

Similarity-based Knowledge Graph Queries for Recommendation Retrieval

Lisa Wenige*, Johannes Ruhland

Chair of Business Information Systems, Friedrich-Schiller-Universität Jena, Germany

E-mail: lisa.wenige@uni-jena.com

Abstract. This paper investigates how similarity-based retrieval strategies can be combined with graph queries to enable users or system providers to explore repositories in the Linked Open Data (LOD) cloud more thoroughly. For this purpose, we developed a content-based recommender system (RS). It relies on concept annotations of Simple Knowledge Organization (SKOS) vocabularies and a SPARQL-based query language that facilitates advanced and personalized requests for openly available and interlinked datasets. We have comprehensively evaluated the novel search strategies in several test cases and example application domains (i.e., travel search and multimedia retrieval). The results of the web-based online experiments showed that our approaches increase the recall and diversity of recommendations or at least provide a competitive alternative strategy of resource access when conventional methods do not provide helpful suggestions. The findings may be of use for Linked Data-enabled recommender systems as well as for semantic search engines that can consume LOD resources.

Keywords: Recommender Systems, Linked Open Data, Information Retrieval, SPARQL, Semantic Search, SKOS

1. Introduction

A recommender system (RS) component is usually one of the key features in online portals. It helps users to discover items that reflect their interests [3]. Personalized recommendations are based on a user profile that contains either implicit (e.g., access statistics or click behavior) or explicit preference information (e.g., ratings for items). Common RS techniques are collaborative filtering (CF) or content-based (CB) algorithms. CF approaches derive suggestions from users with similar tastes, whereas CB methods are based on similar items according to metadata descriptions [26]. CB engines often face the problem that these descriptions are insufficient to generate useful suggestions [1]. Hence, given the abundance of metadata in the Linked Open Data (LOD) cloud [50], it is not surprising that RS researchers have begun to harness these data sources to tackle data sparsity issues. Recommender systems operating at least partly on RDF resources are called Linked Data Recommender Systems (LDRS) or LOD-enabled recommender systems. Some stud-

ies have shown that they can compete with traditional systems and help to overcome shortages in the local database. Researchers have applied content-based approaches by utilizing LOD metadata for offline computation of item similarities [12, 13, 27, 39]. This means that, prior to the actual process of generating recommendations, data must be extracted from the LOD cloud. Thus, the RS operator has to set up data models representing both the user profile file and the features of the items before the system runs. It may happen that user preferences are too simplistically represented in the system. Hence, despite the comprehensiveness and expressiveness of RDF graphs, metadata descriptions are still structured in the table-like format of attribute-value pairs [12, 13, 22, 24, 29, 47, 61]. Although, this approach is effective in terms of data enrichment (after all, it is possible to fill in the gaps in the local database), the consequence is that the central strengths of the LOD cloud are not fully exploited. The flat data structure reduces the richness of information from the RDF graph and prevents that the full potential of LOD enhancement can be realized for recommender systems or personalized search in LOD repositories. Consider the following example for illustration. Sup-

*Corresponding author. E-mail: lisa.wenige@uni-jena.com.

pose a user has stated that he/she likes a particular Indie rock band. A conventional LDRS would extract LOD-based metadata descriptions for the given band and determine similar bands from the same item feature table. However, it may well be that the descriptions contain irrelevant information, e.g., just the place of foundation or the founding year of the band, instead of the genre and the style of music they usually play. Such metadata would not yield useful results in a purely content-based system and only achieve a few random hits. On the other hand, a recommendation request, which takes into account the band's position in the RDF graph, has a higher chance of identifying relevant suggestions. For instance, a graph-based query could explore the music items (e.g., albums or songs) released by the particular band and extract metadata for these items. For this purpose, labeled links can be exploited to determine the respective descriptions in the RDF graph. The data web can answer such requests because the LOD technology stack provides the SPARQL Protocol and RDF Query Language (SPARQL) and suitable query engines that enable fast retrieval [52]. However, graph-based queries alone are not yet sufficient to generate personalized suggestions, since they do not consider user preferences. What is missing is a tool that can combine graph-based query elements with similarity calculation. For this purpose, it is important that recommendations can be generated on-the-fly. Only then, a system will be able to switch between processing of graph- and search-based query parts. To the best of our knowledge, there does not yet exist a recommendation engine that can perform these tasks on RDF data. Instead, current LDRS rely on a fixed set of item features, for whom values have to be extracted from the LOD cloud before suggestions can be generated. In this way, the strengths of graph-based querying, unfortunately, remain largely unused. A purely SPARQL-based request, on the other hand, only returns results for graph patterns that exist in the repository. Consider the following example: Consumers may like to use recommendation engines that work on multiple domains simultaneously. For instance, a user, who has stated that he/she likes certain items from one domain may like to receive suggestions from another domain as well (i.e., for cross-domain recommendation retrieval). In case a user profile contains feedback information for books, it would be desirable if the system could generate movie suggestions based on this data. While the approach of querying RDF graphs can be useful for this recommendation scenario (since it can identify matching objects based

on entity type declarations or typed link information), it may also be the case that there are no direct links from a preferred book to a suitable movie in the RDF graph. Therefore, RS designers should not rely too heavily on graph-based queries, but also explore possibilities of similarity-based retrieval. A possible solution would be to identify books that are similar to the ones in the user profile. In a subsequent processing step and through a graph-based query, the engine could explore whether the similar books have any connections to movie items in the repository. This approach can reveal implicit connections in the data and might help to return fewer empty result sets to the user. Therefore, the system should process the data online. This is because one processing step (identification of suitable movies through graph pattern matching) relies on the outcome of a preceding one (computation of similar book items). Almost none of the existing LDRS and LOD-enabled search systems addresses these issues, which prevents efficient usage of available knowledge sources for retrieval tasks.

The LOD cloud stands out for its open availability, the wealth of data sources and the expressive power of the RDF data model. It requires methods and tools that can mine the data effectively to improve recommendation results. On the other hand, however, accessing LOD repositories also has some challenges. It is a problem that many collections utilize different RDF-based vocabularies [50]. This causes data heterogeneity and complicates on-the-fly retrieval since the engine might not be able to process all types of LOD resources right away. Fortunately, there exist de-facto standard vocabularies that are widely enough used to be applicable to ad-hoc item-to-item similarity calculation. In this context, a key standard is the Simple Knowledge Organization System (SKOS) vocabulary [38]. SKOS vocabulary concepts often annotate items in LOD repositories. Thus, they help to describe real-world entities. For instance, in the DBpedia repository, an LOD resource representing a movie is characterized by SKOS-based subject descriptors (e.g. stating the genre, topic or most important actors of the movie). The descriptors are part of the DBpedia category graph [11]. SKOS subjects are uniquely identified LOD resources. Therefore, they can be conveniently processed for ad-hoc querying [38].

This paper presents a SKOS-based recommendation engine, called SKOSRecommender (SKOSRec), that combines on-the-fly similarity calculation and SPARQL-like techniques to leverage the full potential of LOD repositories for retrieval tasks. It is structured

as follows: Section 2 gives an overview over the most important related approaches from the fields of information retrieval (IR), RS and LDRS and will identify open research questions in these areas. Section 3 introduces the SKOSRec engine's key techniques and illustrates in which ways they can help to alleviate personalized retrieval from LOD repositories. In this context, example recommendation tasks demonstrate the feasibility of the approaches. Therefore, the application domains of travel destination search and multimedia retrieval (movie, music and book recommendations) serve as showcase scenarios since these domains are among the most prevalent types of LOD-enabled RS [59]. The LOD repository DBpedia provides the respective metadata descriptions because it has already been successfully applied for LOD-enhancement in the context of recommendation retrieval [12, 13, 16]. Subsequent to this, Section 4 presents the results of a series of web-based user experiments in the selected usage scenarios in which the developed approaches were tested. It will be clarified whether the new types of recommendation queries of the SKOSRec engine actually have an added value for users. Finally, Section 5 summarizes the most important findings and will outline future research directions.

2. Related Work

Effective answering of retrieval requests has been investigated by researchers for many years. Even before the evolution of the LOD cloud, scientists developed strategies for extracting relevant information from text data. The most prevalent IR engines perform relevance rankings for unstructured resources and offer means to state filter conditions that complement queries in a faceted search interface. The approaches can combine query constraints with similarity-based retrieval. In these systems, user requests are quickly answered due to extensive preprocessing and indexing of resources before runtime execution [55]. However, low latency of search results comes at the cost of a hard-wired data model which prevents individual customizations and causes limitations concerning up-to-date retrieval [18].

Other researchers have developed strategies for answering customized recommendation requests from structured data sources. For instance, Adomavicius et al. propose the REQUEST query language that is supposed to generate suggestions from a relational database that contains information on both user pro-

files and items. Thus, personalized recommendation queries can be formulated. However, a disadvantage of this method is that it fortifies data sparsity issues since restrictive requests reduce the item space even further, which may be a problem when the database already contains too few high quality data sources [2]. Another interesting approach in the category of recommendation retrieval over relational databases is the FlexRecs system by Koutrika et al. The key advantage of their course RS is that it enables highly individual requests in which different query parts (i.e., profile generation, filtering of recommendations or like-minded peers) can be combined into so called recommendation workflows. Hence, highly individual recommendation queries can be formulated with FlexRecs [32].

Despite the strengths and weaknesses of the discussed approaches, it has to be taken into account that they are not designed to consume LOD. However, LOD-enabled retrieval and recommender systems, often do not make use of the full potential of RDF data as well. For instance, among the many engines that are designed to search LOD resources [4, 9, 10, 20, 25, 40, 45, 58], there are only a few that facilitate expressive graph-based facets to filter results [4, 45]. Another interesting research direction in the category of LOD-enabled retrieval engines involves techniques of ad-hoc query processing through spreading activation algorithms. These systems can process information from the LOD cloud right away. While this is a considerable advantage in terms of fast retrieval, the downside of these approaches is that the application of graph-based query constraints is only possible to a limited extent due to the applied probabilistic path traversal techniques [16, 17, 28, 33, 35, 53]. Existing Linked Data Recommender Systems, on the other hand, mostly perform offline computation of item-to-item similarities from LOD metadata descriptions. While this approach offers more control over the retrieval process, it prevents individual customizations and runtime responses [12, 13, 22, 24, 29, 47, 61].

