Discovering Diversified Paths in Knowledge Bases

Christian Aebeloe*, Gabriela Montoya*, Vinay Setty*†, Katja Hose*
*Aalborg University, Denmark
{caebel, gmontoya, vinay, khose}@cs.aau.dk
†University of Stavanger, Norway
vinay.j.setty@uis.no

ABSTRACT

Vast amounts of world knowledge is now accessible through Knowledge Graphs (KGs) in RDF format and can be queried using SPARQL. Yet, finding paths between nodes in such graphs is not part of the official SPARQL 1.1 standard; only the simpler functionality of checking reachability is supported, i.e., assessing whether two nodes are connected based on certain conditions formalized as property paths but without providing information on how they are actually connected. To close this gap of functionality, we present JEDI, a system that extends a popular SPARQL engine, Jena, with the ability to compute paths connecting entities in a KG. JEDI shows the k most relevant results to the user where relevance is assessed as a trade-off between path length and diversification of the intermediate nodes in the path. Furthermore, our solution is not limited to a single property path pattern but supports queries containing multiple property path patterns. While JEDI supports arbitrary KGs, for demonstration purposes some predefined KGs, such as YAGO and DBLP, will be used.

PVLDB Reference Format:
Christian Aebeloe, Gabriela Montoya, Vinay Setty, Katja Hose. Discovering Diversified Paths in Knowledge Bases. *PVLDB*, 11 (12): 2002-2005, 2018.
DOI: https://doi.org/10.14778/3229863.3236245

1. INTRODUCTION

In the past decade, representing and querying knowledge in a structured way using Knowledge Graphs (KGs), such as YAGO [5] and DBpedia [2], has become more and more popular. Recently, several commercial KGs such as the Google KG, the Facebook Entity Graph, and Microsoft Satori have been proposed with applications to search engines and personal voice assistants. Given the proliferation of KGs and their verbose structure, there is a clear need for algorithmic techniques to facilitate interactive exploration and analysis of semantic relationships between entities in a KG.

Due to their versatility and expressiveness, the W3C standards RDF[1] and SPARQL[2] have been widely adopted in the Semantic Web community. The recent introduction of SPARQL 1.1 has enabled novel approaches and optimizations [6, 7, 9] incl. property paths[3], which made it possible to formulate SPARQL queries checking the transitive reachability of entities (vertices) in RDF graphs using regular expressions involving operators, such as '*' (zero or more occurrences–kleene star), '|' (OR operator), '∧' (NOT operator), etc.

However, these operators are only of very limited use for exploring semantic relatedness between entities as they do not return the full paths between the entities but merely check for its existence. For example, a property path query with the clause "`Princess_Margaret_of_Connaught hasChild* Margrethe_II_of_Denmark`" only checks if `Margrethe_II_of_Denmark` is transitively reachable from `Princess_Margaret_of_Connaught` via any number of `hasChild` property relationships but omits the intermediate entities in the path.

There is currently only very little related work in this area. Some systems try to address this issue by computing the complete paths connecting a given pair of entities defined in a user query [4, 11] or sets of entities [12], respectively. However, these systems are either limited to queries involving a single property (as opposed to queries supporting arbitrary paths involving a diverse range of properties occurring in a KG) or ignore the properties completely. There is usually no advanced support for combining property path expressions with other standard SPARQL operators. Moreover, to the best of our knowledge there is no system that supports retrieving full paths for queries involving multiple property path patterns.

To address these issues, we propose a system coined JEDI (Jena Extension for DIscovering diversified paths in knowledge bases). JEDI extends the property path grammar with a new operator '$\rightarrow$', which, in addition to checking the reachability, also enumerates all the property paths matching an arbitrary given property path query. In general, however, it is not sufficient to simply enumerate all paths fulfilling a property path pattern as such results are often large in number and redundant. We therefore introduce a ranking scheme using a diversification metric as a post-processing step. As we implement our solution as an extension to Jena[4], a popular framework for parsing and processing SPARQL queries, JEDI is able to combine property paths with other operators known from standard SPARQL queries.

