*

*Ecole Polytechnique, Inria, and Université Paris Saclay, France

Abstract. RDF is the format of choice for representing Semantic Web data. RDF graphs may be large and their structure is heterogeneous and complex, making them very hard to explore and understand. To help users discover valuable insights from RDF graph, we have developed Dagger, a tool which automatically recommends *interesting aggregation queries* over the RDF graphs; Dagger evaluates the queries and graphically shows their results to the user, in the ranked order of their interestingness. We propose to demonstrate Dagger to the ISWC audience, based on popular real and synthetic RDF graphs.

1 Introduction

RDF (the Resource Description Framework) is the W3C standard for describing interconnected *resources*, by specifying the *values* of their *properties*. Given that property values may also be resources, RDF data is organized in *graphs*; within a graph, some resources may have one or more *types*, or they may lack types.

RDF graphs may be very large and their structure complex and heterogeneous, thus finding interesting information they comprise may be hard. We developed Dagger a system which automatically identifies *the most interesting insights* in a given RDF graph G, defined as *RDF aggregate queries* evaluated over G; Dagger ranks such insights in the decreasing order of their interest, evaluates and plots the most interesting ones as bar charts, and shows them to the user. Figure 1 exemplifies interesting aggregates selected and visualized by Dagger in a popular RDF graph.

Dagger is the first system to support RDF graph discovery by *recommending interesting aggregation queries*. Below, we formalize the problem and present Dagger's solution, then outline the demonstration scenario, and briefly discuss related works.

2 The insight selection problem

An **RDF graph** G is a set of *triples*, typically denoted (s, p, o), where s is called subject, p is the property, and o is the object, or value of the property p of s. s and p are unique resource identifiers (URIs), or *blank nodes* (a special form of unknown/unspecified nodes). The special property $rdf{:}type$ (τ, for short) allows a form of resource typing, e.g., the triple $(s_1, \tau,$ *Student*$)$ states that s_1 is a person; a resource may have one or several types, or it may lack types. Property values can be URIs, blank nodes, strings, integers, floating point numbers, dates etc. Values may be typed explicitly, as in "$12.5\hat{}\hat{}$xsd:decimal", but may also appear simply as "12.5". An *ontology* is sometimes associated to an RDF graph, to describe its semantics, e.g., stating that any *Student* is a *Person*; this may lead to implicit triples, e.g., $(s_1, \tau,$ *Person*$)$ in our example. An RDF graph is *saturated* if it contains all its implicit triples. Below, we assume G is saturated.

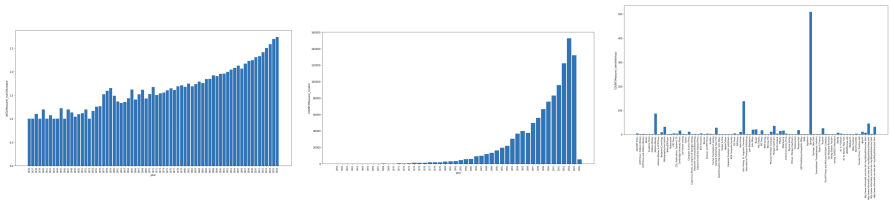


Fig. 1: Dagger-selected aggregates in DBLP (http://dblp.uni-trier.de/) bibliographic data. From left to right: the average number of authors of an article, per year (from 2 to 5); the number of conferences per year (up to 22.000); the number of books per publisher (highest for Springer).

We consider **RDF aggregation queries** which are a subset of the RDF analytical queries introduced in [3] and also of W3C's SPARQL 1.1 aggregation queries. On a bibliography RDF graph, sample queries are (q_1) : "For each year, how many authors have published conference papers that year" and (q_2) : "For each year, what is the average number of authors of conference papers published that year".

