Spade: A Modular Framework for Analytical Exploration of RDF Graphs

Yanlei Diao[1,2] Paweł Guzewicz[1,3] Ioana Manolescu[1,3] Mirjana Mazuran[1,3]
[1] LIX (UMR 7161 and Ecole polytechnique), France
[2]University of Massachusetts Amherst
[3] Inria, France

yanlei.diao@polytechnique.edu, {pawel.guzewicz,ioana.manolescu,mirjana.mazuran}@inria.fr

ABSTRACT

RDF data is complex; exploring it is hard, and can be done through many different metaphors. We have developed and propose to demonstrate Spade, a tool helping users discover meaningful content of an RDF graph by showing them the results of *aggregation (OLAP-style)* queries automatically identified from the data. Spade chooses aggregates that are *visually interesting*, a property formally based on statistic properties of the aggregation query results.

While well understood for relational data, such exploration raises multiple challenges for RDF: facts, dimensions and measures have to be *identified* (as opposed to known beforehand); as there are more candidate aggregates, assessing their interestingness can be very costly; finally, *ontologies* bring novel specific challenges but also novel opportunities, enabling *ontology-driven exploration* from an aggregate initially proposed by the system.

Spade is a *generic, extensible framework*, which we instantiated with: (*i*) novel methods for enumerating candidate measures and dimensions in the vast space of possibilities provided by an RDF graph; (*ii*) a set of aggregate interestingness functions; (*iii*) ontology-based interactive exploration, and (*iv*) efficient early-stop techniques for estimating the interestingness of an aggregate query.

The demonstration will comprise interactive scenarios on a variety of large, interesting RDF graphs.

1. INTRODUCTION

RDF graphs are increasingly being published and shared, as part of the Linked Open Data (LOD) movement. However, given their size, heterogeneity and complexity, their information content is hard to grasp, in particular for non-expert users. Figure 1 shows a portion of an RDF graph about food, recipes, ingredients, etc. from the **Foodista** dataset[1]. It illustrates the heterogeneity frequently encountered in RDF. Here, n_4 and n_9 are labeled with the *Recipe* type; n_2 and n_{11} are *Food*; n_3 lacks a type, but its outgoing properties suggest that it is a *Recipe*; n_8's outgoing properties hint that it is probably *Food*. This example also shows that logically similar resources do not always have the same structure, e.g., n_4, n_5 and n_9 are all *Recipe*, have a *title* and a *country*, but only n_5 has a *depiction*. Similarly, n_5 has two *categories* while n_4 has only one and n_9 none at all.

This small portion of data shows RDF's flexibility and heterogeneity. Its flip side is the difficulty, especially for casual users, in approaching the data, that is, viewing and understanding its content and discovering, at a glance, useful insights. This fundamental issue indicates a need for *automatic* (or *very low-effort*) *tools* to guide users in exploring RDF graphs for insight discovery.

The automatic extraction of interesting aggregates is one among the data exploration techniques that have been studied recently [11, 12, 13, 9]. All of these systems assume a fixed relational data warehouse schema, which is not available for RDF graphs. Some recent works [2] consider graphs, and use entropy to identify interesting portions of the graph. However, they assume that the graph has a very regular and simple structure, and that a cube over the data has been computed. In contrast, we seek to work with any graphs, and aim to compute only interesting aggregates. Dagger [4] was an earlier version of our work that was designed to identify only mono-dimensional aggregates and evaluate them in a naive time-consuming fashion. In [10] sampling was used to speed up the evaluation process but the aggregate selection accuracy is sensitive to the chosen sampling strategies. As we show below, our work explores rich multi-dimensional aggregates from RDF graphs, which drastically expands the computation space and calls for novel techniques.