Example Query. Suppose we are interested in finding any possible links between the Danish and Swedish royal families along with their lineage. We can express a property path query with the $\rightarrow$ operator to retrieve all the common ancestors of Margrethe II of Denmark, the current Queen of Den-

[1] https://www.w3.org/RDF/
[2] https://www.w3.org/TR/rdf-sparql-query/
[3] https://www.w3.org/TR/sparql11-property-paths/
[4] https://jena.apache.org/

Proceedings of the VLDB Endowment, Vol. 11, No. 12
Copyright 2018 VLDB Endowment 2150-8097/18/08.
DOI: https://doi.org/10.14778/3229863.3236245

mark, and Carl XVI Gustaf of Sweden, the current King of Sweden – the corresponding SPARQL query is shown in Listing 1.

```
PREFIX :<http://yago-knowledge.org/resource/>
PREFIX rdfs:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT * WHERE {
 ?a :hasChild-> :Margrethe_II_of_Denmark   .(pp1)
 ?a :hasChild-> :Carl_XVI_Gustaf_of_Sweden .(pp2)
 ?a rdfs:label ?b
}
```

Listing 1: SPARQL query Q1 to retrieve links between Danish and Swedish royal families

Figure 1 illustrates the graph formed by the top-3 paths matching the query in Listing 1, which clearly shows the need for diversification. When we rank the results purely based on path lengths (Figure 1(a)), we can see that all paths share the same subpath after the entity `Princess_Margaret_of_Connaught`. In Figure 1(b), when we rank equally based on the diversification metric and path length to select the top-3 paths, we discover new paths, e.g., from the paternal side of `Margrethe_II_of_Denmark`, via the entity `Frederik_XI_of_Denmark` even though it is a much longer path. These new paths do not involve the same subpath as when we rank purely based on path lengths, and thus are more diverse.

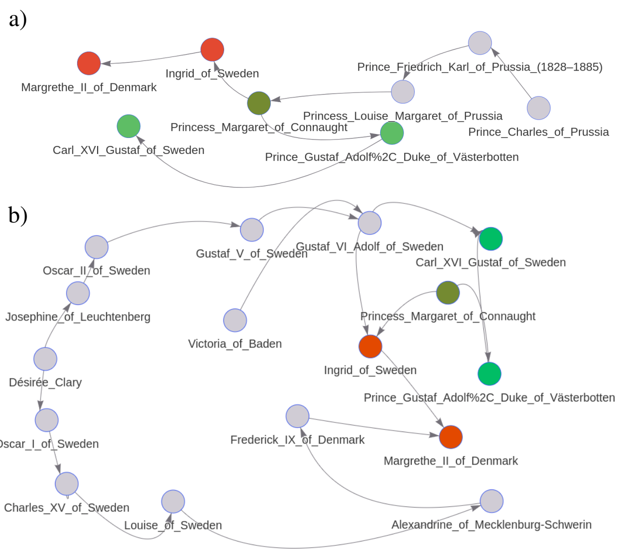

a)
Margrethe_II_of_Denmark
Ingrid_of_Sweden
Prince_Friedrich_Karl_of_Prussia_(1828–1885)
Princess_Louise_Margaret_of_Prussia
Princess_Margaret_of_Connaught
Prince_Charles_of_Prussia
Carl_XVI_Gustaf_of_Sweden
Prince_Gustaf_Adolf%2C_Duke_of_Västerbotten
b)
Oscar_II_of_Sweden
Gustaf_V_of_Sweden
Gustaf_VI_Adolf_of_Sweden
Carl_XVI_Gustaf_of_Sweden
Josephine_of_Leuchtenberg
Victoria_of_Baden
Princess_Margaret_of_Connaught
Ingrid_of_Sweden
Désirée_Clary
Prince_Gustaf_Adolf%2C_Duke_of_Västerbotten
Oscar_I_of_Sweden
Frederick_IX_of_Denmark
Margrethe_II_of_Denmark
Charles_XV_of_Sweden
Louise_of_Sweden
Alexandrine_of_Mecklenburg-Schwerin

