SemFacet: Making Hard Faceted Search Easier

Evgeny Kharlamov
University of Oxford
evgeny.kharlamov@cs.ox.ac.uk

Luca Giacomelli
Sapienza University of Rome
giacomelli.1543147@studenti.uniroma1.it

Evgeny Sherkhonov
University of Oxford
evgeny.sherkhonov@cs.ox.ac.uk

Bernardo Cuenca Grau
University of Oxford
bernardo.cuenca.grau@cs.ox.ac.uk

Egor V. Kostylev
University of Oxford
egor.kostylev@cs.ox.ac.uk

Ian Horrocks
University of Oxford
ian.horrocks@cs.ox.ac.uk

ABSTRACT

Faceted search is a prominent search paradigm that became the standard in many Web applications and has also been recently proposed as a suitable paradigm for exploring and querying RDF graphs. One of the main challenges that hampers usability of faceted search systems especially in the RDF context is information overload, that is, when the size of faceted interfaces becomes comparable to the size of the data over which the search is performed. In this demo we present (an extension of) our faceted search system SemFacet and focus on features that address the information overload: ranking, aggregation, and reachability. The demo attendees will be able to try our system on an RDF graph that models online shopping over a catalogs with up to millions of products.

CCS CONCEPTS

• **Information systems → Enterprise information systems**;

KEYWORDS

Faceted Search, Aggregation, Recursion, Ranking

1 INTRODUCTION

Faceted search is a prominent search paradigm that became the standard in many Web applications including online shopping and real-estate portals[1], where users can progressively narrow down the search results by applying filters, called *facets* [22] which are organised in faceted interfaces. Faceted search has also been proposed in the Semantic Web context as a suitable paradigm for exploring and querying RDF graphs and a number of RDF-based faceted search systems have been developed [4, 6, 11, 13–18, 20].

One of the main challenges that hampers usability of faceted search systems is *information overload* [22]: 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. Consider in Figure 1, left, a screenshot from Amazon.co.uk with a fragment of the faceted interface that one gets using the keyword 'Samsung'. The complete

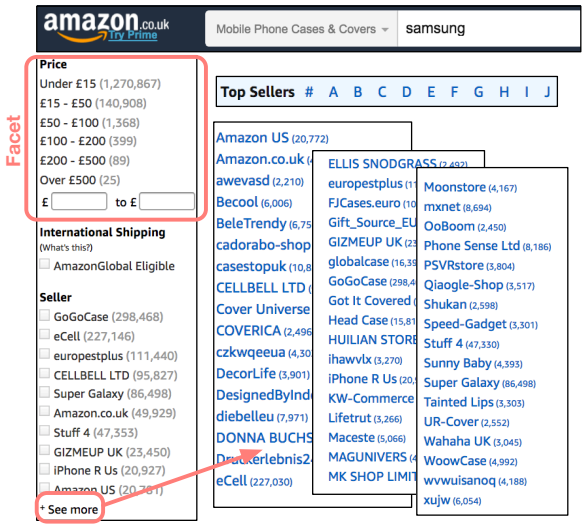

Figure 1: Faceted interface of Amazon.

faceted interface contains dozens of facets, including Price and Seller, some of which have up to thousands of values. E.g., the facet Seller has pages of values that Amazon organises alphabetically as shown in Figure 1, right. There are many ways in which the information overload can be addressed for classical faceted interfaces including ranking of facets and their values and various compact data representations such as value ranges, groups, sliders.

The information overload 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 2, 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.

[1]E.g., amazon.com, booking.com, rightmove.co.uk and immobilescout24.de

CIKM'17, November 6–10, 2017, Singapore.
 ISBN 978-1-4503-4918-5/17/11…$15.00
DOI: https://doi.org/10.1145/3132847.3133192