Spade outline We have developed Spade, a system that provides users with *top-k most interesting aggregate queries identified automatically in a given RDF graph.* For instance, from the Foodista dataset, Spade recommends "*the number of recipes by category*" (Figure 2(a)) and "*the number of recipes by ingredient and category*" (Figure 2(b)). Figure 2(b) is *interesting* as there is a peak of *sugar* usage in *fruits* and *desserts* (the two yellow cells), leading to a huge variance among ingredients/categories. Users may further *refine* a given aggregate, e.g., selecting the *dessert* category allows zooming on its semantic subclass *thaiDesserts* etc.

Inspired by classical data warehousing, we define a (potentially multi-dimensional) **aggregate** centered around a set of *facts*, which are nodes of the RDF graph, e.g., recipes in the above example. An aggregate comprises: (*i*) a set of *dimensions* along which the facts are studied, e.g., category and country; (*ii*) a set of *measures*, e.g., the time to cook, the ingredients, and (*iii*) a set of *aggregate functions* which define how the measures should be aggregated, e.g., the average cooking time, the number of ingredients, etc. Given an integer k, identifying automatically the k most interesting aggregates in a given RDF graph raises many challenges. We outline them below, and highlight Spade's contributions.

1. Identifying candidate facts sets In the absence of a relational data warehouse schema, it is not clear where "the facts" are in an RDF graph. Spade uses a variety of methods to identify *candidate fact sets* (CFS, in short). Be-

[1]`https://old.datahub.io/dataset/foodista`

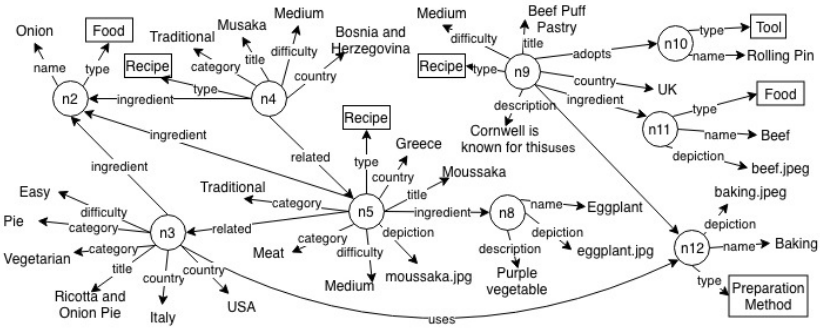

Figure 1: Sample data inspired from the Foodista dataset.

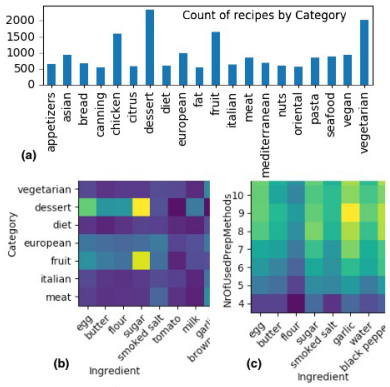

Figure 2: Aggregates from Foodista.

sides *type* and *user input* based methods as in Dagger [4], Spade also identifies CFS *automatically based on their incoming and outgoing properties*, with the help of an RDF graph summary [5]. This new feature is crucial as in some graphs, e.g. Foodista, many nodes of interest have no type.

2. Identifying candidate dimensions and measures The dimensions and measures, according to which a given CFS is analyzed, need to be automatically found. Besides the properties or *derived* properties (e.g., "the number of ingredients in each recipe") of CFS nodes, Spade is also capable of using *keywords* extracted from text nodes as dimensions, e.g., "group recipes by the keywords in their titles" ("Ricotta" in the title of n_3), or *paths* as dimensions or measures, e.g., "group recipes by the names of their ingredients" ("Beef" is the name of n_{11}, an ingredient for n_9). In addition, multi-dimensional aggregates that combine the variety of dimensions significantly increase the richness, as well as the computation complexity, of such RDF exploration.

3. Interestingness functions There is no standard view of what makes an aggregate "interesting"; intuitively, it corresponds to a trend, peak, outlier, etc. in the aggregation query result. To capture this, we rely on statistic moments (variance, skew or kurtosis) but we provide an open architecture to accommodate more measures as user needs arise.