Figure 1: Top-3 paths for the query in Listing 1: (a) Without diversification, (b) With diversification. For both only the top-1 result path is highlighted: red nodes are part of property path *pp1*, green nodes are part of property path *pp2*, mixed color nodes are part of both, and gray nodes are part of other results.

2. SYSTEM OVERVIEW

JEDI is implemented on top of Jena[4]. We have extended the grammar of property path operators accepted by Jena with a new operator, '→'. This operator outputs the actual paths connecting two nodes in addition to performing the reachability check between them. JEDI comes with a Java Web Application for graphical user interaction that allows the user to explore the query results both in a tabular format and in a visual graph-based format. JEDI allows the user to select one or more results in the table to highlight the corresponding paths in the graph. JEDI's code is available at http://qweb.cs.aau.dk/jedi/. The overall architecture of JEDI is sketched in Figure 2. JEDI's three main components are: a Graphical User Interface, a Query Executer, and a Diversifier.

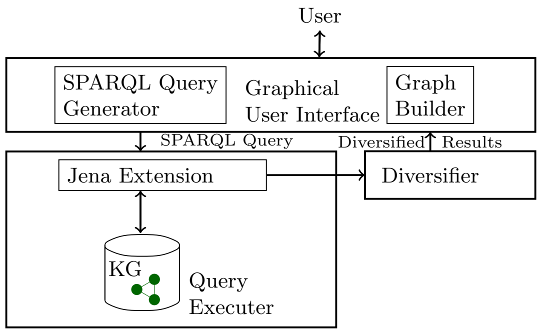


Figure 2: System architecture of JEDI.

Graphical User Interface. This component provides the user with a Web Interface to interact with JEDI in two modes: (i) a *simple query mode* for less experienced users requiring help in formulating SPARQL queries and (ii) an *advanced query mode* allowing users to directly input arbitrary SPARQL queries. In the simple query mode (see Figure 3 (b)), users are guided to provide every component of a SPARQL query using input fields and the assistance of auto-completion functionality to compute and present suggestions to the user. Property path operators can be chosen from a list; for binary operators another field is dynamically added for the remaining operand. Additional triple patterns or property path patterns can easily be added as well. The '→' operator is enabled by activating a check box (see Figure 3 (c)); if not activated the standard implementation of property paths is applied during query evaluation. For experts, the advanced query mode provides the user with a textbox where he/she can freely edit and input an arbitrary SPARQL query. The Web Interface also displays the results both as a table and as a graph, and allows users to explore the results by selecting some query result (rows in the table) to highlight the corresponding paths in the graph and choosing the colors used to highlight the paths.

In the simple query mode, this component relies on the SPARQL Query Generator to obtain the SPARQL query that is passed on to the Query Executer. In both modes, the Graph Builder is used to present the top k diverse results to the user as a graph that allows for visualizing and exploring the obtained paths.

Query Executer. The → operator computes the paths that connect entities in a Knowledge Graph (KG). To compute these paths, the nodes reachable from the source node are visited in depth-first order and their immediate predecessors are stored. Then, these predecessors are used to compute the actual paths, while using the number of paths required to avoid the computation of an exponential number of paths. As paths with loops are not shorter or more diverse, they are omitted. JEDI relies on a SPARQL compliant service to access the KG. This service does not handle the → operator, but is used to retrieve triples in the KG that possibly fulfill some conditions, e.g., with a given subject or property. Hence, JEDI can be used together with arbitrary SPARQL endpoints. However, in the context of this demonstration, JEDI relies on a Virtuoso 7.2.4.2[5] endpoint as back-end.

