Ranking, Aggregation, and Reachability in Faceted Search with SemFacet*

Evgeny Kharlamov[1] Luca Giacomelli[2] Evgeny Sherkhonov[1]
Bernardo Cuenca Grau[1] Egor V. Kostylev[1] Ian Horrocks[1]

[1]University of Oxford, UK [2]Sapienza University of Rome, Italy

1 Introduction

Motivation. Faceted search is a prominent search paradigm that became the standard in many Web applications including online shopping and real-estate portals, where users can progressively narrow down the search results by applying filters, called *facets* [9] which are organised in faceted interfaces. Faceted search has also been proposed in the Semantic Web context for exploring and querying RDF graphs and a number of RDF-based faceted search systems have been developed (see [3,7,4,5,2,6] for an overview).

One of the main challenges that hampers usability of faceted search systems is *information overload* [9]: when the size of the faceted interface becomes comparable to the size of data over which the search is performed. The overload is already a challenge in the case of the classical faceted search and it becomes even stronger when faceted search is applied to RDF data. Indeed, observe that in both classical and RDF settings the search is over annotated entities, at the same time, in the latter case the *number* of annotations and possible values is typically much larger: *any* predicate occurring in an RDF data set can give a facet during a search, and *any* entity or value that it points to in the RDF data can become a value in this facet. At the same time, in the classical case the list of possible facets is typically predefined and controlled.

Moreover, differently from the classical case, in the RDF settings entities are also *interconnected*. Thus, in an RDF driven faceted interface one has to *nest facets* in order to reflect this interconnections. In Figure 1, left, one can see a possible way to nest facets which is implemented in our SemFacet system. This not only raises a non-trivial problem of how to arrange nested facets within an interface in an intuitive and user oriented way, but also leads to an unavoidable overload of the interface with various *paths of nested facets*. Thus, faceted navigation over RDF requires from users to be experts in the structure of the underlying RDF graph in order to be able to find the required facet which can be deeply nested.

In order to see why it might be hard to find a required nested facet, consider in Figure 2 an example RDF data. Assume that one is looking for a smartphone with the price within £500–£900 and the processor that is manufactured by a company with headquarters in Asia. One can see in Figure 2 that Samsung S8 is such a smartphone. At the same time, finding this smartphone via a faceted interface requires one to traverse the data via 4 nested facets (Figure 1, left).

SemFacet System. In our RDF-based faceted search system SemFacet we propose to address the information overload by

- ranking facets and their values and thus offering to users top-k most prominent facets and values;

* This work was supported by the Royal Society under a University Research Fellowship and the EPSRC under an IAA award and the projects DBOnto, MaSI^3, ED^3, and VADA.

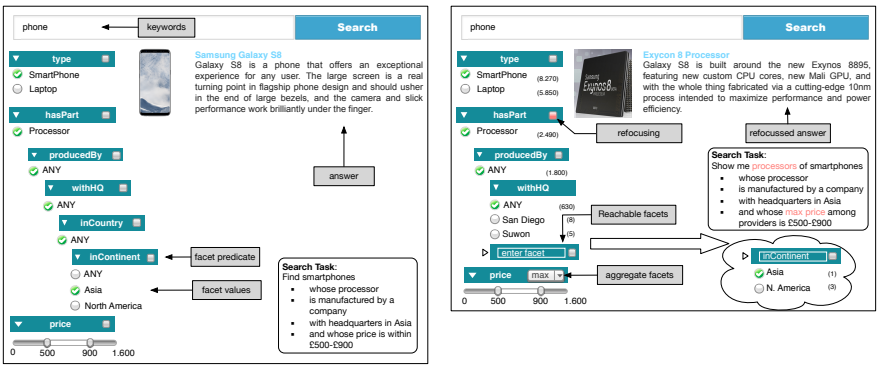

Fig. 1. Left: SemFacet with deep nesting over RDF data; Right: SemFacet enhanced with ranking, aggregation, and reachability

- minimising the number of values within a facet by first grouping them according to the corresponding entities and then aggregating them with the standard aggregate functions max, min, count, sum, avg;
- shortcutting paths of nested facets with the help of a reachability operator.