4. Efficient aggregate evaluation The above extensions lead to an *explosion in the number of aggregates* to be evaluated in order to find the most interesting ones. To address this, Spade employs key techniques including new extensions of lattice-based computation for multi-dimensional aggregates [14], and new extensions of online aggregation [8] to stop the computation of an aggregate as soon as it is deemed not in the top-k, which distinguishes from our prior work [4, 10] with *formal guarantees* of the correctness of pruning.

5. Handling and exploiting semantics An RDF graph may feature an *ontology*, stating relationships that may exist between types and properties, e.g., any *FrenchRecipe* is a *Recipe*, or a recipe with a *correction* is a *ModifiedRecipe*. An ontology may lead to *implicit data*: for instance, if r_1 is of type *FrenchRecipe* in an RDF graph, r_1 is also of type *Recipe*, even though this is not explicitly part of the graph. Spade handles implicit data when exploring CFS, dimensions and measures by relying on an ontology-aware RDF platform. Further, based on the ontology, users may *navigate* from an interesting aggregate to another through *generalization*, e.g., changing the CFS from *FrenchRecipe*s to *Recipes*, or on the countrary through *specialization*, e.g., going from *FrenchRecipe*s to *BourgogneRecipe*s to see how the aggregate results (and thus, its interestingness) vary.

2. Spade DESIGN AND ARCHITECTURE

We explain the concepts and operations underlying Spade (Figure 3), before outlining our algorithms.

2.1 Spade Concepts

A **candidate fact set (CFS)** is a set of RDF resources on which we may build an interesting aggregate; we call *candidate fact* (CF) a member of this set.

We call **attribute (A)** a property attached to CFS nodes, which can be used as a dimension (to group facts by the value of this property) or as a measure (for each fact group, the values of this property will be aggregated). An attribute is either an RDF property (denoted **P**) that some CFS nodes have, or a *derived property* (denoted **DP**). As **aggregation function (Agg)**, we rely on the common set of sum, average, count, max and min.

A **multi-dimensional aggregate (MDA)** is determined by a CFS, one or several dimensions (which are attributes), a measure (also an attribute), and an aggregation function. The MDA's semantics [1] accounts for: (i) dimensions and/or measures which may be missing for some CFs; such CFs do not contribute to the aggregate; (ii) each attribute that is defined multiple times for some CFs; it contributes to several aggregate groups (if used as a dimension) or contributes with several values (if used as a measure).

An **interestingness function (IF)** takes as input a set of $(\bar{d}_i, Agg_i)$ pairs; each $\bar{d}_i$ is a vector of dimension values (e.g., vegan Italian recipes) and Agg_i is the result of the aggregation function on the corresponding group, e.g., the number of such recipes. An IF returns a positive real number, reflecting some measure of interestingness of the aggregate.

2.2 Interesting MDA Selection Framework

Spade is designed as a framework (see Figure 3) where *operations* are chained to identify the most interesting MDAs. We implemented it as a Java-based tool (circa 40 classes and 10K lines of code) running on top of OntoSQL[2], a Java and Postgres-based platform we developed [3] for efficient RDF storage and query answering. Spade stores RDF graphs in OntoSQL; it builds an RDFQuotient [5] summary; it also finds CFS, enumerates and analyzes their properties, creates and stores derived properties directly in Postgres.

[2] `https://ontosql.inria.fr`

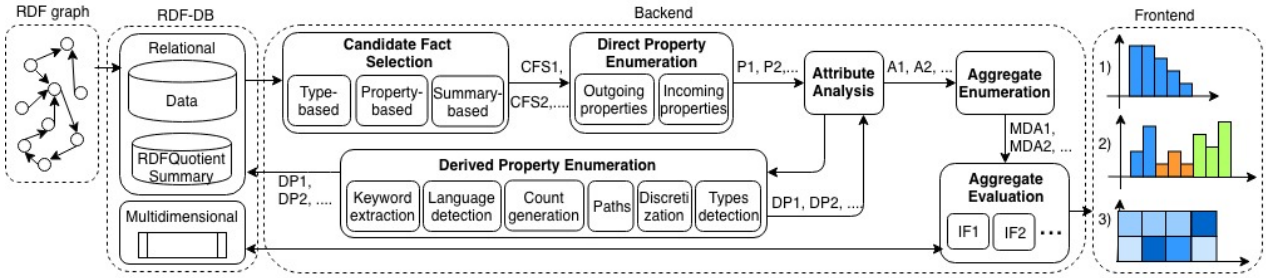

Figure 3: The architecture of Spade.