Diversifier. Finding all possible paths connecting entities can be very expensive and overwhelm the user with too many results.

[5]https://virtuoso.openlinksw.com/

In order to provide the user with the top-k most relevant results only, JEDI diversifies the results using a diversification metric (Section 2.1). JEDI is able to diversify results even for queries with multiple paths by using the diversification metric over the combination of all the connected paths present in each query result.

2.1 Diversification Metric

Our diversification metric aims at balancing the path length and the similarity among the result paths R of a given query Q. The balance between these two is determined by a user-defined diversification parameter $0 \leq \lambda \leq 1$. If $\lambda = 0$, we rank the paths purely according to their lengths. On the other hand, if $\lambda = 1$, the k most dissimilar paths according to Jaccard similarity irrespective of their path lengths are chosen. Given a query Q, with matching results R, our goal is to select $S \subseteq R$ according to the following objective:

$$DivP(R) = \arg\max_{S \subseteq R} \sum_{P_i \in S} (1-\lambda) \cdot \frac{\min_{P_j \in R} |P_j|}{|P_i|} + \lambda \cdot \left(1 - \max_{P_j \in S, P_i \neq P_j} \frac{|P_i \cap P_j|}{|P_i \cup P_j|}\right) s.t. |S| = k \tag{1}$$

Each path P is represented as a set of entities and properties. This allows us to compute the path lengths ($|P|$) and Jaccard similarity using set intersection and union sizes. In order to bring the path lengths to the same range ($[0,1]$) as the Jaccard similarity value, it is normalized by computing the ratio between each path's length and the shortest path length in the result set R. Note that the objective function $DivP(R)$ in Equation 1 is agnostic to the number of properties and entities in the paths and we can also replace the Jaccard similarity metric with any other similarity metric.

THEOREM 1. *The objective function for $DivP(R)$ in Equation 1 is submodular.*

PROOF. The proof follows from the Maximal Marginal Relevance problem proposed in [3]. □

COROLLARY 1.1. *A greedy algorithm for $DivP(R)$ guarantees $(1 - \frac{1}{e})$-Approximation.*

PROOF. Follows from the results in [10]. □

This allows us to design a greedy approximation algorithm that iteratively selects a path P_i in each step that makes the maximum contribution to the objective in Equation 1.

3. PATH DISCOVERY WITH JEDI

Using JEDI, users can explore paths for various use cases. In this section, we focus on a specific use case using the YAGO3 [8] KG and show how JEDI can be used to explore it.

Consider query Q1 in Listing 1. With Q1 it is possible to find links between the Danish and Swedish royal families. Q1 includes two property path patterns that start from the same common ancestor and end with the Danish Queen and the Swedish King, respectively. The property path pattern *pp1* is satisfied by any entity X that is connected directly to `:Margrethe_II_of_Denmark` through the property `:hasChild`, i.e., if the triple `X :hasChild :Margrethe_II_of_Denmark` is in the KG, or if X is connected directly to another entity Y that is connected (directly or through additional entities) to `:Margrethe_II_of_Denmark`. With the property path pattern *pp2*, the condition that the entity X should also be connected in a similar way to `:Carl_XVI_Gustaf_of_Sweden` is imposed.

Using the JEDI Web interface, Q1 can be input using the simple query mode as shown in Figure 3 (b). In this mode, the use of the $\rightarrow$ operator is activated by ticking the checkbox next to the pattern (Figure 3 (c)). To obtain the top k diverse results, k can be specified through the field highlighted in Figure 3 (e) – 3 in this example. Moreover, the value of λ can be specified through the field highlighted in Figure 3 (d). In this example, $\lambda = 0.2$.