Observe in Figure 1, right, the SemFacet interface where all these three features are incorporated. The facets and values are now ranked and only the top 10 of them are displayed. The users can choose whether to look for a maximal price of entities by activating aggregate functions within facets with numerical values. In the screenshot the user selected max and thus the system is searching for smartphones whose *maximal price* across providers is within the range £500–£900. Finally, the users can enter the name of a desired predicate instead of choosing a facet value in order to ask the system to find a shortcut (a reachable facet). In the screenshot the user entered inContinent instead of selecting SanDiego or Suwon.

Demo Overview. The goal of the demo is to show the attendees how our novel features, that is, ranking, aggregation, and shortcutting, make hard faceted search—over RDF datasets that are highly interconnected and with many data values—easier. In order to experience the impact of these features the attendees will be able to explore the demo dataset using two versions of our SemFacet systems: with and without the features.

2 SemFacet System

We first give an overview of SemFacet with the focus on its novel components and then present technical details behind ranking, aggregation, and shortcuts.

System Overview. SemFacet is implemented in Java and available under an academic license [1]. In Figure 2 there is the architecture of SemFacet where the arrows denote the data flow between the systems' components.

On the client side, SemFacet has an HTML 5 based GUI consisting of three main parts: a free text search box for keywords, a hierarchically organised faceted interface, and a scrollable panel containing snippet-shaped answers. User keywords are sent by the client to the server, evaluated and this gives the initial faceted interface. User selections in the faceted interface are compiled into a SPARQL query using the *SPARQL query constructor* and then sent to the server for evaluation. The *snippet composer* and *facet composer* receive information about facets and answers that should be displayed to the user and update the currently displayed interface and query answers. The system updates the faceted interface incrementally: only the parts of the interface that are affected by users' actions are updated, which allows for a significantly faster response

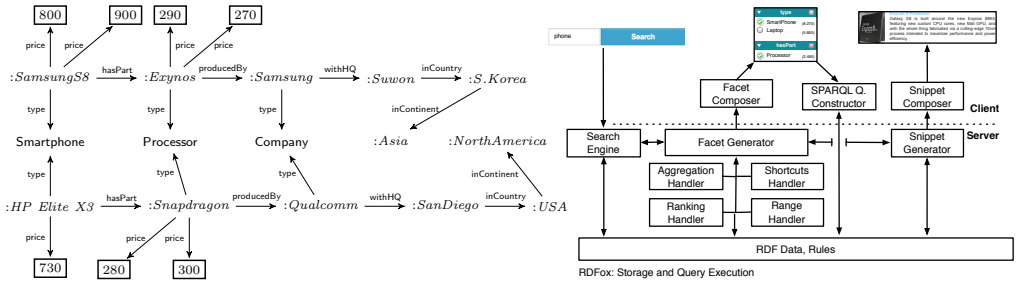

Fig. 2. Left: Example RDF graph about products; Right: Architecture of SemFacet

time. The user is able to do *refocusing*. This feature of SemFacet can be observed in Figure 1, right, where the user can click on a box in the hasPart facet to change the answers from phones to their parts, that is, from Samsung S8 to its processor Exycon 8.

On the server side, the system relies on an in-memory triple store RDFox [8] to store the inverted index, input RDF data, query answers, and all necessary auxiliary information such as materialisation rules which we discuss later in this section. One of the server components is an inverted index based full-text *search engine*; in order to ensure a better integration between full-text and faceted search and thus achieve good efficiency of SemFacet we implemented our own search engine. Another backend component is *snippet composer* that for given answer entities retrieves their textual descriptions, images, and links. The next component is *facet generator* that constructs faceted interfaces in response to user actions. This component is backed by four handlers: *aggregation*, *shortcuts*, *ranking*, and *ranges*. The range handler computes and stores left and right bounds for the range sliders for numerical facet values, like the price-slider in Figure 1, that correspond to the lower and upper bounds of possible values for this property name in the underlying RDF data.

Ranking facets. Whenever SemFacet updates the interface it should decide in which order to present relevant facets. This is done in two steps. First, we compute the set S of all relevant facets that should be displayed in the same level of nesting. Assume that S has n facets. Then, for each of $n!$ possible orderings of S we find the optimal order using our scoring function f and SemFacet displays the facets from S (or some of them) according to this order. Our function f computes the score of a given ordering of facet by combining the following three characteristics of the order: *(1)* selectivity of its facets, *(2)* diversity of its facets, and *(3)* nesting depth of its facets. In particular, for (1) we prefer those facets that are more selective, i.e., ticking values in the facet narrow down the search result more rapidly than doing so in other facets. For (2), we prefer those facets that lead to results which are not covered by selecting other facets. Finally, for (3), we prefer facets that allow deeper nesting thus allowing explore the graph structure of the underlying data. We now define the functions corresponding each of (1)-(3) and then combine them into the final scoring function f.