For exploration, the first operation, after an RDF graph has been loaded, is **Candidate Fact Selection**. Spade considers the following CFS: (*i*) for each type T in the graph, the set of resources of type T; (*ii*) for a (user-specified) set of properties, all the resources having those outgoing properties; (*iii*) each set of resources identified as equivalent by the RDFQuotient summary. In the third case, the summary divides RDF graph nodes in *equivalence classes* based on flexible criteria on their incoming/outgoing properties and/or their types (when present). Nodes in the same class tend to have many common properties, making them interesting candidates to be analyzed together as a CFS. Other CFS selection criteria, e.g. based on another summary or the results of a given query, can be easily plugged in here.

Next, for each CF, **Direct Properties** and **Derived Properties** are **Enumerated** to obtain all possible CF attributes. The (possibly missing, or multiple) values of these attributes are computed for each CF and stored in the RDF database (RDF-DB in Figure 3). A *null* is used to record absent values. Spade derives the following properties: (*i*) *counts*, e.g., how many ingredients a recipe has; (*ii*) *keywords*, e.g., if a recipe is called "Apple and Cinnamon Rolls", we consider the keywords "Apple", "Cinnamon" and "Rolls" as values of the property *kw-title* for the same resource; (*iii*) the *language* obtained by analyzing text attributes; (*iv*) *paths*, e.g., a recipe with an ingredient whose name is "Beef", has the attribute *ingredient-name* with the value "Beef"; (*v*) *types, sub-types and super-types*: when CFS are typed, we pre-compute (from the ontology which may come with the graph) their possible more/less general types, to prepare for semantic navigation; (*vi*) *discretized values*, e.g., transforming the minutes needed to cook a recipe into buckets: [0-10], [10-20], [20-100], [>100] etc.

Then we have **Attribute Analysis** which computes a set of *statistics*: number of CFs having an attribute, type (e.g., String, Integer, Date) of the attribute values, their number of distinct values, lowest and highest value etc. Spade exploits these statistics in various ways in different steps, e.g., to guide the choice of dimensions, measures and aggregation functions, and to improve the aggregate evaluation.

Next, we move to **Aggregate Enumeration** to find combinations of a set of dimensions, a measure, and an aggregation function. To avoid meaningless aggregates, e.g., "the number of ingredients for each recipe ingredient", we apply rule-based pruning to restrict the measure to differ from each dimension, choose the aggregate function based on the measure type (e.g., only average numeric measures), etc.

Finally, in **Aggregate Evaluation** the enumerated candidate aggregates are actually evaluated and their *interestingness* measures are computed based on the chosen IF. Spade supports variance, skewness and kurtosis (second to fourth statistic moments) computed over the set of Agg_i values. In Figure 2, (b) and (c) show two bi-dimensional aggregates. There is a cell for each dimension value pair, e.g., ingredient="sugar" and category="dessert". The colors of the cells represent the measure, i.e., the number of recipes. Values in (b) vary a lot, e.g., the two yellow cells represent peaks of values; thus this aggregate is considered interesting. In contrast, in (c), cells have a very mixed color pattern, and hence this result is deemed not interesting. The interestingness of highly dimensional aggregates (3, 4 or more dimensions), when statistic moments are considered, is harder for users to interpret. More sophisticated strategies for insight detection, e.g., such as proposed in [11], can be plugged in.