The top-3 most diverse results are obtained and combined into a graph that is shown in Figure 3 (a). In Figure 3, the user has selected a particular result, which is consequently highlighted both in the graph representation as well as in the results table (Figure 3 (a) and Figure 3 (f)). The orange path marks the path to the Danish Queen (first property path in the query, *pp1*), and the green path marks the path to the Swedish King (second property path in the query, *pp2*). Note that `Princess_Margaret_of_Connaught` is present in both paths of the selected row as it represents the connection between the two paths corresponding to `?a` in the given query. In general, nodes that belong to more than one path are highlighted using the color that results from mixing the colors of the involved paths. In this case, it is highlighted in a mix of the green and orange. It is also possible to edit the color for each result path manually (Figure 3 (g)). Moreover, it is also possible to select multiple rows, and to reset the selection.[6]

JEDI is, of course, not restricted to this use case but can be used for numerous different use cases other than finding common ancestors between royal families, e.g., finding connections between politicians and researchers, etc.

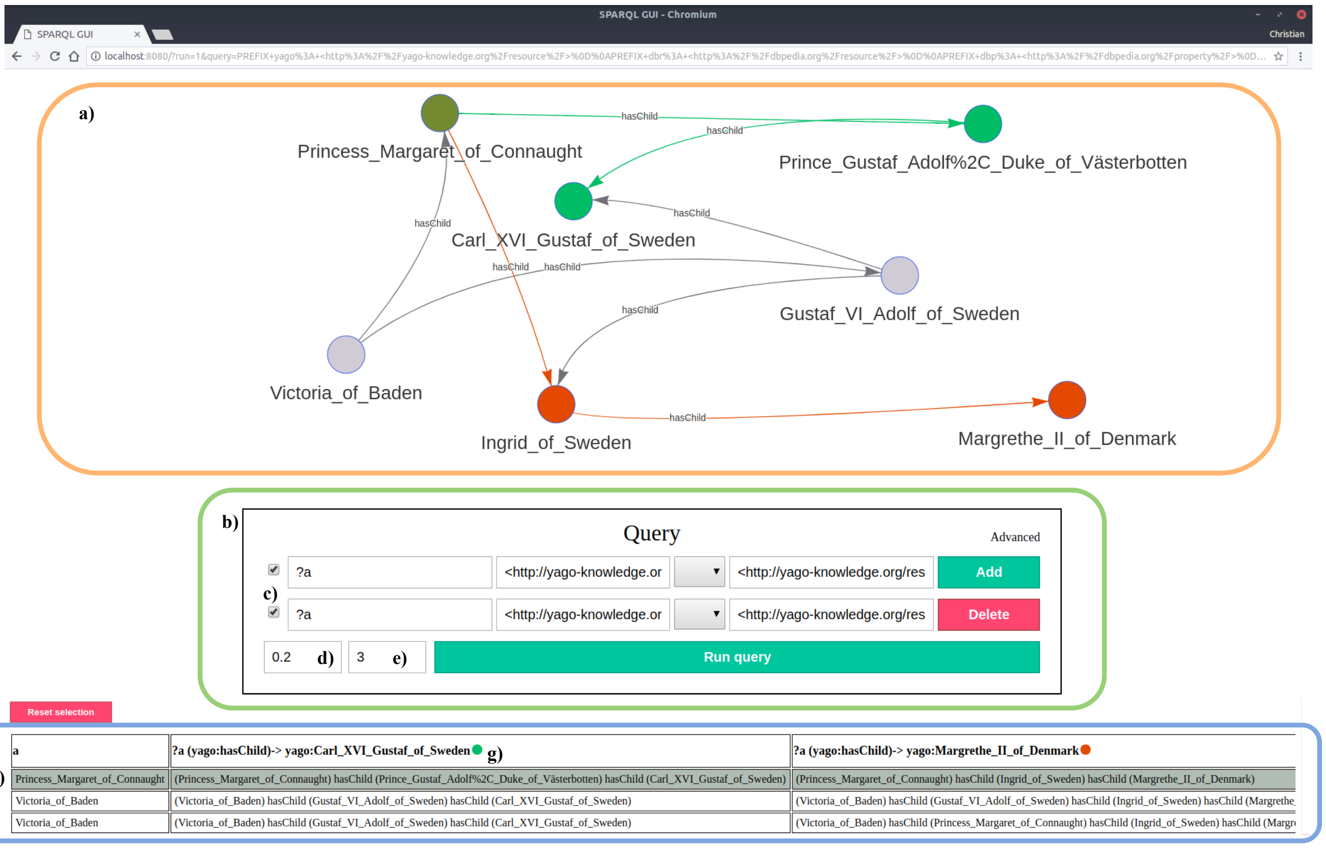
4. DEMONSTRATION

At the conference, we will demonstrate JEDI using a number of selected KGs. Although JEDI can work with any KG – as its implementation only relies on the availability of a SPARQL endpoint to access the data (supporting endpoints hosted on other servers) – for the demonstration we will run JEDI and the SPARQL endpoint on the same machine to avoid any problems caused by unstable Internet connections during the conference.