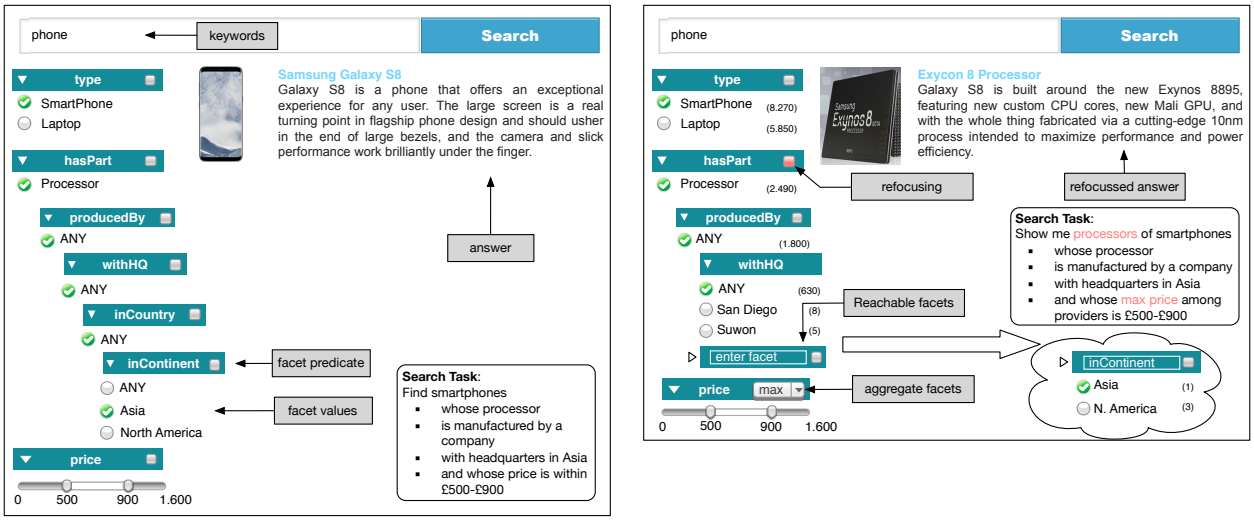

Figure 2: Left: SemFacet **with deep nesting over RDF data; Right:** SemFacet **enhanced with ranking, aggregation, and reachability**

In order to see why it might be hard to find a required nested facet, consider in Figure 3 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 3 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 2, left).

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;
- 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 2, 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 demonstration 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.

Delta from Previous Demos. Earlier versions of SemFacet were presented at [3, 5, 7–9, 12]. The current demo focuses on the novel features (ranking, aggregation, reachability) and their benefits.

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 4 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 time. The user is able to do *refocusing*. This feature of SemFacet can be observed in Figure 2, 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 [21][2] 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*

[2]RDFox is an in-memory RDF triple store that supports shared memory parallel Datalog reasoning. It is written in C++ and comes with a Java wrapper allowing for a seamless integration with Java-based applications.

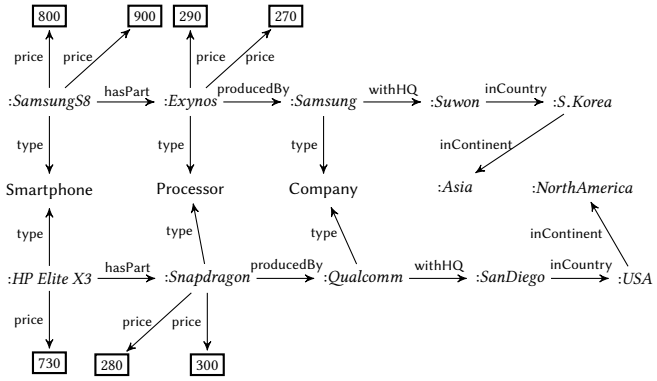

Figure 3: Example RDF graph about products

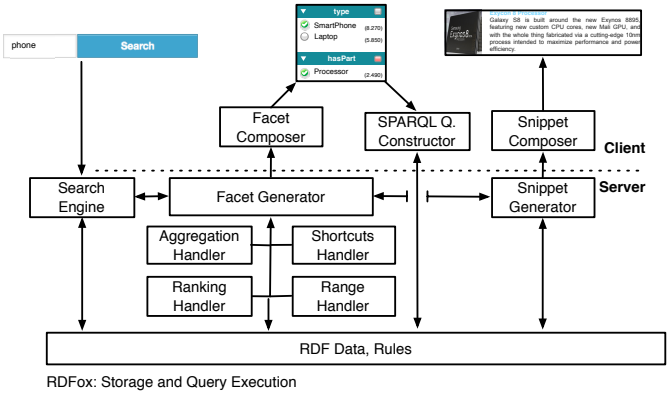

Figure 4: Architecture of SemFacet

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 2, 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 the following scoring function f and SemFacet displays the facets from S (or some of them) according to this order. Let $O = [F_1, \ldots, F_n]$ be a possible ordering of S. Our function f computes the score of O by combining the following three characteristics of O:

(1) selectivity of facets in O,
(2) diversity of facets in O, and
(3) nesting depth of facets in O.

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.

Let F denote a facet, $\mathcal{A}$ the set of current answers, $r(F) \subseteq \mathcal{A}$ the set of answers obtained by selecting any value in F, and $d > 0$ a predefined threshold parameter for nesting depth. To account for (1), we compute the *selectivity score* of F given $\mathcal{A}$ as follows:

$$\mathsf{sel}(F, \mathcal{A}) = 1 - \log_{|\mathcal{A}|} |r(F)|.$$

In particular, the less search results is obtained by applying F, the higher the selectivity score of F is. To account for (2), we consider a variation of the Set-Cover ranking [10] to compute the *overlap score* of $F_i \in O$ w.r.t. O as follows:

$$\mathsf{overlap}(F_i, O) = \frac{1}{n \times |r(F_i)|} \sum_{j=1}^{n} | \, r(F_i) \setminus \bigcup_{m=1, m \neq i}^{j} r(F_m)|$$

In particular, a high overlap score of facet F_i means that selecting values in it leads to results that are not present in the first highly ranked facets. Thus, highly positioned facets (according to the overlap score) provide more diverse results. To account for (3), we compute the *depth score* $\mathsf{depth}(F, d)$ of F given d as the minimum between d and the maximal length of nesting starting with F. Thus, the depth score prioritises those facets that are more interesting from the graph navigation point of view. Finally, we obtain f:

$$f(O, k, \mathcal{A}, d) = \sum_{i=1}^{k} \mathsf{sel}(F_i, \mathcal{A}) \times \mathsf{overlap}(F_i, O) \times \frac{\mathsf{depth}(F_i, d)}{i}.$$

Note that SemFacet does not compute f for each possible order of S since going through $n!$ orderings is impractical, instead we developed an approximation algorithm that computes a nearly optimal ordering.

Ranking facet values. Besides ranking of facets, it is important to rank facet *values* as well. We adopt the count-based ranking (see also [19, 23]): 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 straightforward 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.

RDFox performs materialisation via the use of aggregate Datalog rules. We give an example how the max aggregate facet is created for the facet price from Figure 2, right. Suppose that input RDF graph contains facts about the property price. Then at data loading time, RDFox fires the following aggregate Datalog rule

$$\text{price\_max}(?X, ?N) \ :- \ \text{price}(?X, ?Z),$$
$$\text{AGGREGATE}(\text{price}(?X, ?Y) \text{ ON } ?X \text{ BIND MAX}(?Y) \text{ as } ?N),$$

which means it materialises aggregate property price_max, i.e., it adds to the store facts of the form price_max(?X, ?N) where for each node ?X in the graph we store the numeric literal ?N that is the maximum among all numeric literals price-related to ?X. Then the materialised aggregate property is treated as a regular property. In particular, since such aggregate properties are numeric, they are presented as value ranges as discussed earlier.

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 2 (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.

3 DEMONSTRATION SCENARIOS

We now describe the demo dataset and scenarios.

Dataset. The benefits of our ranking, aggregation, and reachability components can be best witnessed on RDF datasets that are highly interconnected and with many data values. We prepared such RDF dataset that describes a product catalog of an online shop and contains geographical locations, designers, companies that produce and companies that manufacture products, as well as the actual products of 1.000 types described with 200 data properties, and interconnected with 100 object properties. In the data the average path (without repetitions) from each product is of length 10. Such paths intuitively represent compatibility between products, their relations to manufacturers, and the locations where they can be purchased or collected. In data preparation we used the data generator of WatDiv [2].

Scenario 1: Shopping. We have prepared 10 search scenarios that reflect typical online shopping. They are analogous to the search tasks in Figure 2. In these taks the attendees will have to find the right keywords to initialise the faceted search, the right facets from which to explore reachable facets, and also the right refocussing of answers.

Scenario 2: Market Analyses. For this scenario we prepared 10 tasks that mimic the search which is typically done by marketing companies that explore and aggregate data at the same time. Consider examples of three such tasks:

(T1) find smartphones with the minimal price in the UK at most £300 and with the maximal delivery time of 5 days;
(T2) find companies producing at least ten different models of smartphones with high resolution cameras;
(T3) find the most popular batteris of Korean smartphones that were manufactured from 2008 to 2015.