There are only a few LDRS that take advantage of RDF graph structures and are query-based. However, these systems fall short regarding scalable on-the-fly similarity calculation [7, 46] and profile-based retrieval from SPARQL endpoints. The latter shortcoming can be attributed to the fact that in the existing query-oriented LDRS, user preferences and metadata information reside in the same repository [5, 43]. However, this goes against the notion of the LOD cloud as a decentralized and openly accessible data space, which can be

queried through API-like interfaces.

In summary, existing non-Linked Data as well as Linked Data-enabled retrieval and RS engines already provide useful features for personalized resource access. Nevertheless, the full potential of LOD for recommendation and search tasks has yet to be realized. The novel query strategies of the SKOSRec engine are attempts to tackle the previously outlined research gaps to improve existing RS as well as to offer new search strategies for LOD repositories that are guided by personal preferences.

3. The SKOS Recommender

3.1. Architecture

Figure 1 shows the architecture of the SKOSRec engine. The system's key components were developed with the help of the Apache Jena framework.¹ The system is designed to consume SPARQL-like recommendation requests. Thus, a compiler regulates the correct execution of critical operations of the engine (e.g., graph-based filtering, result set joins, similarity calculation).

Another important feature of the architecture is the SKOSRec parser. It checks the syntactic correctness of recommendation queries. Since the SKOSRec parser utilizes SPARQL 1.1 syntax elements, a SPARQL JavaCC grammar definition served as the skeleton for the grammar of the parser. Therefore, we adapted and enhanced existing components of the SPARQL specification which can be found in the software package ARQ.² The JavaCC (Java Compiler Compiler) library was applied for this purpose. It is a tool that generates Java parsers by conversion of grammar specifications to Java source code.³ Thus, the SKOSRec parser was semi-automatically developed from a respective syntax specification.

The engine's similarity calculation unit is almost as equally important as the parser and compiler since it generates suitable suggestions for user preferences. Besides the query processing and similarity calculation features, other system components are utility tools. They help to alleviate the retrieval process. Features, such as the ranker (ordering recommendations accord-

ing similarity scores), the optimizer (optimization of the retrieval procedure) and data processor help to quickly handle LOD resources. From Figure 1, it can be seen that the engine can interact with both local and remote LOD servers.

In the standard configuration, the system processes RDF data from a default SPARQL endpoint that is accessed over HTTP. Aside from the LOD repositories containing item feature data, the engine needs to process SKOS vocabulary information (i.e., the SKOS thesaurus, SKOS mappings to other vocabularies, similarity values of SKOS concept pairs for concept expansion). These data sources either reside in local or external repositories and can be accessed through SPARQL endpoints over HTTP as well.

Each SKOSRec instance loads a configuration object. It defines the means of access to the default LOD repository. It is also specified, with which SKOS vocabularies the LOD repository is associated. In case the system runs in cross-repository mode (i.e., triggered by a *ServiceIntegration* statement, see Subsect. 3.2), the configuration object states an additional (potentially remote) LOD repository as well as the location of the SKOS mappings. Thus, the engine can correctly execute similarity calculation for distributed LOD repositories.

The architecture does not dictate a particular SPARQL server implementation since the system is decoupled from the endpoint that is accessed during retrieval. However, the prototypical implementation of the SKOSRec engine utilizes the OpenLink Virtuoso server.⁴

3.2. Query Syntax

The query handler is the backbone of the SKOS- Recommender. A SKOSRec requests triggers a recommendation workflow, in which similarity-based retrieval steps can be flexibly combined with SPARQL-like filter expressions. Listing 1 shows the syntax of the SKOSRec query language. In the given grammar, underlined parts represent SPARQL syntax elements that have been directly taken from the latest W3C specification [21]. Expressions in capital letters denote query keywords. As a regular SPARQL SELECT query, a SKOSRec query always generates a table-like result set that contains at least a mapping to a single variable.

¹<https://jena.apache.org>

²https://github.com/apache/jena/blob/master/jena-arq/Grammar/sparql_11.jj

³<https://javacc.org/>

⁴<https://virtuoso.openlinksw.com/>

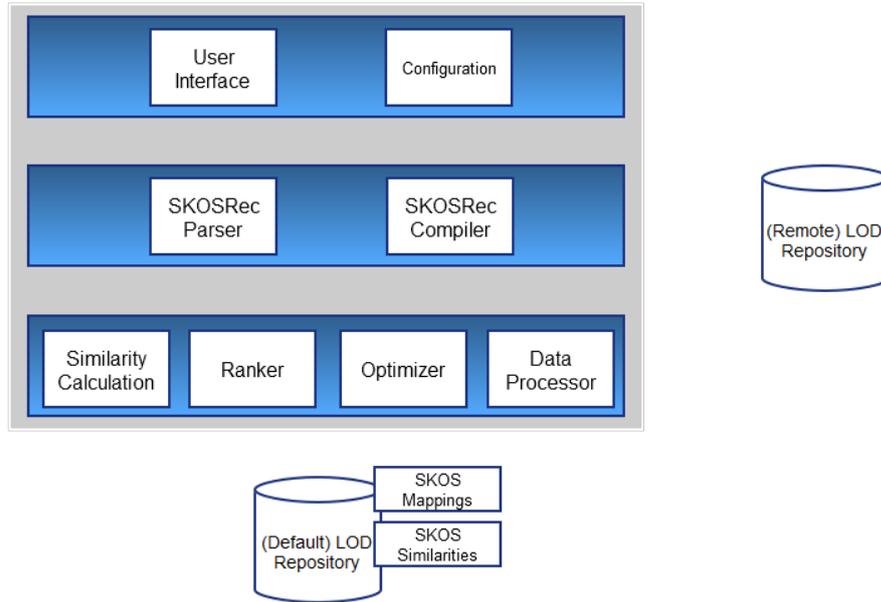


Fig. 1. Architecture of the SKOSRec engine

A SKOSRec query can be comprised of up to six elements, of which the parts *SimProjection* and *ItemPart* are obligatory. Hence, users need to state at least their preferences (*ItemPart*) and how many recommendations they would like to receive (*SimProjection*). Advanced retrieval patterns can be formulated as well. For instance, similar resource retrieval can be combined with prefilter (*RecWhereClause*) and/or postfilter (*SelectPart*) constraints for LOD resources. In case the query contains a postfilter statement, it can be formulated as an aggregation-based request (*Aggregation*). As in SPARQL, the query may start with a prologue that abbreviates IRIs with namespace declarations [21]. The SKOSRec engine can retrieve suggestions either from a default SPARQL endpoint stated in the configuration of the engine or from other potentially remote SPARQL endpoints with additional collections. In case, users intend to receive recommendations from a non-default repository, they have to state the SPARQL endpoint URL, under which the collection can be accessed in the *ServiceIntegration* part of the query. In a SKOSRec request, the user profile is set up with *ItemPart* statements. The profile needs to contain at least a single *ItemPart* declaration, but can also be comprised of many such statements. The *ItemPart* is either a preference for a certain

item (e.g., a movie) represented by a LOD resource (*IRI*) or a SPARQL-like subquery based on which preferred LOD resources are determined for the profile (*VarPart*). For each *VarPart* statement, the SKOSRec compiler needs to make sure that the variable (*Var*) occurs in the *WHERE* condition (*RecWhereClause*) of the SPARQL-like subquery as well. The same holds for any pre- or postfilter constraints. These specifications also contain a *RecWhereClause*. For prefilter constraints, the variable represents potentially relevant LOD resources, while for postfilter statements it stands for generated suggestions. Both need to be present in the graph pattern(s) of the *RecWhereClause* to enable the system to correctly apply the filter statements on the set of retrieved resources. In case, the *RecWhereClause* comprises a *RecMinusGraphPattern*, it also has to be checked that the variable is not only contained in the *MINUS* part of the group. The *RecWhereClause* closely resembles the *WhereClause* of the SPARQL 1.1 specification with all its subsequent parts. Hence, it enables different combinations of graph pattern matching. However, the *RecWhereClause* of the SKOSRec language allows fewer pattern matching expressions than the *WhereClause* of the regular SPARQL syntax specification [21] (see Listing 1). For instance, it is not permitted to formulate subqueries in this section. This

modification has been made to ensure that the process of similarity calculation can be correctly carried out. Since the SKOSRec engine computes recommendations with the help of the SKOS annotations of the LOD resources stated in the user profile, the knowledge graph structures of SKOS thesauri can be exploited by incorporating additional relevant SKOS concepts into the retrieval procedure (i.e., concept expansion). These additional concepts can be identified by concept-to-concept similarity scores that are determined according to a similarity metric for knowledge organization systems (KOS). The functioning of this approach, called flexible similarity detection, is described in detail in [60]. Users can trigger a similar resource retrieval procedure by a *Sim* statement that contains a concept-to-concept similarity threshold either stating that scores should be „larger“, „larger than“ or „equal to“ the threshold value.

3.3. On-the-Fly Recommendations

A simple on-the-fly request is the most basic form of a SKOSRec query. It is based on the engine’s ability generate ad-hoc recommendations from a user profile containing preference statements for LOD resources. After parsing an on-the-fly request, the system identifies SKOS annotations by matching a suitable property (i.e., annotation property) in an RDF dataset. (Definition 1).

Definition 1 (Annotation property). *An annotation property is an IRI (Listing 2), that is defined in the DCMI specification for subject annotations. It can occur in conjunction with one of the two namespaces, which are stated by the standard [15].*

Listing 2: Annotation property

```
<ANNOT.PROP> ::= dc:subject | dct:subject
```

These properties help to retrieve SKOS annotation triples for items preferred by a user (Definition 2).

Definition 2 (SKOS annotation triple). *A SKOS annotation triple (*st*) is a triple that is comprised of an IRI in the subject position, an annotation property (ANNOT.PROP) as predicate and a concept *c* in the object position. The concept is part of a knowledge organization system *S* in SKOS format ($C = \{c \mid c \in S\}$) (Eq. 1).*

$$st \in I \times \langle ANNOT.PROP \rangle \times C \quad (1)$$

The identification of SKOS annotation triples facilitates a fast retrieval of potential similar items. Therefore, the engine evaluates shared features between resources from the SKOS annotation dataset (Definition 3).