To ease interaction with the system, we will prepare several interesting SPARQL queries involving property paths. Conference attendees will then have the opportunity to build upon these example queries by editing parameters and exploring the results. In addition, experts are invited to freely formulate SPARQL queries using the extended property path operator proposed in this paper. After JEDI has computed and displayed the results, including their graphical representation, users will be able to freely explore the results by selecting specific paths, marking them in the graph, and in doing so discover connections between specific entities.

One of the KGs that we will use for the demonstration is YAGO3 [8]. It provides information extracted from Wikipedia and contains 1+ billion triples covering a broad range of topics. This dataset, for example, supports the queries and use cases outlined in Section 3. In addition to YAGO3, we will also prepare queries over other KGs: DBLP[7] and DBpedia[8]. These KGs support queries about cross domain knowledge, such as "Brazilian book authors that have been influenced by French writers" or queries about specific domains, such as "How are authors of papers connected to track chairs via co-author relationships?". DBpedia provides information extracted from Wikipedia in a wide range of topics including facts about persons, places, and organizations. For the demonstration, we use the English DBpedia 2016-04 with over 396 millions facts. DBLP consists of bibliographic information about sci-

[6] A video of JEDI is available at http://qweb.cs.aau.dk/jedi/
[7] https://dblp.org/
[8] http://dbpedia.org/


SPARQL GUI - Chromium
SPARQL GUI
Christian
localhost:8080/?run=1&query=PREFIX+yago%3A+<http%3A%2F%2Fyago-knowledge.org%2Fresource%2F>%0D%0APREFIX+dbr%3A+<http%3A%2F%2Fdbpedia.org%2Fresource%2F>%0D%0APREFIX+dbp%3A+<http%3A%2F%2Fdbpedia.org%2Fproperty%2F>%0D...
a)
Princess_Margaret_of_Connaught
Prince_Gustaf_Adolf%2C_Duke_of_Västerbotten
Carl_XVI_Gustaf_of_Sweden
Gustaf_VI_Adolf_of_Sweden
Victoria_of_Baden
Ingrid_of_Sweden
Margrethe_II_of_Denmark
hasChild
b)
c)
d)
e)
f)
g)

Query | Advanced

✓	?a	<http://yago-knowledge.or	▾	<http://yago-knowledge.org/res	Add
✓	?a	<http://yago-knowledge.or	▾	<http://yago-knowledge.org/res	Delete