Generally, an RDF aggregation query $q = \langle f, d, m, \oplus \rangle$ consists of three RDF queries f, d, m and an *aggregation function* $\oplus$. f determines the *facts*, or resources over which the aggregate is computed (in q_1 and q_2, the conference papers); d defines the *dimension* (in q_1 and q_2, the year); m specifies a *measure* (the authors in q_1, and the number of authors in q_2); while the latter is typically not present in publication graphs, Dagger *derives* such information. Finally, the aggregation function $\oplus$ is *count(distinct(·))* in q_1 and *average* in q_2; $\oplus$ ranges over the set $\Omega = \{$*count*, *avg*, *sum*, *min*, *max*$\}$, possibly combined with a *distinct*. The *result* of q on a graph $\mathtt{G}$, denoted $q(\mathtt{G})$, is the set of pairs $\{(x, \oplus\{m_j \mid \exists f_i \in f(\mathtt{G}), f_i.d = x, f_i.m = m_j\})\}$, where f_i is a resource obtained by evaluating f on $\mathtt{G}$, $x = f_i.d$, respectively $m_j = f_i.m$ is a result of evaluating d, respectively m on f_i, and $\oplus\{\cdot\}$ is the result of calling $\oplus$ on an input set. For instance, $q_1(\mathtt{G})$ may be $\{(2015, 1200), (2016, 2420), (2017, 3760)\}$ while $q_2(\mathtt{G})$ may be $\{(2015, 2.5), (2016, 2.4), (2017, 2.5)\}$.

While relational aggregation queries are well-known, RDF graph heterogeneity makes our queries different in many respects. (i) A fact may lack the dimension (for instance, a paper may lack its publication year): such a fact does not contribute to the query. (ii) A fact may have several values along d, e.g., if the dimension is author affiliation and authors have multiple affiliations; such a fact contributes to the aggregation group of each of its dimension values. (iii) A fact may lack a measure, e.g., the institution may be unspecified for an author. If $\oplus$ is *count*, such a fact contributes with a count of 0; otherwise, it does not contribute to the query result. (iv) A fact may have several values for m, e.g., a paper with several authors; $\oplus$ then applies over the complete set of measure values obtained for the facts having the same value for the dimension [3].

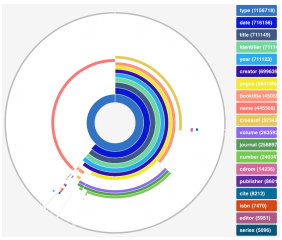
The interestingness $I(q, \mathtt{G})$ of a query q on a graph $\mathtt{G}$, where $q(\mathtt{G})$ is the set of pairs $\{(x_1, \oplus_1), \ldots, (x_n, \oplus_n)\}$ for some $n \geq 1$, is a measure of the set of values $\{\oplus_1, \ldots, \oplus_n\}$. We have mostly experimented with I_1 defined as the *variance* (second statistic moment) [1] of the value set; high variance reflects a strong difference among values, which may correspond to a trend, and/or to spikes in the data, as illustrated in Figure 1. In our example, $q_1(\mathtt{G})$ is more interesting than $q_2(\mathtt{G})$, as there is more variation in $\{1200, 2420, 3760\}$ (strong growing trend) than in $\{2.5, 2.4, 2.5\}$. Dagger also supports the interestingness measures *skew* I_2 and *kurtosis* I_3 [1] of the measure value set.

The problem considered by Dagger is: given a graph G, an integer k, and an interesting measure I, find the k most interesting aggregate queries $q(\mathtt{G})$. The search space is huge, as many f, m, d queries can be asked on a graph G. Below, we present Dagger' practical approach, which has allowed finding many interesting aggregates efficiently.

3 Dagger's architecture and algorithms

We have built Dagger as a Java and Python application on top of the PostgreSQL 9.6 relational database server; the GUI is based on Jupyter notebooks (http://jupyter.org). Dagger stores G as a collection of tables in PostgreSQL, one table for each class and property. While PostgreSQL bears the brunt of aggregate query evaluation, Dagger carefully translates between the RDF world and PostgreSQL, to faithfully reflect the RDF-specific features of our aggregation queries we explained above.

Dagger selects interesting insights through the following steps: **1.** identify several *candidate fact sets*; **2.** for each candidate fact set, explore a set of *candidate dimensions*; **3.** for each (f, d) pair, explore *candidate measures* m; **4.** for each (f, d, m) combination, pick *applicable* $\oplus$ functions; **5.** evaluate the interestingness of the queries resulting from steps 1. to 4. and retain those with top k values. In the sequel, we outline each step; for simplicity, we use f, d, m to refer to the respective queries *or* to their results on G.



1. Candidate fact sets (i) We identify the classes of G (that is, the set of all URIs c such that $(x, \tau, c) \in \mathtt{G}$), and for each such c, the set f_C of all resources of type c in G is a candidate fact set; (ii) given a user-specified support threshold t_{supp} between 0 and 1, for any set of properties $P = \{p_1, p_2, \ldots, p_n\}$ such that at least t_{supp} of the triple subjects in G have (at least once) each property in P, the set f_P of G resources having the properties in P is a candidate fact set. t_{supp} can be tuned using a J3S GUI (see sample screenshot above for illustration) showing the fraction of G resources having various set of properties. Internally, these are computed by `cube` SQL queries evaluated by PostgreSQL.