Ranking facet values. Besides ranking of facets, it is important to rank facet *values* as well. We adopt the count-based ranking (see also [10]): facet values that lead to a larger number of results are ranked higher. Although the computation of counts is conceptually trivial, the main challenge is in their update. Indeed, the integers associated to each facet value in the interface must be updated every time the user interacts with the faceted interface without affecting the performance of the system. Additional challenges come from *(i)* graph structure of the data and *(ii)* interplay of conjunctive and disjunctive interpretation of facet values. In our experience, implementing the straight-

forward approach to updating counts leads to a significant increase of the user interface response time. To mitigate this problem, we adopted a multi-threading solution: each thread receives a set of facet values for which counts need to be updated and, additionally, the load among the threads is balanced. For instance, we avoid the situation when one thread is busy with updating counts for values with a high number of results only, while another gets away with updating counts for values with a low number of results. This approach led to a significant response time decrease.

Aggregation. Recall that every interface update performed by the user (i.e., refocusing, selection of a facet or a facet value) results in formulating a corresponding SPARQL query on the fly that is then issued to RDFox. Then the interface is updated (i.e., with search results and available facets at this point) depending on the result of this query. When aggregate facets are considered, it is possible to follow the same approach and formulate *aggregate* SPARQL queries. However, we decided to do materialisation of aggregate information at loading time instead since (1) RDFox, our back-end system, supports efficient materialisation of aggregate information and, more importantly, (2) non-aggregate SPARQL queries are usually faster to answer which is essential for responsive user interface updates.

Reachability. SemFacet provides the shortcut functionality described in Section 1. In the user interface, SemFacet offers a search box within each facet that allows users to search for reachable facets. As the user has typed in a facet name F in the box, the system checks if such a facet is reachable from the current facet. For this we perform breadth-first search to find all reachable nodes with an outgoing property F and we store corresponding witnessing paths. These paths are important in constructing the corresponding SPARQL query for fetching possible facet values for F and for further facet navigation. For a faster response time, we predefine a parameter B and check facet reachability up to length B instead. In our example, in Figure 1 (right) the user can search for the inContinent facet, which in this case is reachable within 2 steps, and then select 'Asia', thus selecting processors produced by an asian company.

References

1. SemFacet Project Page. http://www.cs.ox.ac.uk/isg/tools/SemFacet/
2. Arenas, M., Cuenca Grau, B., Kharlamov, E., Marciuska, S., Zheleznyakov, D.: Towards Semantic Faceted Search. In: Proc. of WWW (Companion Volume). pp. 219–220 (2014)
3. Arenas, M., Cuenca Grau, B., Kharlamov, E., Marciuška, Š., Zheleznyakov, D.: Faceted search over RDF-based knowledge graphs. J. Web Semantics 37, 55–74 (2016)
4. Arenas, M., Cuenca Grau, B., Kharlamov, E., Marciuška, Š., Zheleznyakov, D.: Enabling Faceted Search over OWL 2 with SemFacet. In: Proc. of OWLED. pp. 121–132 (2014)
5. Arenas, M., Cuenca Grau, B., Kharlamov, E., Marciuška, Š., Zheleznyakov, D., Jiménez-Ruiz, E.: SemFacet: Semantic Faceted Search over Yago. In: Proc. of WWW (Companion Volume). pp. 123–126 (2014)
6. Cuenca Grau, B., Kharlamov, E., Marciuška, Š., Zheleznyakov, D., Zhou, Y.: Querying Life Science Ontologies with SemFacet. In: Proc. of SWAT4LS (2014)
7. Grau, B.C., Kharlamov, E., Marciuska, S., Zheleznyakov, D., Arenas, M.: Semfacet: Faceted search over ontology enhanced knowledge graphs. In: ISWC Posters & Demos (2016)
8. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In: Proc. of AAAI (2014)
9. Tunkelang, D.: Faceted Search. Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers (2009)
10. Wagner, A.J.: Faceted semantic search, technical report. Tech. rep.