In this and in previous scenarios the attendees will see the impact of our ranking, aggregation, and shortcutting by using SemFacet with and without these features.

Acknowledgement. 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.

REFERENCES

[1] SemFacet Project Page. http://www.cs.ox.ac.uk/isg/tools/SemFacet/, 2017.
[2] Günes Aluç, Olaf Hartig, M. Tamer Özsu, and Khuzaima Daudjee. Diversified stress testing of RDF data management systems. In *ISWC*, pages 197–212, 2014.
[3] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, and Dmitriy Zheleznyakov. Towards Semantic Faceted Search. In *Proc. of WWW (Companion Volume)*, pages 219–220, 2014.
[4] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška, and Dmitriy Zheleznyakov. Faceted search over RDF-based knowledge graphs. *J. Web Semantics*, 37:55–74, 2016.
[5] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška, and Dmitriy Zheleznyakov. Enabling Faceted Search over OWL 2 with SemFacet. In *Proc. of OWLED*, pages 121–132, 2014.
[6] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška, and Dmitriy Zheleznyakov. Faceted Search over Ontology-Enhanced RDF Data. In *CIKM*, pages 939–948, 2014.
[7] Marcelo Arenas, Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška, Dmitriy Zheleznyakov, and Ernesto Jiménez-Ruiz. SemFacet: Semantic Faceted Search over Yago. In *Proc. of WWW (Companion Volume)*, pages 123–126, 2014.
[8] Bernardo Cuenca Grau, Evgeny Kharlamov, Šarūnas Marciuška, Dmitriy Zheleznyakov, and Yujiao Zhou. Querying Life Science Ontologies with SemFacet. In *Proc. of SWAT4LS*, 2014.
[9] Bernardo Cuenca Grau, Evgeny Kharlamov, Dmitriy Zheleznyakov, Marcelo Arenas, and Šarūnas Marciuška. On Faceted Search over Knowledge Bases. In *Proc. of DL*, pages 153–156, 2014.
[10] Wisam Dakka, Panagiotis G. Ipeirotis, and Kenneth R. Wood. Automatic construction of multifaceted browsing interfaces. In *CIKM*, pages 768–775, 2005.
[11] Pavlos Fafalios and Yannis Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In *Proc. of SIGIR*, pages 1089–1090, 2013.
[12] Bernardo Cuenca Grau, Evgeny Kharlamov, Sarunas Marciuska, Dmitriy Zheleznyakov, and Marcelo Arenas. Semfacet: Faceted search over ontology enhanced knowledge graphs. In *ISWC Posters & Demos*, 2016.
[13] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, and Ulrich Scheel. Faceted Wikipedia Search. In *Proc. of BIS*, pages 1–11, 2010.
[14] Philipp Heim, Jürgen Ziegler, and Steffen Lohmann. gFacet: A Browser for the Web of Data. In *Proc. of IMC-SSW*, pages 49–58, 2008.
[15] Michiel Hildebrand, Jacco van Ossenbruggen, and Lynda Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In *ISWC*, 2006.
[16] David Huynh, Stefano Mazzocchi, and David R. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. *J. Web Sem.*, 5(1):16–27, 2007.
[17] E. Hyvönen, E. Saarela, and K. Viljanen. Ontogator: Combining View- and Ontology-Based Search with Semantic Browsing. In *Proc. of XML Finland*, 2003.
[18] G. Kobilarov and I. Dickinson. Humboldt: Exploring Linked Data. In *LDOW*, 08.
[19] Jonathan Koren, Yi Zhang, and Xue Liu. Personalized interactive faceted search. In *WWW*, pages 477–486, 2008.
[20] m.c. schraefel, Daniel Alexander Smith, Alisdair Owens, Alistair Russell, Craig Harris, and Max L. Wilson. The Evolving mSpace Platform: Leveraging the Semantic Web on the Trail of the Memex. In *Proc. of Hypertext*, 2005.
[21] Boris Motik, Yavor Nenov, Robert Piro, Ian Horrocks, and Dan Olteanu. Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. In *Proc. of AAAI*, pages 129–137, 2014.
[22] Daniel Tunkelang. *Faceted Search.* Synthesis Lectures on Information Concepts, Retrieval, and Services. Morgan & Claypool Publishers, 2009.
[23] Andreas Josef Wagner. Faceted semantic search. Technical report.