2. Candidate dimensions We look for dimensions among *properties which all facts in f have*, as facts lacking dimensions do not contribute to $q(\mathtt{G})$. Further, such a property should have *much fewer distinct values than there are facts*, otherwise aggregation groups may be too many and too small. We use a distinct-value threshold $t_{dist} = 0.05$ to retain only properties whose values are shared on average by at least $1/t_{dist}$ facts. Next, for each property p_d thus identified, we *materialize as a derived dimension* the *number (count)* of p_d values of each candidate fact. We denote this $\#p_d$, and store it in a separate derived property table; both p_d and $\#p_d$ serve as candidate dimensions.

An interesting question is the *order* among the values of a candidate dimension. For instance, the visible yearly trends in Figure 1 (left and center) would be lost with a different bar order. To determine the best order, for each candidate dimension property p_d, we *detect the majoritary data type* (string, integer, date, double) among the values of p_d for all candidate facts. If a value is typed, we use that information, otherwise we attempt to cast it to integers, floats etc. and record the most specific type to which the cast succeeded. Again we use a threshold t_{type}, and if at least t_{type} of the values of facts f match a type (other than string), we store the typed values of $f.p_d$ in a separate

PostgreSQL table, with the appropriate types. In Figure 1, Dagger found that years are integers, and plotted query results in a visually compelling fashion.

3. Candidate measures Given f and d, a candidate measure is a property p_m which all f facts have; which is not d; and such that d is not $\#p_m$, and p_m is not $\#d$. To prepare the ground for aggregation function selection (see below), we detect the type of each p_m property thus identified, just like we did for p_d in step 2.

4. Candidate aggregate functions Given (f, d, m), candidate functions $\oplus$ are chosen based on the type of m values: all Ω functions apply to numeric types; *min*, *max*, *count* and *count(distinct)* apply to dates; *count* and *count(distinct)* apply to strings.

5. Ranking by interestingness Dagger evaluates each $q = (f, d, m, \oplus)$ using PostgreSQL, computes $I(q)$ and plots the k most interesting query results.

4 Demonstration scenario

Users interacting with Dagger may: (i) select an RDF dataset among those pre-loaded; we plan to use a dozen datasets, real (DBLP, Linked Clinical Trials, BBC Music Programs, DBPedia etc.) and synthetic (LUBM and BSBM benchmark data); users may also point us to datasets of their choice; (ii) select an interestingness measure I (variance, skew, or kurtosis); (iii) visualize the resulting top-k insights, either pre-computed, or computed on the spot; (iv) modify the pre-set values of t_{supp}, t_{dist} and t_{type} and trigger the recomputation of the top-k insights.

A video of our demo appears at `https://team.inria.fr/cedar/projects/dagger`.

5 Related work

Tools comparable to Dagger have been developed in relational data warehouses, when the possible aggregation dimensions and measures are known [6,7]; [6] also investigates how to efficiently derive an interesting aggregate from another. In contrast, Dagger applies on a heterogeneous RDF graph in which no facts, dimensions, and measures are known, and recommends candidate queries through a set of effective heuristics. [2] selects interesting cuboids from an "aggregated graph", pre-built by analysts from relational or any other kind of data, counting associations between two resources of the same kind (e.g., social network interactions between users), while Dagger applies directly on (any) RDF graph. Dagger complements many other RDF exploration techniques such as mining, summarization, faceted search, statistics extraction etc. [4,5]. Ongoing work on Dagger aims to optimize insight computations along the lines of [7].

Acknowledgements Zheng Zhang has developed the J3S property set analysis GUI.

References

1. *Encyclopedia of Statistical Sciences*. Wiley, 2014.
2. D. Bleco and Y. Kotidis. Entropy-based selection of graph cuboids. In *GRADES*, 2017.
3. D. Colazzo, F. Goasdoué, I. Manolescu, and A. Roatiş. RDF analytics: lenses over semantic graphs. In *WWW*, 2014.
4. A. K. Joshi, P. Hitzler, and G. Dong. Logical linked data compression. In *ESWC*, 2013.
5. S. Khatchadourian and M. P. Consens. Understanding billions of triples with usage summaries. *Semantic Web Challenge*, 2011.
6. B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *SIGMOD*, 2017.
7. M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13), 2015.