2.3 Optimization for Improved Efficiency

In this section, we discuss optimization of expensive operations in Spade. Aggregate evaluation is a very expensive operation because our rich variety of ways to derive properties and enumerate multi-dimensional aggregates leads to an explosion of the space of aggregates for evaluation. Thus, we introduce novel techniques to improve efficiency.

Lattice-based computations MDAs can be captured in lattices [7], e.g., the attributes {category, difficulty, keyword} lead to a lattice of $2^3 = 8$ nodes (that is, MDAs), from the top (group by the three attributes) to intermediary levels (group by two, successively by one attribute) to the bottom node that does not group at all. To optimize the computation we build on the efficient technique [14], which takes a *known* set of dimensions, a *fixed* measure and an aggregate function, and computes the whole lattice in one pass over the data, *sharing* computation across all the nodes. The technique also relies on a smart spatial data storage layout. Our new RDF scenario raises several challenges: (*i*) a lattice's dimensions are not known at the start and hence we cannot pre-load the data as in [14]; (*ii*) we may recommend MDAs from different lattices; (*iii*) a given set of dimensions can be associated to different measures and aggregate functions.

We extend the technique described in [14] to address these challenges as follows. (1) We adapt maximal frequent pattern mining [6] to find, inside each CFS, the sets of dimensions to be used for MDAs. For recipes, the sets of frequent attributes can be: {{nrOfIngredients}, {country, category}, {category, difficulty, keyword}}. Each corresponds to a lattice of MDAs. (2) While materializing each lattice, we adapt the multidimensional data structure to keep track of several measures, e.g. the number of recipes, the maximum and average time of cooking, etc. (3) In the same step as (2), we aggregate not only along the measures, but also opportunistically along the dimensions, e.g., while computing the number of recipes in the {category, difficulty, keyword} lattice, we also compute the average difficulty per category, the number of keywords by category, etc. In this way we make the best use of each pass on the data and maximize

the number of MDAs that are computed.

Novel early-stop techniques are used to stop the evaluation of an aggregate as soon as we can determine (with high probability) that it will not be among the k most interesting. We build on top of [8], which *approximately* computes aggregation query results in a given *confidence interval*. Our problem is harder because we want to approximate *the IF computed over the aggregation results*. Using advanced statistical tools such as the Delta Method[3], we construct confidence intervals for variance, skewness and kurtosis over partially evaluated results of candidate aggregates. This allows us to prune a large set of aggregates early, and focus computation on the interesting ones.

While the above techniques significantly improve the efficiency of aggregate evaluation, Spade also optimizes the *order* in which all tasks in Figure 3 are executed.

Task-based execution Spade relies on a priority queue where small-granularity *tasks* are added and taken from for execution. For instance, *SummaryBasedCFSenumeration()* is one such task, which, upon execution, adds to the queue a task *EnumerateAttributes(CFS$_i$)* for each enumerated CFS$_i$. Each such task has a *priority*, derived from the number of resources in CFS$_i$. Next, *EnumerateAttributes(CFS$_i$)* adds to the queue tasks such as *EnumerateDirectProperties(CFS$_i$)* and *ExtractKeywords(CFS$_i$, a)*, where a is a property; the latter implements derived property enumeration.

This fine-grained decomposition of operations also provides much flexibility and modularity as it allows to: (*i*) explore in parallel different MDAs; (*ii*) show *some* MDAs fast, even if they have not all been explored yet (informing users that the result may change); (*iii*) flexibly distribute a given time budget between the different operations, e.g., exploring more linguistic features if a given CFS has few properties which are quickly computed etc. Task priorities are set in a dynamic fashion, and can be controlled between a depth-first style exploration of the space to a breadth-first one. We use a combination of timers (how much to spend on tasks) and division of a given time budget, in order to set the timers of the tasks on the queue and thus control the execution order.