Definition 3 (SKOS annotation dataset). *A SKOS annotation dataset (*AD*) is a subset of an RDF dataset that contains SKOS annotation triples (Eq. 2).*

$$AD \subset I \times \langle ANNOT.PROP \rangle \times C \quad (2)$$

Before the similarity calculation process starts, the engine determines SKOS annotations for each LOD resource (*r*) from the user profile by issuing a subsequent SPARQL query to the LOD repository from which the user intends to receive suggestions. The notion of SKOS annotations is specified in Definition 4.

Definition 4 (SKOS annotations). *In the annotation dataset, LOD resources directly link to concepts of a SKOS system via a predefined annotation property. SKOS annotations for an input resource *r* are defined as in Equation 3.*

$$Annot(r) = \{c \in S \mid \exists (r, \langle ANNOT.PROP \rangle, c) \in AD\} \quad (3)$$

Afterward, the engine identifies which of the other resources in the dataset shares annotations with it. Similarities only have to be computed for items with matching concepts [13, 49]. Therefore, triples can be efficiently joined along item features, since native RDF storage systems usually index triples rather than columns [41]. Thus, the quick identification of mutual annotations through SPARQL requests facilitates ad-hoc similarity calculation. Hence, the method reduces the number of triple statements to be processed and prevents high throughput rates. Definition 5 specifies the notion of relevant resources and their annotations that are needed to identify similar items. The applied syntax follows Pérez et al. [42].

Definition 5 (Relevant resources and their annotations). *The conjunctive graph pattern P_r matches all items and subjects that are potentially relevant for recommendation retrieval (Eq. 4).*

$$P_r = (r, \langle ANNOT.PROP \rangle, ?c) \text{ AND } (?q, \langle ANNOT.PROP \rangle, ?c) \quad (4)$$

Listing 1 Grammar of the SKOSRec query language

```

SKOSRecQuery ::= Prologue SelectPart? SimProjection PREF ItemPart+ RecWhereClause?
SelectPart ::= SELECT (DISTINCT | REDUCED)? Var+ RecWhereClause
LimitClause? Aggregation?
Aggregation ::= AGG IRIref Var (SUM | MAX | AVG )
SimProjection ::= RECOMMEND Var TOP INTEGER ServiceIntegration?
ServiceIntegration ::= FROM SERVICE IRIRREF
ItemPart ::= ( VarPart | IRI ) Sim?
VarPart ::= [ Var RecWhereClause ]
Sim ::= SIM Relation DECIMAL
Relation ::= ( > | >= | = )
RecWhereClause ::= WHERE RecGroupGraphPattern
RecGroupGraphPattern ::= '{' RecGroupGraphPatternSub '}'
RecGroupGraphPatternSub ::= TriplesBlock? ( RecGraphPatternNotTriples '.' ? TriplesBlock? )*
RecGraphPatternNotTriples ::= RecGroupOrUnionGraphPattern | RecOptionalGraphPattern |
RecMinusGraphPattern | Filter | Bind | InlineData
RecGroupOrUnionGraphPattern ::= RecGroupGraphPattern ( 'UNION' RecGroupGraphPattern )*
RecOptionalGraphPattern ::= 'OPTIONAL' RecGroupGraphPattern
RecMinusGraphPattern ::= 'MINUS' RecGroupGraphPattern

```

The mapping Ω_r of relevant resources and their SKOS annotations is obtained by retrieving all resources q that share at least one SKOS concept c with resource r from AD (Eq. 5). The corresponding result set contains mappings for all variables (i.e., $?c$ and $?q$) in the triple statements of P_r .

$$\Omega_r = [[P_r]]_{AD} \quad (5)$$

Upon extraction of relevant resources and annotations, the process of similarity calculation can start. The engine determines the IC of the shared SKOS annotations of two resources. This approach is based on the hybrid similarity metric proposed by Meymandpour and Davis for LOD-enabled RS [37]. However, while their measure considers all types of IRI resources for the retrieval, our similarity metric purely relies on SKOS annotations (Definition 6).

Definition 6 (SKOS similarity). *Let $Annot(r)$ be the set of SKOS features of resource r and $Annot(q)$ the set of SKOS features of resource q and $q \in \{\mu(?q) \mid \mu \in \Omega_r\}$, then their similarity can be derived from the IC of their shared concepts $C_{shared} = Annot(r) \cap Annot(q)$*

$$sim(r, q) = IC(C_{shared}) \quad (6)$$

The IC of a set of SKOS concepts is measured by the sum of the inverse logarithms of each concept's frequency in relation to the maximum frequency among relevant resources (Definition 7).

Definition 7 (SKOS Information Content). *The SKOS IC is defined as an aggregation of the individual IC*

values of each concept that is contained in the set of shared features ($c \in C_{shared}$), where $freq(c)$ is the frequency of c among all relevant resources and n is the maximum frequency among these resources. The final similarity score of two resources r and q is determined by summing the IC values of each concept of the shared feature set.

$$IC(C_{shared}) = - \sum_{c \in C_{shared}} \log \left(\frac{freq(c)}{n} \right) \quad (7)$$

After having obtained resource annotations as well as IC values, similarity scores between the input resource r and each potentially relevant resource q can be calculated. When a user profile contains more than a single item, similarity values are aggregated through summation to determine the final recommendation score (Definition 8).

Definition 8 (Recommendation score). *The recommendation score of a potentially relevant resource q is quantified by the sum of similarities with each resource r that can be found in the profile (Pr).*

$$score(Pr, q) = \sum_{r \in Pr} sim(r, q) \quad (8)$$

Upon calculation of similarity values for each potentially relevant item q , the engine ranks the retrieved resources based on a given limit (k) that is specified by the user in the SKOSRec query.

With the presented methods of on-the-fly retrieval, recommendations can be generated from LOD repository

ries through an API interface without requiring local metadata or excessive preprocessing operations other than the mapping of profile items with LOD resources.⁵ Listing 3 depicts an example SKOSRec on-the-fly request that obtains suggestions based on a user’s preference for the western movie „They Call Me Trinity“. In the given query, the preferred movie is represented by the DBpedia resource (`dbr:They_Call_Me_Trinity`). The shown query also contains a prefix with a namespace declaration and a specification regarding the number of suggestions the engine should generate.

Listing 3: A simple recommendation query for the movie domain (Q1)

```
PREFIX dbr: <http://dbpedia.org/resource/>
RECOMMEND ?movie TOP 5
PREF dbr:They_Call_Me_Trinity
```

In this basic form, the query does neither contain any pre- or postfilter conditions nor a preference query statement. Thus, the engine simply determines all movies that are similar to the preference in the profile. When executed over DBpedia, the query returns the solution set that is depicted in Table 1.

Table 1
Result set for Q1

| ?movie |
|-------------------------------|
| dbr:God_Forgives...'_I_Don't! |
| dbr:Ace_High_(1968_film) |
| dbr:Boot_Hill_(film) |
| dbr:Trinity_Is_Still_My_Name |
| dbr:Troublemakers_(1994_film) |

3.4. Preference Querying

In the previous subsection, it was assumed that users express their tastes in the form of preference statements for a certain item in the form of an *IRI* referring to a LOD resource. However, it may also be the case that likings are only vaguely known, for instance, when a user has not yet interacted with an online plat-

⁵Di Noia et al. describe, how a mapping procedure can be conducted for a real-world implementation. For their LDRS, they matched movie titles with the corresponding LOD resources in DBpedia. The authors extracted IRIs, labels and publication dates for all movie resources. They applied the Levenshtein distance on these resources to identify matching movies items [13].

form but can just provide general information, while creating a profile (e.g., regarding genre preferences). Another example is when users prefer products with specific features but are not able or willing to specify the actual items. For instance, consumers are often fans of particular movie directors or book authors. However, on popular commercial platforms, such as Amazon⁶ or Netflix⁷ common retrieval strategies either search items by their creators or obtain recommendations from past consumption behavior. A novel and potentially interesting search paradigm would be to combine both methods in a unified approach. With this approach, a profile is created based on a single user statement, upon which the engine extracts the required information from the LOD repository. This method is equally applicable to databases. However, they might lack persistent authority control which hinders extraction of all the relevant data for a particular entity. On top of that, the data may not be available in the local database, such that LOD repositories can offer valuable additional information sources.

For these reasons, the SKOSRec engine offers the option to use a graph-based request for profile generation from LOD repositories in the *VarPart* section of a SKOSRec query. The notion of a queried profile is specified in Definition 9.

Definition 9 (Queried profile). *A queried profile is a user profile that has been compiled by retrieving all LOD resources r that are part of an annotation graph AD and are contained in the solution mappings resulting from the evaluation of a specified graph pattern (P_{pref}) over a dataset D (Eq. 9).*

$$Pr = \{ \mu(?r) \mid \mu \in [[P_{pref}]]_D, ?r \in \text{dom}(\mu), r \in AD \} \quad (9)$$

The engine obtains such a profile by issuing a subsequent SPARQL query that contains the preference graph pattern (P_{pref}) in the *WHERE* condition. Suppose a user is interested in Quentin Tarantino movies (`dbr:Quentin_Tarantino`), but does not specify the precise items he likes. He could send a preference query to the SKOSRec engine. The graph pattern in the request would match all movies that are con-

⁶<https://www.amazon.de/>

⁷<https://www.netflix.com>

ected to `dbr:Quentin_Tarantino` with the properties `dbo:director` or `dbo:producer` (see Listing 7).

Listing 4: SKOSRec query to generate recommendations based on a preference query in the movie domain (Q2)

```
PREFIX dbr: <http://dbpedia.org/resource/> .
PREFIX dbo: <http://dbpedia.org/ontology/> .

RECOMMEND ?movie TOP 5
PREF [ ?prefMovie
WHERE {
  {?prefMovie dbo:director dbr:Quentin_Tarantino .}
UNION
  {?prefMovie dbo:producer dbr:Quentin_Tarantino .}
} ]
```

Table 3.6 shows the corresponding solution set to the preference query containing films from DBpedia that are similar to Quentin Tarantino movies.

Table 2
Result set for Q2

| ?movie |
|---------------------------|
| dbr:The_Godfather_Part_II |
| dbr:The_Prestige_(film) |
| dbr:The_Dark_Knight |
| dbr:Planet_Terror |
| dbr:Clerks |

3.5. Prefiltering

To facilitate exploratory search in LOD repositories, users should also have the option to formulate conditions in their queries. A possible way to enable filters during retrieval is before similarity calculation. Thus, recommendation lists can be adjusted to individual needs. Due to the expressiveness of RDF data, filter conditions can not only be applied on attributes that are directly connected to potentially relevant items but can also be declared as a graph pattern (P). By this means, it is possible to retrieve LOD resources that are both similar to the user profile as well as fulfill an advanced condition. Before the extraction of relevant resources and annotations, the recommendation engine identifies the resources that satisfy the filter. (Definition 10).

Definition 10 (Prefiltered resources). *The mapping of filtered resources $\Omega_{prefilter}$ is obtained by matching a specified graph pattern (P) to the RDF graph of a dataset D (Eq. 10).*

$$\Omega_{prefilter} = \{ \mu_{|?q} \mid \mu \in [[P]]_D \} \quad (10)$$

After preselection, resources are brought together with user profile data to retrieve the final set of recommendations (Definition 11).

Definition 11 (Annotations and relevant resources (prefiltered retrieval)). *The solution mappings of annotations and relevant resources in prefiltered retrieval mode is obtained by joining the set of prefiltered resources with the mappings Ω_r of all relevant resources and annotations as determined from the user profile.*

$$\Omega_{pre} = \Omega_{prefilter} \bowtie \Omega_r \quad (11)$$

After retrieving annotations and relevant resources in prefiltering mode, the system generates constraint-based recommendations. In this context, the ranking procedure is adapted to the restricted set of relevant resources and annotations. Hence, IC values of matching annotations and respective SKOS similarities are quantified according to the user filter (Definitions 12 and 13). By this means, recommendation scores reflect the similarity of items for the actual set of filtered LOD resources.

Definition 12 (Conditional SKOS similarity). *Let $Annot(r)$ be the set of SKOS features of r and $Annot(q)$ the set of SKOS features of resource q , which is contained in the filtered set of annotations and relevant resources ($q \in \{ \mu(?q) \mid \mu \in \Omega_{pre} \}$), then the similarity can be derived from the IC of their shared concepts ($C_{cond} = Annot(r) \cap Annot(q)$) (Eq. 12).*

$$sim_{cond}(r, q) = IC(C_{cond}) \quad (12)$$

Definition 13 (Conditional SKOS Information Content). *The conditional IC of a set of SKOS annotations is defined by the sum of the IC of each concept $c \in \{ \mu(?c) \mid \mu \in \Omega_{pre} \}$, where $freq_{cond}(c)$ is the frequency of c among all filtered resources and n is the maximum frequency among these resources (Eq. 13).*

$$IC(C_{cond}) = - \sum_{c \in C_{cond}} \log \left(\frac{freq_{cond}(c)}{n} \right) \quad (13)$$

The system carries out the calculations (i.e., determination of similarity values, the ranking of LOD resources) in the same way as in non-filtering mode.

The strengths of the approach will be illustrated in the context of a travel recommendation scenario. Suppose a customer expressed a preference for the Lake Baikal region (e.g., as indicated by a positive rating on a travel community site). With simple on-the-fly retrieval, he

would receive a recommendation list that primarily consists of travel destinations that are located in the proximity to this geographical area. A graph-based filter, on the other hand, facilitates advanced retrieval options. In the given example, it enables the user to obtain suggestions for Southeast Asian travel destinations that have similar features as the Lake Baikal region, even though the resource `dbr:Lake_Baikal` is not directly connected to any destination with the annotation `dbc:Southeast_Asia` in the DBpedia but can only be identified by exploring the surrounding graph structure of the filter attribute. In the given example, the advanced constraint ensures that the generated result set is not empty. A simple attribute-level filter, which is commonly utilized in faceted search systems, would not generate any search hits. By these means, graph-based filter patterns can help users to better explore knowledge graphs thereby overcoming data quality issues in LOD repositories. The corresponding SKOSRec query to this example is shown in Listing 5.

Listing 5: SKOSRec query with an expressive pre-filter condition to retrieve Southeast Asian destinations based on a preference for the Lake Baikal region (Q3)

```

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbc: <http://dbpedia.org/resource/Category:>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

RECOMMEND ?destination TOP 5
PREF dbr:Lake_Baikal
[ WHERE
{ ?destination dbo:country ?country.
  ?country dct:subject ?countrySubject.
  ?countrySubject skos:broader* dbc:Southeast_Asia.
  ?destination rdf:type dbo:Place.
}]

```

Table 3.5 shows the solution to this query. It contains travel destinations and regions of interest that are both similar to the previously liked destination `dbr:Lake_Baikal` and are located in Southeast Asia. Ecoregions around lakes, rivers or mountains from Southeast Asian countries, such as Cambodia, Vietnam or Malaysia are presented to the user.

3.6. Postfiltering & Combinations

The SKOSRec engine cannot only combine similarity calculation and graph pattern matching to pre-filter relevant LOD resources, it can also apply user constraints after similar items have already been de-

Table 3
Result set of Q3

| ?destination |
|--------------------|
| dbr:Tonle_Sap |
| dbr:Mekong |
| dbr:Mount_Kinabalu |
| dbr:Laguna_de_Bay |
| dbr:Lake_Toba |

termined. By this means, recommendations are filtered ex-post to the similarity detection process. This feature enables novel retrieval patterns, in which suggestions are part of a subquery. Definition 14 specifies the notion of SKOSRec recommendations that may potentially undergo postprocessing.

Definition 14 (SKOSRec recommendations). *SKOS-Rec recommendations ($R(Pr)$) are the set of suggestions generated by processing the solution mappings (Ω) obtained from recommendation retrieval. They can be a result of simple on-the-fly retrieval, prefiltering or a result of a combination of these techniques. Pr represents the input profile that contains at least one preference for an item. The cardinality of $R(Pr)$ is the intended number of recommendations (k) that is specified by the user, based on which the engine selects the resources with the highest recommendation scores that are not contained in the profile (Eq. 14).*

$$R(Pr) = \{x \mid x \in \{q \mid \text{score}(Pr, q) \geq \max_{q \in \Omega} \text{score}(Pr, q), q \notin Pr\}\} \quad (14)$$

The approach of subquerying with recommendation results builds on the idea that SKOSRec suggestions $R(Pr)$ can be joined with any graph pattern P_{post} past to the process of similarity calculation (Definition 15). It does not make a difference how the engine obtained these recommendations.

Definition 15 (Postfiltered recommendation mapping). *The postfiltered recommendation mapping is obtained by joining solution mappings generated from SKOS-Rec recommendations with the results of matching a postfilter graph pattern (Ω_{post}) (Eqs. 15 and 16).*

$$\Omega_{postfilter} = \{\mu \mid \mu \in [[P_{post}]]_D, ?x \in \text{dom}(\mu)\} \quad (15)$$

$$\Omega_{post} = \{\mu_{|?x} \mid x \in R(Pr)\} \bowtie \Omega_{postfilter} \quad (16)$$

As the prefilter, P_{post} can be comprised of triple statements that are eligible in the `RecGroupGraphPattern` part of the SKOSRec grammar. As an example scenario for a postfilter recommendation task, consider the following SKOSRec query which obtains suggestions for movies and directors based on a user's preference for the director `dbr:Quentin_Tarantino` (see Listing 6). For the given request, the engine would identify similar directors based on Quentin Tarantino's characteristic features (e.g., geographic origin or awards won) and would determine the movies these similar directors have shot. Thus, users may receive interesting movie recommendations.

Listing 6: SKOSRec query to generate simple post-filtered recommendations in the movie domain (Q4)

```

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?director ?movie
WHERE {
?movie dbo:director ?director .
?director rdf:type yago:Director110014939 .
}
LIMIT 50
RECOMMEND ?director TOP 5
PREF dbr:Quentin_Tarantino

```

Table 5 shows the solution the SKOSRec system returns when Q4 is issued against DBpedia. The ellipses in the displayed tables indicate records that have been excluded for brevity reasons.

However, this approach has its weaknesses. For instance, the engine only considers the metadata descriptions of the director, whereas related movies have to be retrieved anew, upon execution of the postfilter section. Thus, they are not ranked according to similarity in the final output table.

Another weakness is the biased semantics of the query. Usually, when a person favors a movie director, this preference refers to the movies this director has shot and not to the personal features of the person. Two directors might be similar according to certain features such as birth year, nationality or received awards, but the style and genre of the movies they have created

Table 4
Result set of Q4

| ?director | ?movie |
|-------------------|------------------------------------|
| dbr:Kirk_Douglas | dbr:Scalawag_(film) |
| dbr:Kirk_Douglas | dbr:Posse_(1975_film) |
| dbr:Kevin_Costner | dbr:The_Postman_(film) |
| dbr:Kevin_Costner | dbr:Dances_with_Wolves |
| ... | ... |
| dbr:Gene_Kelly | dbr:Hello,_Dolly!_(film) |
| dbr:Gene_Kelly | dbr:Invitation_to_the_Dance_(film) |
| ... | ... |
| dbr:Alan_Alda | dbr:Goodbye,_Farewell_and_Amen |
| dbr:Alan_Alda | dbr:Margaret'_s_Engagement |
| ... | ... |
| dbr:Steve_Buscemi | dbr:Interview_(2007_film) |
| dbr:Steve_Buscemi | dbr:Animal_Factory |

can still be different. A third factor is that recommendation retrieval for movie directors should also take into account that a film artist is likely more relevant, the more movies he/she has shot that are in line with the preferred movies of the user. Hence, the SKOSRec engine enables another postfiltering strategy, which is called aggregation-based retrieval. This strategy can be helpful whenever an entity type is linked to a couple of LOD resources in a dataset, such that similarity scores of related entities (e.g., movies) can be aggregated to an overall recommendation score for a certain item (e.g., for a movie director). Aggregation-based queries have to be treated differently than a regular postfilter request. Apart from excluding sublevel items of the profile from the set of similar resources, the engine should also omit any connections between the preferred superordinate item (a) and similar sublevel entities. Additionally, the user has to specify the variable in his profile (e.g., $?y$) that represents the items within the postfilter graph pattern P_{post} . By this means, aggregation-based recommendations can be generated (Definition 16).

Definition 16 (Resources for aggregation-based retrieval). *The set of resources for aggregation-based retrieval is obtained by retrieving LOD resources that have a connection to a previously generated suggestion thereby excluding all resources that are linked to the specified superordinate item a in the profile (Eq. 17).*

$$R_{agg}(a) = \{\mu(?y) \mid \mu \in \Omega_{post}, ?y \in \text{dom}(\mu), a \notin \mu(?y)\} \quad (17)$$

For each potential aggregation-based suggestion (y), similarity scores of connected entities (x) have to be

considered for the final ranking. In this context, different approaches to aggregation can be applied. In cases when both similar items and aggregation-based recommendations are entities on comparable levels of granularity, then choosing the maximum similarity score might be the best way to represent the relatedness of the resource to the user profile. On the other hand, when postfiltered recommendations are based on similarity scores of their sublevel entities (e.g., as in the movie recommendation example), the scores of related suggestions should be aggregated by the sum or the average of individual scores.

Definition 17 (Aggregation-based ranking score). *The ranking score of an LOD resource y that is connected to a set of recommended items is determined by aggregating scores of connected entities either by the maximum (Eq. 18), the sum (Eq. 19) or the average (Eq. 20) of individual scores.*

$$score_{aggMAX}(y) = \max_{x \in \mu(?x,y)} score(Pr, x) \quad (18)$$

$$score_{aggSUM}(y) = \sum_{x \in \mu(?x,y)} score(Pr, x) \quad (19)$$

$$score_{aggAVG}(y) = \frac{\sum_{x \in \mu(?x,y)} score(Pr, x)}{|x \in \mu(?x,y)|} \quad (20)$$

Aggregation-based requests can be well combined with other retrieval patterns, such as preference queries. In the case of the movie recommendation scenario, a user could state that he enjoyed Quentin Tarantino movies in the past and would like to receive suggestions for directors that shot similar film. Usually, this kind of question can typically only be answered by another human being, e.g., in a personal interaction among friends or in an online forum. The fact that the language facilitates these kinds of queries is one of its key advantages.

Table 5 depicts the results of Q5 as retrieved from DBpedia. When comparing the solutions sets for Q4 and Q5, it becomes clear that they are totally different. While Q5 generated a list of directors, who have mostly created action and thriller movies with a twist (i.e., Tarantino’s unique filmmaking style⁸), Q4

Listing 7: SKOSRec query to generate post-filtered recommendations based on a preference query in the movie domain (Q5)

```

PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX yago: <http://dbpedia.org/class/yago/>

SELECT ?director ?movie
WHERE {
?movie dbo:director ?director .
?director rdf:type yago:Director110014939 .
}
LIMIT 50
AGG dbr:Quentin_Tarantino ?director SUM
RECOMMEND ?movie TOP 100
PREF [ ?prefMovie
WHERE {
?prefMovie dbo:director dbr:Quentin_Tarantino .
} ]

```

Table 5
Result set of Q5

| ?director | ?movie |
|--------------------------|----------------------------|
| dbr:Steven_Spielberg | dbr:Indiana_Jones... |
| dbr:Steven_Spielberg | dbr:Jurassic_Park_(film) |
| ... | ... |
| dbr:Francis_Ford_Coppola | dbr:The_Godfather_Part_II |
| dbr:Francis_Ford_Coppola | dbr:The_Godfather_Part_III |
| ... | ... |
| dbr:Christopher_Nolan | dbr:The_Prestige_(film) |
| dbr:Christopher_Nolan | dbr:The_Dark_Knight |
| ... | ... |
| dbr:Robert_Rodriguez | dbr:Planet_Terror |
| dbr:Robert_Rodriguez | dbr:Machete_(film) |
| ... | ... |
| dbr:Kevin_Smith | dbr:Clerks |
| dbr:Kevin_Smith | dbr:Clerks_II |

provided a completely different output. It shows US American actor-directors, whose movies range from musical films (dbr:Hello,_Dolly!_(film)) to western movies (dbr:Dances_with_Wolves) or drama films (dbr:Interview_(2007_film)). Thus, the directors in the solution table of Q4 (Tab. 3.6) seem rather arbitrary and not exceptionally helpful concerning the initial user request. The evaluation section will have to verify, whether this assumption can be confirmed for a multitude of user requests.

The combined aggregation-based query can also be described as a rollup request since suggestions are based on preferences for sublevel entities. Another example for a roll-up request stems from the application scenario of travel search. This recommendation query derives suggestions for city trip destinations from the

⁸https://en.wikipedia.org/wiki/Quentin_Tarantino

points of interest (POI) a user has visited and liked in another city (e.g., London). Listing 8 shows the corresponding SKOSRec query for this scenario.

Listing 8: SKOSRec query to generate post-filtered recommendations (aggregation-based) for a preference profile containing the city of London (Q6)

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dul:
<http://www.ontologydesignpatterns.org...>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:
<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?city
WHERE {
?poi ?property ?city .
?city rdf:type dbo:City .
?property rdfs:subPropertyOf dul:hasLocation .
}
LIMIT 3
AGG dbr:London ?city SUM
RECOMMEND ?poi TOP 100
PREF
dbr:Oxford_Street
dbr:Notting_Hill
dbr:South_Bank
```

The SKOSRec request contains subquery commands that refer to the user's preferred POIs. It specifies, how similarity scores of the generated suggestions should be aggregated (i.e., sum-based). The retrieval is based on the 100 POIs that are most similar to the ones in the preference section. Additionally, the subquery part refers to the DBpedia resource that represents London (AGG dbr:London ?city) thus indicating that the list of recommendations should not contain any POIs in this city. Upon processing the depicted query over the DBpedia repository, the engine displays the three cities, for whom the highest aggregated scores have been detected (see Listing 6).

Table 6
Result set of Q6

| ?city |
|-------------------------|
| dbr:Oxford |
| dbr:Abingdon-on-Thames |
| dbr:Wallingford,_Oxford |

The motivation for formulating such a query is that when searching for cities, preferences might be better represented through the sights a user has visited elsewhere rather than through a simple on-the-fly request solely containing a preference statement for a city in the user profile section. This assumption was tested in

subsequent web experiments (see Sect. 4).

Besides the above presented roll-up retrieval patterns, the SKOSRec grammar may facilitate the formulation of additional types of advanced recommendation queries. For instance, cross-domain queries are an interesting additional pattern to be explored for personalized retrieval. Listing 9 shows an example request that generates movie suggestions based on the preference for a music band (i.e., dbr:The_Jackson_5).

Listing 9: SKOSRec query to generate cross-domain recommendations (Q7)

```
PREFIX dbr: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?movie ?participated ?musicalAct
WHERE {
?movie rdf:type dbo:Film .
VALUES
{ dbo:basedOn dbo:musicComposer
  dbo:genre dbo:starring }
{ ?musicalAct ?participated ?movie . }
UNION
{ ?movie ?participated ?musicalAct . }
}
LIMIT 10
AGG dbr:The_Jackson_5 ?movie MAX
RECOMMEND ?musicalAct TOP 100
PREF dbr:The_Jackson_5
WHERE {
VALUES ?actType { dbo:MusicalArtist dbo:Band }
?musicalAct rdf:type ?actType .
}
```

The query contains a prefilter condition that triggers retrieval of music artists and bands. Afterward, the scores of the top 100 suggestions are summarized with maximum-based aggregation. Thus, the highest similarity value among the set of music acts determines the ranking score of the film. Also, note because of the aggregation-based retrieval procedure, connections between movies and the LOD resource dbr:The_Jackson_5 are excluded from similarity calculation. The scenario exemplifies, how semantic relationships can help to generate suggestions for a target domain (i.e., movies) that are based on a profile containing items from a different source domain (i.e., music acts). While regular SPARQL queries perform exact matching of triple statements, the SKOSRec syntax facilitates the integration of similarity- and graph-based retrieval to enrich result lists with other potentially relevant items. As a point of reference, consider the SPARQL query for the given cross-domain scenario (Listing 10).

The query omits the similarity calculation part. Thus, it only retrieves movies linked to the resource dbr:The_Jackson_5. By this means, the SPARQL results have a better chance of being related to the initial user profile. On the other hand, empty result lists can occur when the preferred LOD resource does not have any direct connections to

Listing 10: SPARQL query to generate cross-domain recommendations (Q8)

```

PREFIX dbr: <http://dbpedia.org/resource/> .
PREFIX dbo: <http://dbpedia.org/ontology/> .
PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

SELECT ?movie ?musicalAct
WHERE {
?movie rdf:type dbo:Film .
VALUES { dbo:basedOn dbo:musicComposer
dbo:genre dbo:starring }
{ dbr:The_Jackson_5 ?participated ?movie . }
UNION
{ ?movie ?participated dbr:The_Jackson_5 . }
LIMIT 5

```

Table 7
Result set of Q7

| ?movie | ?musicalAct |
|------------------------------|------------------------------|
| dbr:Moonwalker | dbr:Michael_Jackson |
| ... | ... |
| dbr:The_Wiz_(film) | dbr:Ashford_&_Simpson |
| dbr:Garfield_Gets_a_Life | dbr:The_Temptations |
| dbr:Pipe_Dreams_(1976_films) | dbr:Gladys_Knight_&_the_Pips |
| dbr:The_Jacksons:... | dbr:Jermaine_Jackson |

Table 8
Result set of Q8

| ?movie | ?musicalAct |
|----------------------|-------------------|
| dbr:The_Jacksons:... | dbr:The_Jackson_5 |

recommendable items. This assumption is backed up by the result lists of the two queries (Tabs. 7 and 8): The SKOS-Rec query produces a more comprehensive solution than a regular SPARQL query. The evaluation will have to show whether this finding can be confirmed for a sufficient quantity of test cases.

4. Evaluation of the SKOS Recommender

4.1. Experimental Setup

The novel recommendation strategies of the SKOSRec engine were evaluated in the context of web-based experiments for the usage scenarios of travel destination search and multimedia RS. The online approach helped to reach out to more participants with diverse demographic backgrounds, which would not have been possible in a laboratory experiment. Another positive side-effect of the web-based setting was that participants could test and rate recommendations without being directly observed by an experimenter, which might have caused biased results otherwise. Hence, the online approach softened common limitations of laboratory studies [19].

The experiments were carried out in the defined usage scenarios with metadata descriptions from DBpedia. The database containing the local mirror of DBpedia ran on a virtual server. Besides the triple store, a web application was hosted on the same machine. The website ran on an Apache Tomcat 7 application server and was implemented with the help of Java servlet technology.^{9,10}

We conducted the online experiments between April 2016 and January 2017. Upon setting up the web application, a pretest was carried out to control for potential pitfalls, such as misleading navigation. Two test users ran step by step through the web interface and provided their feedback. Because of their insights, we slightly modified the interface (e.g., through rephrasing user instructions) to increase the chances of successful completion.

Subjects were recruited through the clickworker.com platform.¹¹ Participants received a compensation fee of 1.00€ or 1.50€ for a completed survey. 103 participants took part in the study on travel destination search. In the multimedia domains, in total 154 subjects evaluated the SKOSRec engine. The survey template started with a consent form. It informed participants about the context and the procedure of the study. The web form also assured subjects that no deception was involved in survey completion and underlined the participants' right to refrain from the experiment at any time. Additionally, participants were asked to confirm that they were 18 years or older. The web form obtained the subjects' direct consent to take part in the experiment under the provided conditions thereby helping to carry out the study in an ethically sound manner [57]. We utilized a standard consent form commonly applied in social psychology research and adapted it slightly to the different usage scenarios.¹² Figure 2 shows an example consent from the study on music RS.

After having filled in the consent form, participants were navigated to the first part of the experiment. It collected data on subjects' demographics and their consumption and search habits (see Fig. 3).

After the preparation phase, participants worked on the first test case (TC1) in each usage scenario. In this test case, the performance of the SKOSRec engine was assessed in the simple execution mode of on-the-fly retrieval. It was done to gather data on the usefulness of suggestions resulting from a baseline method, with which more advanced recommendation strategies could be compared at a later stage. In cases, where users did not state any items, the application showed an error page. While setting up a profile required some effort on the side of the user, it was a necessary step for the experiment. Since we did not have access to the log files of a live application, profile generation had to be simulated. The process was facilitated by a specifically tailored interface (see

⁹<http://tomcat.apache.org/>

¹⁰<http://www.oracle.com/technetwork/java/javaee/servlet/index.html>

¹¹<https://www.clickworker.de/>

¹²<https://www.socialpsychology.org/consentform.htm>

Recommender Systems
User Studies

Home Contact

Media Recommender Systems - Music

Thank you for participating in our experiment on media recommender systems! Your help is very much appreciated. The study examines the usefulness of recommendation algorithms when searching for music acts. Before taking part in this experiment, please read the following consent form and click on the "I Agree" button at the bottom of the page if you understand the statements and freely consent to participate in the study.

Consent Form

This web-based experiment is designed to understand preferences for different recommender systems' algorithms in the music domain. The study is being conducted by Lisa Wenige (M.Sc.), research assistant at the Chair of Business Information Systems, Friedrich Schiller University Jena, Germany. No deception is involved, and the study involves no more than minimal risk to participants (i.e. the level of risk encountered in daily life).

Participation in the study typically takes 20 minutes and is strictly anonymous. Participants start by providing demographic information and answering a few questions regarding their habits when searching for music acts. Then, they will be asked to evaluate recommendation results in a different music search test cases. Sufficient instructions are provided for each test case.

All responses are treated as confidential, and no case will responses from individual participants be identified. Rather, all data will be published in aggregate form only. Participants should be aware, however, that the experiment is not being run from a "secure" HTTPS server of the kind typically used to handle credit card transactions, so there is a small possibility that responses could be viewed by unauthorized third parties (e.g. computer hackers).

Many individuals from previously conducted similar experiments found it to be at least interesting, if not enjoyable, and no adverse reactions have been reported thus far. Participants from clickworker.com will receive a monetary compensation. Other visitors to this web site are welcome to complete the study, although they will receive no compensation. Participation is entirely voluntary. Participants are free to refuse to answer any question or withdraw from the experiment at any time. The decision about whether to take part in the study will not affect any future interactions of the participants with media content providers or their relations with the Friedrich Schiller University Jena.

Participants who have further questions about this study or their rights, or wish to issue a complaint or concern may contact us via phone or e-mail.

If you are 18 years of age or older, understand the statements above, and freely consent to participate in the study, click on the "I Agree" button to begin the experiment.

I Agree Do Not Agree

Fig. 2. Consent form, page 0 (music domain)

Recommender Systems
User Studies

Home Contact

Media Recommender Systems - Music

Please start by answering a few general questions about yourself and your music search habits.

Demographic Information

Q1. Your age range
under 20

Q2. Your Gender
female

Music

Q3. What is your favorite genre?
Alternative Music

Q4. How many hours do you spend listening to music per day (on average)?
1-2 hours

Music Search

Q5. How do you hear/find out about interesting music acts?
(Multiple responses are possible)

Family and friends
 Media coverage (TV, magazines, newspapers)
 Media store
 Local store
 Online communities (e.g. Last.fm, Blogger)
 Online shops, Streaming websites (e.g. Amazon, Spotify)
 Others

Q6. Please complete the following sentence: Finding interesting music acts is ...
 very easy easy manageable hard very hard

Fig. 3. Demographic section of the web experiment (music domain)

Fig. 4), where subjects could search for items through an autocomplete field that applied AJAX technology.¹³

It enabled users to type in keywords (e.g., the name of a music act or the author of a book), for which matchings were instantaneously displayed without reloading the site. Hence, participants could quickly find and select their items of interest. Whenever a user entered a query through the AJAX interface, the system matched the query keywords against an Apache Solr/Lucene search index.¹⁴ The index was built with item metadata, which had been extracted from DBpedia prior to setting up the website. The index comprised the names and abstracts of items. Whenever available, information on

Recommender Systems
User Studies

Home Contact

Q7. Test Case 1

This section explores the ability of automatically generated recommendations for music acts. To get a better idea of your preferences, please provide 3 music acts whose music you particularly like. For the recommendation algorithms to work, we kindly ask you to only select music acts from the auto-complete lists that appear as soon as you type.

Favorite music acts

Music act 1 The Police ID: The_Police

Music act 2 Sting (musician) ID: St

Music act 3 Bono

Bono
Paul David Hewson (born 10 May 1960), known by his stage name Bono, is an Irish singer, musician, venture capitalist and humanitarian best known for being the main vocalist of the Dublin-based rock band U2. Bono was born and raised in Dublin, Ireland, and attended Mount Temple Comprehensive School where he met his future wife, Alison Stewart, and the future members of U2. Bono writes almost all U2 lyrics, often using profanity, social and religious themes.

Sarah De Bono
Sarah De Bono is an Australian singer-songwriter and pianist, born and raised in Melbourne. She participated on the first season of The Voice (Australia), coming in fourth place shortly after she signed a record deal with Universal Music Australia. On 24 June 2011, De Bono scored her first top 10 hit with "Beautiful", which peaked at number five on the ARIA Singles Chart and was certified gold.

Elphig Blue Albman
Elphig Blue Albman (born July 10, 1976) is the son of Cher and her second husband Gregg Albman and half brother of Cher Bono, Cedric Albman, Michael Albman, Layla Albman and

Fig. 4. User profile generation, TC1 (music domain)

thumbnail pictures were saved in the index to be ready for display in the AJAX interface. In case, the Apache Solr/Lucene search engine found index keywords that matched the user query, metadata information for these items were immediately shown on the screen. This information was displayed to help participants make an informed decision on whether the AJAX result list contained suitable items for profile generation.

Aside from the AJAX feature, the appeal of the interface was increased by a progress bar at the top of the webpage (see Fig. 4). Progress bars are commonly utilized tools for web surveys. In conventional paper-based studies, subjects usually see, how far ahead they are, while in a web survey they cannot check their progress immediately. This is especially true for studies that apply a screen-by-screen navigation, which was the chosen approach for the conducted web experiments. Progress bars prevent users from tiring of the questionnaire and withdrawing from the study altogether by showing, how close participants are to complete the experiment [14].

Upon profile generation, the SKOSRec engine calculated on-the-fly recommendations, which were shown to users in a separate screen. Suggestions were displayed with a sufficient amount of information e.g., thumbnails, labels or a short description to help participants assess the quality of the recommendation. In addition to this information, users could also access the respective Wikipedia article by following the hyperlink on the label. Item-level assessments of recommendation results were partly carried out with the help of relevance sliders, which are often applied in evaluations of IR systems [48, 51]. They let participants state a degree of relevance for each item, instead of asking them to make a binary decision on item relevance. Scores were handled as points on a 0 to 100 scale of the relevance slider (outer left side: lowest relevance, outer right side: highest relevance, see Fig. 5).

This approach was taken, because it conveniently enabled calculation of a fine-grained precision score that helped to assess the accuracy of the results. The score, called mean

¹³<http://api.jquery.com/jquery.ajax/>

¹⁴<http://lucene.apache.org/solr/>

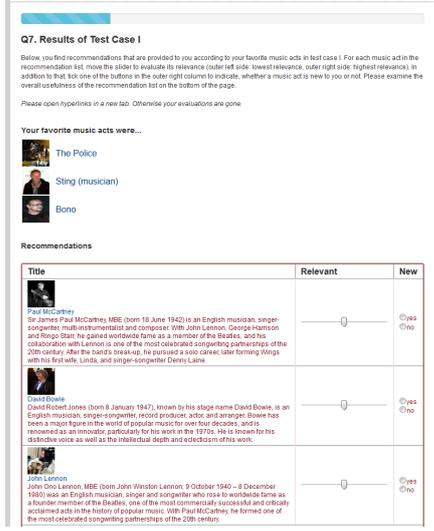


Fig. 5. Recommendations resulting from on-the-fly recommendation retrieval, TC1 (music domain)

relevance (mrs), was measured with Equation 21.

$$mrs = \sum_{i=1}^k \frac{score_i}{k} \quad (21)$$

In addition to the precision score, the web application also determined the size of the recommendation list produced by the engine as an indicator for recall. However, recommendation list size can only be utilized for this purpose when mrs scores reach a reasonable level.

While accurate recommendations are useful, because they build trust in a system [54], they only capture a narrow aspect of performance [36]. For instance, an online retailer can tremendously profit from an RS that provides both relevant and unobvious recommendations, since it enables users to explore the product catalog more thoroughly. On the other hand, in the case of an RS only suggesting popular products, it is likely that the user already knows them [3, 23]. A pure accuracy-based evaluation disregards these aspects, which is why other performance indicators, such as novelty, were measured as well. Thus, the evaluation interface also contained a section, where subjects had to tick a radio button that indicated, whether an item was new. In the backend of the web application, novelty was measured as the ratio of relevant items not familiar to the user ($\#nov$) among the total number of relevant items ($\#tp$) (Eq. 22).

$$nv = \frac{|\#nov|}{|\#tp|} \quad (22)$$

An item was considered relevant, when it achieved an mrs score of 50 or higher. In addition to novelty and diversity, the web application also tracked the diversity of recommendation lists since a topically diversified result set has been found to increase overall user satisfaction [31, 62]. In the RS literature, the most widely explored diversification metrics are the ones that measure the similarity between each item pair in the recommendation list. The more the items are alike; the less diverse is the result set [19]. The study series of this paper applied the metric by Castells et al. [8]. It determines item-to-item similarity values with the cosine similarity measure. Scores are then converted to distance values (Eq. 23) and averaged for the recommendation list (Eq. 24)

$$d(i_l, i_m) = 1 - \cos(i_l, i_m) \quad (23)$$

$$divC = \frac{2}{k(k-1)} \sum_{l < m} d(i_l, i_m) \quad (24)$$

The content-based diversity scores ($divC$) were calculated in the backend of the web application and saved in a log file for each generated recommendation list of the SKOSRec engine during runtime execution. In addition to the measurement of $divC$ scores, we asked participants about their opinions on recommendation list diversity directly. Following the RS evaluation framework by Pu et al. the following two Likert items were posed for this purpose [44]:

- The recommendations of list [...] are diverse.
- The recommendations of list [...] are similar to each other.

Besides the above listed algorithmic performance metrics, the authors gathered assessments from users regarding the overall usefulness of the suggestions. This approach is also in line with the empirically verified evaluation framework by Pu et al., which proposes to measure the psychometric construct of *Perceived Usefulness* to get a comprehensive understanding of user opinions on recommendation results. Thus, participants were instructed to state their agreement to positive statements regarding the relevance of the suggestions [44]. The statements were adapted to the specificities of each usage scenario. Figure 6 shows an example collection of Likert questions for the domain of music RS.

After participants had provided the required information, their answers were saved in a log file and they were navigated to test case 2 (TC2), which evaluated the SKOSRec engine's performance for constraint-based recommendations. We tried to find out if the engine was able to improve user satisfaction through filter conditions. Even though IR systems already successfully apply facet filters on result sets, the experiments still had to prove whether LOD-enabled constraints have the same effect on the *Perceived Usefulness* of

| Statement | Agreement |
|---|---|
| The recommendations better fit my preferences than the suggestions I would receive from other sources (see Q5). | <input type="radio"/> strongly agree <input type="radio"/> agree <input type="radio"/> neutral <input type="radio"/> disagree <input type="radio"/> strongly disagree |
| I feel supported to find interesting music acts with the help of the recommender. | <input type="radio"/> strongly agree <input type="radio"/> agree <input type="radio"/> neutral <input type="radio"/> disagree <input type="radio"/> strongly disagree |
| The recommended items influence my selection of music acts. | <input type="radio"/> strongly agree <input type="radio"/> agree <input type="radio"/> neutral <input type="radio"/> disagree <input type="radio"/> strongly disagree |
| The recommended items effectively helped me to find what I like. | <input type="radio"/> strongly agree <input type="radio"/> agree <input type="radio"/> neutral <input type="radio"/> disagree <input type="radio"/> strongly disagree |
| I feel supported in selecting the items to buy with the help of the recommender. | <input type="radio"/> strongly agree <input type="radio"/> agree <input type="radio"/> neutral <input type="radio"/> disagree <input type="radio"/> strongly disagree |

Fig. 6. Evaluation screen for recommendations resulting from on-the-fly recommendation retrieval, TC1 (music domain)

recommendations. For the comparisons of non-filtered and filtered suggestions, we applied a within-subjects design, since between-subjects experiments would have required a larger sample of user assessments [30]. Additionally, within-subjects designs have the advantage that variations between individuals are not an issue. Beel et al. have shown that even small variations in an RS test setting could cause substantial differences in performance outcomes [6]. To keep the extent of variations even lower, comparisons between non-filter and constraint-based recommendations, were based on the same profile for each user in TC2. Therefore, the user profile from TC1 served as a starting point for filtered retrieval. The web application showed the recently generated profiles to users and asked them to provide an additional filter condition. Figure 7 depicts an example web form for a constraint-based recommendation request in the music domain. It comprises the user profile and an additional text field, where users stated their constraint.

Q8. Test Case II

In this test case you can filter recommendations according to your interests. You see the music acts you have stated as your preferences in the last section (Q7). Now, you are able to refine these preferences with a condition. Please choose either a subject (e.g. Folk songs) or a genre (e.g. Soul music) that fits your interests. For the recommendation algorithms to work, we kindly ask you to only select conditions that appear in the auto-complete list of the "Filter" text.

Favorite music acts

- The Police
- Sting (musician)
- Bono

Filter

Subject: British songwriters

Submit

British songwriters
British singer-songwriters
Songwriters
English songwriters
Argentine songwriters
Jazz songwriters
Japanese songwriters
French songwriters

Fig. 7. Web form for a constraint-based recommendation request (music domain)

Afterward, the engine applied this filter condition on the set of potential recommendation results before similarity calculation started. For each usage scenario, different types of constraints were available. Regarding suitable filter options, the author sought to achieve a balance between assisting users in finding filter conditions that would adequately represent their information needs, while keeping the variability among users as low as possible. Therefore, the specificities of each scenario and the availability of the respective data sources were determined. For instance, for the multimedia domains (movie, music, and books) it was assumed that users would like to filter recommendations according to the specific genre of the item. Fortunately, the DBpedia dataset contains this kind of information for the music domain. In this domain, the genre can be retrieved through the properties `dbo:genre`. However, in the movie and book domains, DBpedia does not provide sufficient information. Many LOD resources, which represent books, were not assigned a genre. In the movie domain, a genre property is missing entirely, and genre-specific information can often only be found in the subject categories describing the movie. Hence, in contrast to the experiment on music recommendations, participants in the domains of books and films did not filter their recommendation lists by genre. On the other hand, the property `dc:subject` was applied as recommendation filter in each multimedia domain. Since the subject property often links to many informative features of multimedia items (e.g., release period, geographic information or content information), the author decided that this feature represents a powerful filter dimension.

In the experiment on travel destination search, the web interface offered two filters: a subject and a location-specific filter. The subject filter enabled users to state their preferences regarding the characteristics of a location (e.g., being a nature reserve). The location-specific filter, on the other hand, gave participants the option to specify, where the desired destination should be located.¹⁵

The second fundamental research question of TC2 concerned the question whether expressive constraints can boost recommendation quality even further. Hence, apart from executing a constraint-based workflow with a simple filter (i.e., a direct attribute of an item), the SKOSRec engine generated suggestions with the help of an expressive filter as well. This second procedure, while still applying the same user constraint as the simple filter, utilized an expressive graph pattern to include additional LOD resources in the result set. For instance, in case a subject filter was selected, result set expansion was facilitated by exploring the wider semantic space of the DBpedia category graph through a property path declaration in the graph pattern (e.g., `skos:broader[, 2]`). In the travel experiment, an additional expressive filter retrieved place-related information (`dul:hasLocation`) for location-specific subprop-

¹⁵<http://dbpedia.org>

erty relations that are not materialized in DBpedia. Hence, the expressive graph pattern expanded the filter, such that additional properties were matched (e.g., `dbo:country`, `dbo:city` or `dbo:state`), while still applying the specified user constraint on the set of LOD resources. Upon execution of constraint-based recommendation requests, users received two recommendation lists (resulting from simple and expressive filtering accordingly) in a separate evaluation screen. Participants evaluated result sets in the same manner as in TC1. For each result set, the screen contained an additional Likert item asking subjects to give their agreement to the statement: „The filter has improved the recommendation results of list...“. The sequence of suggestions resulting from the different methods was randomly assigned in each user session to avoid order effects.

After study subjects had completed TC2, they were navigated to test case 3 (TC3). In the TC3 section of the travel experiment, participants could choose between three similar rollup query patterns. The respective templates were formulated in reference to Q6 from Section 3. Users could obtain recommendations for travel destinations based on POIs, they had visited during the stay in another destination (i.e., a city, region or country). Upon entity type selection, the SKOSRec engine generated appropriate suggestions. The three query options were provided to avoid that participants terminated the study early, in case they were not able to state a preferred location for a certain entity type. Figure 8 shows the web form from the travel experiment that facilitated the formulation of advanced rollup requests in TC3.

Fig. 8. Travel - User profile generation, TC3

When users had selected an entity type, stated their favorite travel destination and three belonging POIs, the application generated a SKOSRec query from these parameters. A simple on-the-fly request was sent to the engine as well to retrieve baseline recommendations, with which the suggestions resulting from the advanced request were compared at a later stage of the experiment. The simple query only contained the user's favorite travel destination and the selected entity type option thus omitting information on the POIs. During the travel experiment, the engine processed both the simple and the advanced query and produced two recommendation lists. As in the previous parts of the experiment, participants assessed the quality of the result sets, which appeared in random order on the screen.

The multimedia experiments also contained a section on rollup retrieval (TC3). As the advanced travel queries, the multimedia requests aggregated similarity scores of sublevel entities through summation, which the engine joined with a postfilter. In addition to these features, the pattern contained a preference query part. This section identified a set of preferred items through a single user statement. Participants either specified their favorite director/actor (music domain), music act (music domain) or author (book domain) and received recommendations based on the features of the works created by the artist (see Q5, Sect. 3). The SKOSRec engine generated recommendations based on the creative works of the favored artist. Participants also received suggestions from a simple on-the-fly query, which computed results according to the artist's characteristics. As in the travel experiment, recommendations from the advanced rollup request were compared with the on-the-fly query without letting participants know which approach they were assessing and through random assignment of result set positions.

After participants of the multimedia experiments had evaluated the results of TC3, they were guided to test case 4 (TC4), which evaluated cross-domain queries. The experiment on travel RS ended with TC3 because no suitable graph patterns could be identified to facilitate these kinds of suggestions for the travel usage scenario. In the multimedia domain, however, the data from DBpedia was sufficient for this retrieval task. The entity type of the study (i.e., movie, music act or book) defined the source domain based on which the engine generated suggestions for items from another multimedia target domain (see Q7, Sect. 3). It was not possible to generate cross-domain recommendations for every potential entity type pair in the multimedia scenarios. For instance, no suitable graph patterns were identified to generate suggestions for music items based on preferences for books. However, for the remaining multimedia entity type pairs (i.e., movie/book, movie/music act), enough data sources existed to facilitate cross-domain retrieval. Participants formulated the corresponding queries by stating their favorite item from the source domain. Figure 9 depicts the *cross-domain* web form of the music experiment.

Fig. 9. Music - User profile generation, TC4 (page 9)

The web form assisted subjects in formulating a request, which then obtained movie suggestions based on the favorite music act of the user. Parameters from users were entered in the placeholders of the query pattern in the backend of the application. We formulated the query templates in such a way that the query engine would match any properties or

graph patterns that could potentially connect two items from the specified target and source domains. For instance, such matchings can occur when a user states his favorite movie that is written by a particular author. In case the same author has also written some books, they might be of interest to the user as well. Another example stems from the movie domain. Suppose in his/her profile; a consumer has declared a preference for a movie that links to the corresponding soundtrack. It might well be the case that the user likes the soundtrack and the music acts that were involved in creating it just as much as he/she likes the movie. The practice to distinguish homonymous LOD resources through the property `dbo:wikiPageDisambiguates` is another interesting linking pattern from DBpedia that was exploited for cross-domain requests in TC4 since it connects similar entities (i.e., the book edition of a certain movie).

Cross-domain SPARQL queries, that were similar to Q8 from Section 3, were posed as baseline requests in this test case. While the SPARQL query only matched cross-domain relations for a single LOD resource, the SKOSRec request expanded the search space by considering links from more than one item. As in the previous test cases, TC4 ended with an evaluation screen where users assessed the SPARQL-based as well as the SKOSRec suggestions in random order without knowing which recommendation list belonged to which method. Table 9 gives an overview of the test cases that were applied in the experiments in each usage scenario.

4.2. Results

The gathered samples from the experiments contained answers from both genders with a slight overrepresentation of male participants (53.7%). The prevailing number of study subjects was between 20 and 40 years of age (65.8%). The evaluation of the demographics section also revealed that the majority of participants already perceives the search in their domain of interest as either „manageable“ (38.5%) or even as „easy“ or „very easy“ (44.0%).

Prior to analyzing the performance results of the novel retrieval approaches of the SKOSRec engine, it was checked whether the respective Likert items for *Perceived Usefulness* and user-based diversity (*divU*) reached acceptable reliability levels. For this purpose Cronbach's α values were determined. Table 10 shows the outcome of this analysis for the *Perceived Usefulness* construct. Given the acceptable values throughout the domains, (Cronbach's $\alpha > 0.7$), it could be treated as an interval-scaled variable in the statistical tests.

Besides usefulness, user-based diversity scores were also measured by multiple Likert items. However, Cronbach's α values for this concept did not reach acceptable levels. In order to avoid that valuable user assessments remained unused, agreements to the statement „*The recommendations of list [...] are diverse*“ were taken into account as an ordinal variable (*divU*).

Based on these results, subsequent tests could be carried out. The first part of the analysis concerned the performance of

the on-the-fly retrieval method (baseline) of the SKOSRec engine. The outcome of this analysis confirms that the on-the-fly approach represents a viable recommendation strategy. Table 11 shows the median (M)¹⁶, standard deviation (SD) and number of participants (N) for each performance dimension and metric. It can be seen that users assessed the engine's performance in almost each quality dimension to be above mediocrity. Despite the positive user evaluations with regard to *Perceived Usefulness*, related quality aspects, such as accuracy, novelty and diversity received good assessments as well. For instance, in each domain mean relevance scores (*mrs*) were higher than 50 score points. It means that participants were often convinced of the relevance of the results. Given these positive user assessments with regard to item relevance, it seems to be justified to consider the mean of result list *sizes* as an indicator of recall. The engine was able to generate the required number of recommendations in almost each test case (Tab. 11). The on-the-fly retrieval approach was also tested with regard to the novelty dimension. Table 11 shows the mean novelty scores (*nv*) for the baseline recommendation approach. Novelty scores indicate that the majority of items in a result list were deemed to be new in the travel and book RS, whereas in the experiments on movie and music RS only one third of the recommendation list was unfamiliar to users. In contrast to that, content-based diversity scores (*divC*) were rather high in each domain. Hence, it can be concluded that items in a result set usually have a low rate of matching SKOS annotations. In addition to content-based scores, user assessments of recommendation list diversity (*divU*) were also taken into account. The median level of agreement to the diversity-related Likert item stating that recommendations are diverse lied at 3 („neutral“) in the multimedia domains and at 4 („agree“) in the travel domain. In summary, the baseline approach achieved fairly good performance results in TC1. The findings demonstrate that LOD-enabled recommendations can potentially enhance a user's search experience. They also allow for subsequent tests of advanced retrieval techniques, which can be compared to this standard method.

In the analysis of performance results from TC2 (constraint-based recommendations), particular attention was paid to participants' agreements with the statement: „The filter has improved the recommendation results of the list“. Figure 10 shows the distribution of domain-wise agreement ratios. On average, they were fairly high. Throughout the domains, the vast majority of participants either selected 5 („strongly agree“), 4 („agree“), or 3 („neutral“). Hence, a filter often had a positive impact on results.

The author gathered responses to the question on improvement for recommendation lists resulting from both the simple and the expressive filtering approaches. Hence, they give clues about the general performance of *constraint-based re-*

¹⁶In case of the user-based diversity score (*divU*), M denotes the median, because of the ordinal scale of the variable.

Table 9
Test Cases in the web experiments

| Test Case | Evaluation Purpose | Travel | Movie | Music | Book |
|-----------|---------------------------------------|--------|-------|-------|------|
| TC1 | On-the-fly Recommendations (Baseline) | ✓ | ✓ | ✓ | ✓ |
| TC2 | Constraint-based Recommendations | ✓ | ✓ | ✓ | ✓ |
| TC3 | Advanced Queries (Rollup) | ✓ | ✓ | ✓ | ✓ |
| TC4 | Advanced Queries (Cross-Domain) | x | ✓ | ✓ | ✓ |

Table 10
Cronbach's α values for the concept of *Perceived Usefulness*

| Domain | Cronbach α |
|--------|-------------------|
| Travel | 0.7002 |
| Movie | 0.8579 |
| Music | 0.8515 |
| Book | 0.8949 |

Table 11
Performance results of regular on-the-fly retrieval

| Dimension (Metric) | Domain | Results | | |
|---------------------------|--------|---------|-------|-----|
| | | M | SD | N |
| Accuracy (<i>size</i>) | Travel | 10.00 | 0.00 | 103 |
| | Movie | 10.00 | 0.00 | 50 |
| | Music | 10.00 | 0.00 | 53 |
| | Book | 9.80 | 1.40 | 51 |
| Accuracy (<i>mrs</i>) | Travel | 56.49 | 0.00 | 103 |
| | Movie | 62.25 | 19.15 | 50 |
| | Music | 55.70 | 0.10 | 53 |
| | Book | 57.47 | 15.55 | 50 |
| Novelty (<i>nv</i>) | Travel | 0.60 | 0.34 | 101 |
| | Movie | 0.36 | 0.31 | 50 |
| | Music | 0.36 | 0.39 | 52 |
| | Book | 0.61 | 0.29 | 47 |
| Diversity (<i>divU</i>) | Travel | 4 | - | 102 |
| | Movie | 3 | - | 50 |
| | Music | 3 | - | 53 |
| | Book | 3 | - | 50 |
| Diversity (<i>divC</i>) | Travel | 0.80 | 0.16 | 103 |
| | Movie | 0.87 | 0.12 | 50 |
| | Music | 0.86 | 0.10 | 50 |
| | Book | 0.92 | 0.08 | 50 |
| Usefulness | Travel | 3.34 | 0.85 | 103 |
| | Movie | 3.55 | 0.73 | 50 |
| | Music | 3.18 | 0.85 | 53 |
| | Book | 3.43 | 0.81 | 50 |

trieval. However, since this section of the study revealed the experimental condition to participants (i.e., application of a filter), answers to the Likert statement have to be interpreted cautiously. Results may be slightly biased as participants were able to guess the underlying agenda. However, these

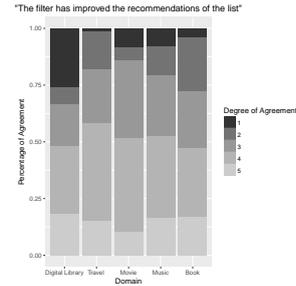


Fig. 10. Participants' agreement with the statement that the filter has improved the result list (domain-wise)

limitations do not exist for comparisons between simple and expressive filters. Since the web application randomly assigned the list order, participants did not know which approach they were assessing. This enables us to take a closer look at the differences between the two filtering approaches. It was assumed that advanced filters improve recommendation quality, as they may identify more relevant resources thereby overcoming problems of data quality or data sparsity in LOD repositories. Table 12 lists response rates of the two methods. The rate measures the ratio of non-zero result sets among all result sets. Throughout the domains, recommendation lists resulting from expressive filtering achieved higher response rates.

Table 12
Response Rates for simple vs. expressive constraint-based retrieval

| Domain | Constr.-based (Regular) | Constr.-based (Expanded) | N |
|--------|-------------------------|--------------------------|-----|
| Travel | 23% | 57% | 103 |
| Movie | 86% | 88% | 50 |
| Music | 72% | 75% | 53 |
| Book | 73% | 76% | 51 |

In addition to response rates, user assessments and scores in the defined quality dimensions were compared. Table 13 depicts the outcome of this analysis. Significantly higher scores are marked in bold figures. In case statistical tests identified no meaningful variations, the results of the better performing approach are underlined. A-levels were adjusted with the false discovery rate (FDR) in order to decrease the probability of making a type I error due to multiple testing.

Table 13
Performance results of constraint-based recommendation retrieval

| Dimension (Metric) | Domain | Simple | | Expressive | | N |
|---------------------------|--------|--------------|-------|----------------|-------------|-----|
| | | M | SD | M | SD | |
| Accuracy (<i>size</i>) | