IcicleQuery: A Web Search Interface for Fluid Semantic Query Construction

Annett Mitschick[1], Franz Nieschalk[2], Martin Voigt[2], and Raimund Dachselt[1]

[1] Technische Universität Dresden, Germany
annett.mitschick,raimund.dachselt@tu-dresden.de
[2] Ontos GmbH, Leipzig, Germany
franz.nieschalk,martin.voigt@ontos.com

Abstract. In their need to find specific entities, lay users often rely on traditional full text searches that render, in comparison to the capabilities of the Semantic Web, an inferior approach. However, the access to semantic data is complicated by technological barriers that non-experts have to overcome. Despite the attempt to make the Semantic Web more accessible, only few user interfaces have been created so far that combine the ease of keyword search with the structuredness of relational queries. None of them focuses explicitly on lay users. This paper presents an approach for closing this gap by fluidly combining the ability of full text search and semantic search in a compact and comprehensible query interface. Therefore, we designed and implemented an innovative web-based interface based on the concept of Icicle Plots retrieving search results from a hybrid data server. A first qualitative study revealed that the interface provides an expressive but still approachable way of querying for specific entities and their accompanied information.

Keywords: user interface; semantic search; keyword search; Icicle Plots

1 Introduction

Nowadays, information search is a main task of people's digital life. Studies revealed that specific entities and their accompanied information take a key role in users' searches. Kumar et al. [8] found that 52.9% of all queries are targeting them, while Guo et al. [6] found that 71.0% contain them. Also Spirin et al. [13] confirmed that named entity and structured queries are essential to address a diverse set of information needs of users. Thus, users have the need to search with and for those entities but may find it difficult to achieve this goal by common interfaces. These usually build on the concept of full text search which heavily rely on documents and their metadata [16]. In contrast, faceted browsing [15] is a more entity-centric approach allowing for filtering things based on their attributes. Unfortunately, filtering entities based on their relations is hard to achieve via current user interface concepts.

However, the advent of Semantic Web technologies and Linked Data and their growing application in real world scenarios [3] but also the advance of graph databases underline that linkage of data and entities is a first class objective during modeling, creation and usage of data. In the constantly growing mass of web documents, which especially increases by social media and user-generated content, technologies like *Named*

Entity Recognition enable the extraction of semantic information from unstructured text which reveals more sophisticated opportunities for search and analytics.

With regard to the commonly used web search user interfaces and the growing amount of interlinked data (which is also challenging in enterprise and industry scenarios [9]), our goal was to develop a smart search prototype that (1) is entity-centric, (2) makes use of semantic information of an entity, and (3) allows to follow the links between entities in order to specify complex queries. Despite the expectable complexity, we wanted to achieve that (4) especially lay users, i. e. non-experts in the field of ontology-based data modeling, could create semantic queries with little effort.

The key contributions of our work are

- a substantial reduction of the complexity of semantic query construction in order to make it is easy to use for lay users,
- a new, extendable, space-saving user interface widget for semantic query construction based on the idea of *Icicle Plots* [7],
- fluid integration between full-featured text search and semantic querying (both on the back-end and front-end side).

This paper is structured as follows: After reviewing related work in Section 2, we describe our design decisions for the implemented search prototype and the proposed integration between *RDF* data and a common full text search index in Section 3. Since we conducted a qualitative user study to evaluate the prototype, we outline its execution and the main findings in Section 4. Finally, we conclude our work and give an outlook on the next steps and future research directions.

2 Related Work

With the aforementioned requirements in mind, we researched existing work with a focus on exploratory semantic search interfaces and visual semantic query building.

A lot of research has been done in the field of exploratory semantic search interfaces. As an example, Mirizzi et al. [10] introduced *Semantic Wonder Cloud* which enables the exploratory traversal of entities from *DBpedia* through their relations. Based on a selected starting node, users navigate the graph structure by a simple point-and-click interaction on the next node. While this allows for an intuitive navigation, also for lay users, this approach does not support complex (chained) queries that also reuse other information of the entities. Another approach was proposed by Popov et al. [11]: *Visor* allows the user to select multiple ontology classes of interest and to display them and their direct relations in a node-link-diagram. Furthermore, it suggests indirect relations over intermediary classes enabling the user to gather a conceptual overview of the underlying data structure. Based on the selection, the data set is queried and the resulting instances are presented in a spreadsheet-based user interface.

NITELIGHT [12] is an exemplary visual query builder that visualizes *SPARQL* graph patterns as node-link diagrams. The tool reuses pre-loaded ontologies in order to provide a general overview of all classes and properties. These can be dragged and dropped onto the query canvas to configure the queries. Although this concept allows

the construction of complex queries, the textual visualization and the focus on *SPARQL*-like queries are targeting expert users.

Another example is *Graph Pattern Builder* [1] which provides a form-based query building approach. The user can only choose from suggested entries according to her precedent input to make sure that the constructed query is syntactically and semantically correct. However, fundamental knowledge about *SPARQL*-queries is required, e. g. regarding the definition of variables.

The hybrid index by Bast and Buchhold [2] allows for another kind of iteratively constructing queries. Based on the user's input, context-sensitive suggestions for words, classes, relations, and instances are retrieved and added to a tree structure. This reduction of complexity from graphs to trees as well as the concept of a query builder that facilitates full-text and semantic queries shaped our own vision for a search user interface. However, their visualization represents the query components in a technical manner like classes, etc., hence, user need certain knowledge about underlying ontologies.

A general finding is that existing tools try to allow for comfortable interaction with interlinked data but are more or less targeting expert users since their visualization makes heavy use of (complex) graphs or textual queries. Vega-Gorgojo et al. [14] reported on the results of a user study comparing the prototypes of a form-based and a graph-based interface to assess the effectiveness of visual query interfaces. They found evidence that experts prefer graph-based interfaces while mainstream or lay users "obtained better performance and confidence with the form-based interface" which is "more easily learned and relieves problems with disorientation". Query construction benefits from term suggestion for classes, relations and literals during typing. This helps to overcome the burden of less domain knowledge and is a widely used technique for keyword-based search today.

Based on these reflections, the next section covers our conceptual approach and its prototypical implementation.

3 Design and Prototype

The two main thoughts regarding our concept included firstly how to combine the abilities of full-text and semantic search in a single hybrid data server (3.1) and secondly how to represent both keywords and semantics in an intelligible user interface (3.2).

3.1 The Search Back-end

Alongside with the front-end concept, a basic decision regarding the search back-end was required. Since we tried to accomplish a semantic search on Linked Data based on the *RDF* standard, the two obvious candidates were to use either a triple store or a dedicated full-text search engine. The advantages of a triple store are to work directly on the *RDF* data and use *SPARQL* as standardized, powerful query language. On the other side, a search engine like *Apache Solr*[3] or *ElasticSearch*[4] scores regarding features

[3] `https://lucene.apache.org/solr/`

[4] `https://www.elastic.co/products/elasticsearch`

like advanced, fast full-text search, faceting, aggregations or query suggestion. As the decision was less about storing data but more about searching and analyzing, we chose to use a search engine (Solr in particular) as back-end and extend it with semantic capabilities.

Data Import In order to transform the *RDF* data to a *Solr* schema, we defined a domain independent, flexible schema that allows for mapping *RDF* triples (subject, predicate, object) to different field types. Here, the URI of the subject - the entity to search for - is used as basic document ID. For each predicate, a field based on the wildcard syntax is automatically created where the field value is the object. In order to keep any available language tags, e. g., for labels, extra fields are generated. Based on this rough concept, we could adequately index ontologies and instance data likewise. As long as the incoming data follows the *OWL* specification and uses *XSD* datatypes, any entity can be consumed and stored in the *Solr* index. Sufficient model information must be provided for the front-end query builder, especially `rdf:type` and `rdfs:label` of classes and properties, and queryable instances should have a `rdfs:label` and a `rdfs:comment` to reasonably execute full-text search on them.

Query Formulation Another important concept was to create relational, nestable *Solr* queries representing adequate SPARQL queries. Basically, *Apache Solr 6* provides two mechanisms for this type of queries: `!join` and *graph traversal* operations. We decided on the first since *graph traversal* is used for collecting documents along a path of properties rather than being a pure inner query like `!join`. These operations could be nested based on the possibility of the *LocalParams* syntax yielding to deep structures of relational statements. Each subject of those statements is held in a unique parameter (e. g., `l0p0n0`) and maps its properties to the field names that corresponds to the index schema. An example *Solr* query is shown in Listing 1.1, comprising two nested `!join` operations (equivalent to the scenario in Figure 1). As the example query shows, keyword searches are executed on labels and comments, where matches on labels are boosted twice as much as comment hits. It is also important to note that we are devaluing keyword matches the deeper they are located in the structure and thus, "further away" from the search target. Depending on the node level l the base boost b is equated by $b = \frac{1}{1+l \cdot 0.1}$, but it should be part of ongoing work to determine a more appropriate formula for this purpose.

Performance Knowing that nested Solr `!join` queries may not perform as well as regular Solr queries [4], we tested the performance of our back-end solution by measuring the query time for queries of rising complexity, starting from a query that contains no `!join` operation up to a chained query of eight nested `!join` operations. The tests were performed on two datasets, one containing 38K and the other 133K entities (3.5 times more). We only measured the first query execution and hence without making use of Solr's caching functionality. The results (cf. Figure 2) show that nested `!join` queries of depth 1, 2 and 4 do not increase query time significantly, but a very significant increase was measured for queries with 8 nested `!join` operations. However, we considered queries of this depth to be not representative being rather too complex for

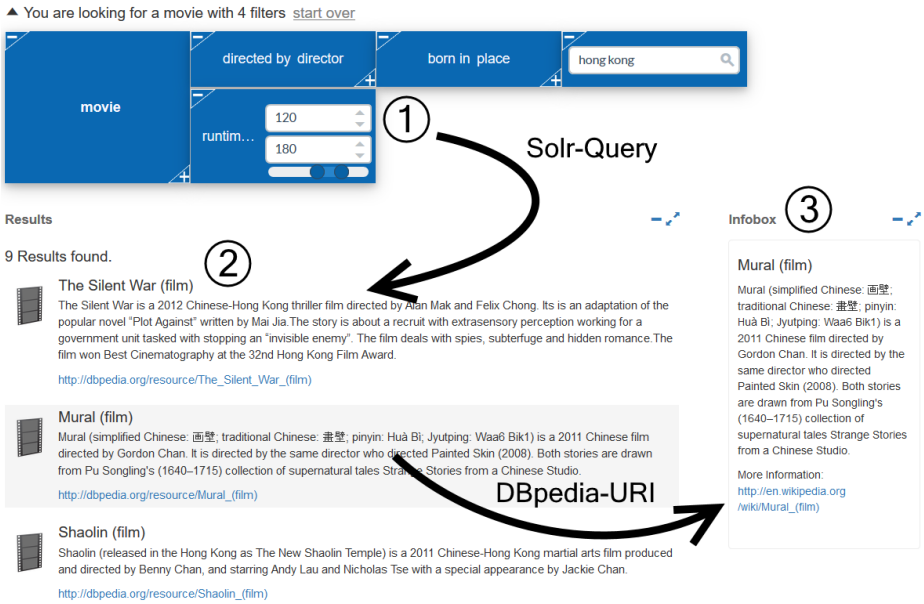

Fig. 1. Overview of the *IcicleQuery* web search user interface: (1) query builder, (2) result list, (3) additional information about a selected resource.

humans (esp. lay users) and leading to a very high specificity of the query issuing too few or even no search results.

3.2 IcicleQuery: The Search Front-end

From the reflections of existing work, we draw the conclusion that as lay users are in general not familiar with semantic graphs, its classes and relations, the user interface should try to abstract from them as much is possible. According to Fu et al. [5], tree-like representations are more intuitive for lay users than graphs. Thus, a hierarchical, tree-like visualization seems to be recommendable. The fundamental idea of our query builder is to reduce the query and visualization complexity from directed graphs to trees. Although this decision implies that not all types of graph patterns could be used for a query, lay users might not need them. With this foundation, we gathered different types of tree visualizations and, finally, considered the approach of *Icicle Plots* [7] as most suitable. This kind of representation is easy to understand and to layout as well as usable for mouse and touch interactions. Furthermore, using a horizontal layout it is an especially compact representation allowing for a space-saving design of the query component, which is most suitable for devices with small displays, and leaves enough room for the result presentation.

A vital principle is a fluid word-completion functionality for all text entry fields in the query builder, providing suggestions for either classes, relations or literals by just typing desired terms. This helps to overcome the burden of less domain knowledge

Listing 1.1. An example of a Solr query

```
?q=*:* AND ${l0p0n0}
&l0p0n0=rdf_type:"dbo:Film" AND
  {!join from=id to=dbo_director_rdf_uri_res
        score=total v=$l1p0n0}  AND
  dbo_runtime_xsd_double_lit:[120 TO 180]
&l1p0n0=rdf_type:"dbo:MovieDirector" AND
  {!join from=id to=dbo_birthPlace_rdf_uri_res
        score=total v=$l2p0n0}
&l2p0n0=rdf_type:"dbo:Place" AND
  (en_rdfs_label:(hong kong)^1.66 OR
        en_rdfs_comment:(hong kong)^0.83)
```

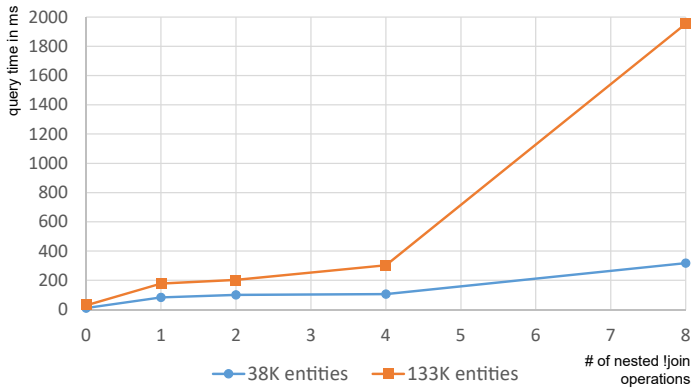

Fig. 2. Query time (in ms, y-axis) depending on the number of entities and the number of nested `!join` operations (x-axis).

and is consistent to user expectations, as it is a common feature of today's web search engines.

Figure 1 provides an overview of the complete web-based search user interface. While the user creates a semantic query in the *IcicleQuery* builder ①, suitable results are shown immediately below ②. Clicking on a result provides additional information from the knowledge graph ③. Our work focused especially on the query builder ①, which is described in the following.

A query construct in *IcicleQuery* comprises of at least one but usually multiple nodes that are visualized as horizontal aligned squares (Figure 1-①). Each node of the tree, including the root node, is initiated based on a search in a text input field. During typing, suitable terms are suggested which are derived from the underlying ontology and instance data. As a class is selected (like *actor* in Figure 3), its human readable label is shown as visual representation. According to the general application language and the available labels in the data, it is easy to change the name of the node dynamically.

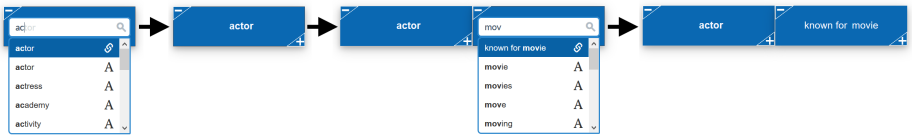

Fig. 3. Iterative construction of the relational statement *"Find actors that are known for movies"*.

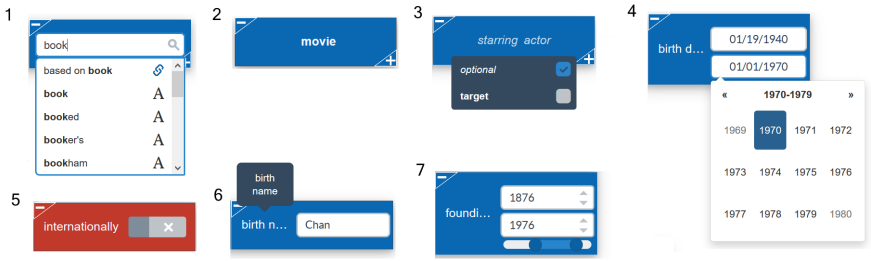

Fig. 4. Several node types provide specific input widgets: (1) text input with word completion, (2) node representing a class filter, with the '+'-button to add a subsequent filter, (3) options to declare a filter to be optional or to be the new target (inverting the subtree), (4) date selection, (5) removing a boolean filter with the '-'-button, (6) tooltip for abbreviated labels, (7) interval input.

Furthermore, the selection of a class is required to add additional filters to the query. The available filters are directly gathered based on the class' attributes and relations. The small example in Figure 3 shows two nodes creating a semantic query for *"Find entities of the type actor that are linked to entities with the type of movies"*. The according semantic query using *SPARQL* syntax would result in a construct of three graph patterns – the underlying query complexity is thus hidden from the user.

By setting no target class, it is also possible to utilize the first node as a conventional full text search, giving inexperienced users the option to bypass advanced query builder features. Since new full text search nodes can be attached to the tree, keyword searches may also be applied to nodes that are positioned deeper in the structure.

The more advanced example in Figure 1 shows further possibilities to extent the query. First, there could be more than one filter attached to an existing node that represents a class or a relation. Here, the sought movie should be "directed by a director'" and should have a defined runtime. Second, the filter for the runtime but also for the birthplace show the possibility to restrict entities according to parameter values. Since the data types are known, the filter widgets are adapted to them. Up to now, we implemented widgets for text, boolean values, numeric ranges and date selections (see Figure 4). Moreover, based on the range of the literal values, the user can only create valid queries that do not lead to empty result sets. This concept was adopted from Faceted Browsing.

A very sophisticated feature is the ability to switch the reading direction of the query by defining a new search target from the existing query with one click. After

selecting an object of the query statement as new target the object becomes the subject of a rearranged query.

The current version of *IcicleQuery* is implemented as a component of Ontos EIGER[5], a web-based Linked Data suite for semantic data integration, including a flexible dashboard in order to allow for data visualization and search for none-data scientists. The *IcicleQuery* prototype was realized using *AngularJS* and *Bootstrap*. A working demo can be tried out here: `http://demo.ontos.com/dashboard`.

4 User Study

In order to understand the applicability and suitability of our concept and receive preliminary feedback for further development, we conducted a formative, qualitative user study. We especially focused on a comparison between experts and lay users to evaluate the potential to reduce the complexity of semantic search. The two main questions we were interested in were: What are the differences between experts and lay users when using the query builder? Which kind of search tasks is it most suitable for? In the following, we describe the design and setup of the study and our results in more detail.

4.1 Procedure

Ten participants volunteered (three female; age between 21 and 40), each session lasted around 30-40 minutes. Besides our target group of six lay users, we also ran tests with four users that had deeper expertise in the field of Semantic Web technologies, acting as a reference group. All of the participants stated, that they regularly search for specific entities in common web search engines. The test data for the study was extracted from *DBpedia* and contained a small set of about 40k distinct entities related to the *DBpedia* category `dbc:2010s_films`. The corresponding model contained 39 distinct classes and properties. We chose a small set to better control the expected result sets and evaluate the search performance of the subjects.

Each session consisted of three parts: (1) free exploration, (2) search tasks, and (3) a questionnaire. The study started with an explanation of the search domain without showing any data model. In order to evaluate the intuitiveness of the prototype, the subjects were asked to try out the search widget on their own. We let the users freely explore the interface and observed their behavior. We were especially interested if they would identify query construction features, e. g., how to add new query nodes to create filters, on their own. After that, we gave a brief introduction to the advanced search features of the interface, which were needed to perform the following tasks. Every user had to solve four different search challenges of increasing complexity in order to find specific movies, actors or production companies in the data set:

1. *Find a movie titled "The Revenant".*
2. *Find actors who received an "Oscar" (Academy Award).*
3. *Find a film studio that produced a movie with a title containing "Love".*
4. *Find an actor known from a movie directed by someone called "Chan".*

[5] `http://ontos.com/`

The users had to build a suitable query for the given demand with the help of the query builder, being told to formulate queries only until they were content with the search results. Based on the *thinking-aloud* methodology, we gathered helpful feedback about the cognitive model of the subjects. These observations were completed by a follow-up questionnaire, which inquired qualitative feedback about user satisfaction, the perception of easiness and precision, and preferences for either Google-like search or our *IcicleQuery* interface using Likert-scales. Further on, we asked which tool they would prefer for which tasks.

4.2 Results

(1) Free exploration: The experts (reference group) did not need guidance to identify and apply the advanced query-building features and easily adopted them. The lay users did not expect those features, but they attempted the full-text search in order to access the data. Only 2 of 6 lay users identified the functionality to add new nodes to the *Icicle* in order to extend the query and restrict the result set on their own. Although this is heavily a usability issue, since the +-button was too small and only shown on mouse-over, we found that especially lay users did not search for the possibility to add a new relation, but tried to type everything into the first box.

(2) Search tasks: After a short explanation of the basic interaction concepts, all subjects, both experts and lay users, were able to solve the given tasks immediately and correctly with neither considerable interruptions nor further questions regarding the query construction.

(3) User feedback from the questionnaires: The results from the questionnaires were particularly encouraging since 6/10 stated that *IcicleQuery* seemed to be faster, and 10/10 found that the results were more precise compared to traditional web search. For tasks only requiring keyword search 8/10 would have preferred Google, but with growing task complexity *IcicleQuery* was judged superior (cf. Figure 5). Regarding the question "How well did you solve the given tasks from your point of view?" (very good/good/poor/very poor) 2/10 of the participants chose "good", 8/10 "very good".

From the given feedback we conclude that in general the participants (i) found the visualization appealing and understandable, (ii) were content with the complexity of specifiable queries, (iii) thought they satisfyingly fulfilled the search tasks, and (iv) envisioned using the interface for general as well as specific search tasks. While all lay users were satisfied with the capabilities of the interface, the expert users missed additional features like declaring statements as negative (`NOT`) or temporarily deactivating branches of the tree. Together with the usability issues, this provides valuable feedback for further development.

4.3 Discussion

We admit that the number of subjects is rather small and in general not sufficient to give adequate proof for any claims. However, we sought preliminary, qualitative feedback to further enhance the prototype to address usability issues. It is not our objective to

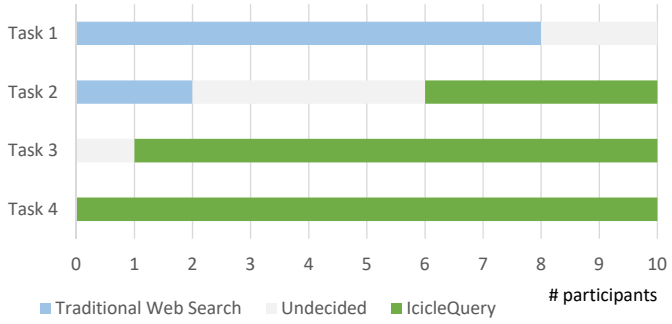

Fig. 5. Preferences of the participants for a search solution depending on the search task (growing complexity from 1 to 4).

judge between search principles or give general preference to semantic querying. Using a high-level query language (SQL, SPARQL) is the most sophisticated, precise way to make use of the semantic richness of complex data sets, but lay users are mostly unable to cope with their complexity. Thus, our goal was to create an easy to use user interface yet providing the power of a rich query language. The promising results of the study and the positive feedback of the participants suggest that we took a large step in this direction. On the other hand, we clearly understand that there are still a couple of usability issues to be solved in order to improve clarity and learnability of the interface to do without specific guidance or assistance.

5 Conclusion and Future Work

In this paper we introduced an innovative, space-saving search interface based on an adaptation of the concept of *Icicle Plots*. It fluidly combines the features of a sophisticated search engine, e. g., advanced text search, faceting or aggregation, with the capabilities of Linked Data sources. Our approach allows for an advanced semantic search especially for lay users who are not familiar with the underlying concepts. We successfully evaluated our concept based on a prototypical implementation using *Apache Solr* as flexible back-end for indexing *RDF* data. A qualitative user study revealed some usability issues and underlined that semantic search is (still) a new topic for lay users. However, it also shows the feasibility of our concept since all participants solved all tasks efficiently and correctly. Furthermore, with increasing complexity of the search task the participants gave preference to *IcicleQuery* as search interface of choice.

For future work, we plan to improve the intuitiveness and extend capabilities of the user interface while maintaining its fluid and lightweight habit. As for the mentioned usability issues we aim at incorporating visual hints for query construction and let users find filter suggestions by synonymous terms through the integration of dictionaries or thesauri. Another improvement could be to hierarchically cluster the suggested lists of properties or classes in case of too many items. For a wider range of functions, we also envision the re-use of specific entities in queries, the ability to shift search focus to

properties or literals or to an aggregation of multiple entities, and enabling the user to configure sorting criteria and priorities through weighting parameters.

Acknowledgements

This work is partly funded by the German Federal Ministry of Education and Research under promotional reference number 03WKCG11B.

References

1. Auer, S., Lehmann, J.: What Have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content, pp. 503–517. Springer Berlin Heidelberg, Berlin, Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-72667-8_36
2. Bast, H., Buchhold, B.: An index for efficient semantic full-text search. In: Proceedings of the 22nd ACM International Conference on Information & Knowledge Management. CIKM '13, ACM, New York, NY, USA (2013), http://doi.acm.org/10.1145/2505515.2505689
3. Bernstein, A., Hendler, J., Noy, N.: A new look at the semantic web. Commun. ACM 59(9), 35–37 (Aug 2016), http://doi.acm.org/10.1145/2890489
4. Erickson, E.: Solr and joins: Experimenting with join performance. https://lucidworks.com/blog/2012/06/20/solr-and-joins/ (06 2012), https://lucidworks.com/blog/2012/06/20/solr-and-joins/
5. Fu, B., Noy, N.F., Storey, M.A.: Indented Tree or Graph? A Usability Study of Ontology Visualization Techniques in the Context of Class Mapping Evaluation, pp. 117–134. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-41335-3_8
6. Guo, J., Xu, G., Cheng, X., Li, H.: Named entity recognition in query. In: Proceedings of the 32nd Int. ACM SIGIR Conference on Research and Development in Information Retrieval. SIGIR '09, ACM, New York, NY, USA (2009), http://doi.acm.org/10.1145/1571941.1571989
7. Kruskal, J.B., Landwehr, J.M.: Icicle plots: Better displays for hierarchical clustering. The American Statistician 37(2), 162–168 (1983)
8. Kumar, R., Tomkins, A.: A characterization of online browsing behavior. In: Proceedings of the 19th International Conference on World Wide Web. pp. 561–570. WWW '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1772690.1772748
9. Marco F. Huber, Martin Voigt, A.C.N.N.: Big data architecture for the semantic analysis of complex events in manufacturing. In: 2nd International Workshop on Big Data, Smart Data and Semantic Technologies (BDSDST). INFORMATIK 2016 (2016)
10. Mirizzi, R., Ragone, A., Di Noia, T., Di Sciascio, E.: Semantic Wonder Cloud: Exploratory Search in DBpedia, pp. 138–149. Springer Berlin Heidelberg, Berlin, Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-16985-4_13
11. Popov, I.O., Schraefel, M.C., Hall, W., Shadbolt, N.: Connecting the dots: A multi-pivot approach to data exploration. In: Proceedings of the 10th International Conference on The Semantic Web - Volume Part I. pp. 553–568. ISWC'11, Springer-Verlag, Berlin, Heidelberg (2011), http://dl.acm.org/citation.cfm?id=2063016.2063052
12. Russell, A.: Nitelight: A graphical editor for sparql queries. In: Proceedings of the Poster and Demonstration Session at the 7th International Semantic Web Conference (ISWC2008). pp. 110–111. ISWC-PD'08, CEUR-WS.org, Aachen, Germany, Germany (2008), http://dl.acm.org/citation.cfm?id=2889529.2889584

13. Spirin, N.V., He, J., Develin, M., Karahalios, K.G., Boucher, M.: People search within an online social network: Large scale analysis of facebook graph search query logs. In: Proc. of the 23rd ACM Int. Conference on Information and Knowledge Management. CIKM '14, ACM, New York, NY, USA (2014), http://doi.acm.org/10.1145/2661829.2661967
14. Vega-Gorgojo, G., Slaughter, L., Giese, M., Heggestyl, S., Soylu, A., Waaler, A.: Visual query interfaces for semantic datasets: An evaluation study. Web Semantics: Science, Services and Agents on the World Wide Web 39, 81 – 96 (2016), http://www.sciencedirect.com/science/article/pii/S1570826816000159
15. Yee, K.P., Swearingen, K., Li, K., Hearst, M.: Faceted metadata for image search and browsing. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '03, ACM, New York, NY, USA (2003), http://doi.acm.org/10.1145/642611.642681
16. Zhou, M.: Entity-centric search: querying by entities and for entities. Ph.D. thesis, University of Illinois at Urbana-Champaign (2015)