3. DEMONSTRATION SCENARIOS

We will use publicly available graphs, including: *Foodista*; *NobelPrizes*[4] (90K triples) and *DBLPArticle* (20M triples)[5]. The RDFQuotient summary is shown as a first overview of the data as it provides a quick glance on their content and might inspire users during the demonstration steps. Our interactive scenarios enable users to get insights from large RDF graphs, refine them and deepen their analysis through semantic-based roll-ups and drill-down.

• **MDA recommendation** Users choose a dataset, an IF and k, the number of desired aggregates, and visualize the top-k results. A specific CFS, among those found by Spade, can also be chosen.

• **Customized CFS selection** Guided by the summary, users can select nodes and/or properties to specify a customized set of data. Resources that comply with the given criteria will be considered as a CFS and processed by Spade.

• **Refining discovered insights** Users can refine a MDA returned by Spade by focusing on some of its groups, e.g., from a recipe aggregate with dimensions {*Category, Difficulty, Keyword*}, they may drill-down to specific values of one or more of the dimensions and visualize the results.

• **Ontology-driven navigation** Let $(S, \bar{D}, m, Agg)$ be an interesting MDA returned by Spade over the CFS S, dimensions $\bar{D}$, with the measure m and aggregation function Agg, such that a large fraction ($> 80\%$) of the S nodes have a common RDF type, e.g., *FrenchRecipe*. Recall the ontology triples we assumed, stating that *Recipe* has the subclass *FrenchRecipe* which in turn has the subclass *BourgogneRecipe*. Users may request navigating through *generalization* to replace S with all the resources of type *Recipe*, evaluate the same aggregate on this modified CFS, and visualize the result. Similarly, they may request *specialization* to replace S with the nodes having the type *BourgogneRecipe*. The variance in the number of *FrenchRecipe*s by ingredient is high because of peaks in the use of sour cream, cheese, etc. If we generalize to *Recipe*s the variance is much lower; if we specialize to *BourgogneRecipe*, the variance remains high. This is reminiscent of roll-up/drill-down, but differs in that the super/sub-type may not have the same dimensions due to the different attributes in the data. Therefore, navigation along the subclass structure can significantly change the set of MDAs. Similar to the navigation based on *subclass* relationships, *subproperty* can be used to replace a dimension (or the measure) defined by a graph property with a more general/more specific related property from the data.

[3] `https://en.wikipedia.org/wiki/Delta_method`
[4] `https://old.datahub.io/dataset/nobelprizes`
[5] `http://www.rdfhdt.org/datasets/`

4. REFERENCES

[1] E. Akbari-Azirani, F. Goasdoué, I. Manolescu, and A. Roatis. Efficient OLAP Operations For RDF Analytics. In *DESWeb*, Apr. 2015.

[2] D. Bleco and Y. Kotidis. Using entropy metrics for pruning very large graph cubes. *Information Systems*, 81, 2019.

[3] D. Bursztyn, F. Goasdoué, and I. Manolescu. Teaching an RDBMS about ontological constraints. *PVLDB*, 9(12), 2016.

[4] Y. Diao, I. Manolescu, and S. Shang. Dagger: Digging for interesting aggregates in RDF graphs. In *ISWC Posters & Demonstrations and Industry Tracks*, 2017.

[5] F. Goasdoué, P. Guzewicz, and I. Manolescu. Incremental structural summarization of RDF graphs (demo). In *EDBT*, 2019.

[6] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *ICDM*, pages 163–170, 2001.

[7] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, 1996.

[8] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[9] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang. DeepEye: Creating good data visualizations by keyword search. In *SIGMOD*, 2018.

[10] I. Manolescu and M. Mazuran. Speeding up RDF aggregate discovery through sampling. In *BigVis*, 2019.

[11] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang. Extracting top-k insights from multi-dimensional data. In *SIGMOD*, 2017.

[12] 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.

[13] Y. Wen, X. Zhu, S. Roy, and J. Yang. Qagview: Interactively summarizing high-valued aggregate query answers. In *SIGMOD*, pages 1709–1712, 2018.

[14] Y. Zhao, P. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. In *SIGMOD*, 1997.