0.2 | 3 | Run query

Reset selection

a	?a (yago:hasChild)-> yago:Carl_XVI_Gustaf_of_Sweden ●	?a (yago:hasChild)-> yago:Margrethe_II_of_Denmark ●
Princess_Margaret_of_Connaught	(Princess_Margaret_of_Connaught) hasChild (Prince_Gustaf_Adolf%2C_Duke_of_Västerbotten) hasChild (Carl_XVI_Gustaf_of_Sweden)	(Princess_Margaret_of_Connaught) hasChild (Ingrid_of_Sweden) hasChild (Margrethe_II_of_Denmark)
Victoria_of_Baden	(Victoria_of_Baden) hasChild (Gustaf_VI_Adolf_of_Sweden) hasChild (Carl_XVI_Gustaf_of_Sweden)	(Victoria_of_Baden) hasChild (Gustaf_VI_Adolf_of_Sweden) hasChild (Ingrid_of_Sweden) hasChild (Margrethe_
Victoria_of_Baden	(Victoria_of_Baden) hasChild (Gustaf_VI_Adolf_of_Sweden) hasChild (Carl_XVI_Gustaf_of_Sweden)	(Victoria_of_Baden) hasChild (Princess_Margaret_of_Connaught) hasChild (Ingrid_of_Sweden) hasChild (Margr

Figure 3: JEDI's graphical interface when obtaining Q1's top-3 paths with $\lambda = 0.2$

entific publications in the domain of computer science. For the demonstration we use DBLP version 2018-02-01 containing over 76 million facts.

5. CONCLUSION

In this paper, we have presented JEDI, a system that enhances an existing SPARQL query engine, Jena, to ease knowledge graph exploration using property path patterns that, differently from the SPARQL 1.1 standard, output the paths connecting entities. The generated paths comply with diversification metrics to produce the k most interesting results satisfying the user's query. The effectiveness of JEDI's diversification metrics have been evaluated in [1], showing the impact of the similarity measures on the obtained top-k paths. In the current implementation, diversification is implemented as a post-processing phase. However, given that the number of paths connecting entities in a KG can be very large, we plan to integrate the diversification metric into the path generation step in our future work. Doing so, will reduce space requirements and increase JEDI's performance.

Acknowledgments

This research was partially funded by the Danish Council for Independent Research (DFF) under grant agreement no. DFF-4093-00301 and Aalborg University's Talent Management Programme.

6. REFERENCES

[1] C. Aebeloe, V. Setty, G. Montoya, and K. Hose. Top-K Diversification for Path Queries in Knowledge Graphs. In *ISWC'18*, 2018.
[2] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives. DBpedia: A Nucleus for a Web of Open Data. In *ISWC'07*, pages 722–735. Springer, 2007.
[3] J. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR'98*, pages 335–336, 1998.
[4] V. Fionda and G. Pirrò. Explaining and querying knowledge graphs by relatedness. *PVLDB*, 10(12):1913–1916, 2017.
[5] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194:28–61, 2013.
[6] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Processing Aggregate Queries in a Federation of SPARQL Endpoints. In *ESWC'15*, pages 269–285, 2015.
[7] D. Ibragimov, K. Hose, T. B. Pedersen, and E. Zimányi. Optimizing Aggregate SPARQL Queries Using Materialized RDF Views. In *ISWC'16*, pages 341–359, 2016.
[8] F. Mahdisoltani, J. Biega, and F. Suchanek. YAGO3: A knowledge base from multilingual wikipedias. In *CIDR'14*, 2014.
[9] G. Montoya, H. Skaf-Molli, and K. Hose. The Odyssey Approach for Optimizing Federated SPARQL Queries. In *ISWC'17*, pages 471–489, 2017.
[10] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions–I. *Mathematical Programming*, 14(1):265–294, 1978.
[11] V. Savenkov, Q. Mehmood, J. Umbrich, and A. Polleres. Counting to k or how SPARQL1.1 Property Paths Can Be Extended to Top-k Path Queries. In *SEMANTICS'17*, pages 97–103, 2017.
[12] S. Seufert, P. Ernst, S. J. Bedathur, S. K. Kondreddi, K. Berberich, and G. Weikum. Instant Espresso: Interactive Analysis of Relationships in Knowledge Graphs. In *WWW '16 Companion*, pages 251–254, 2016.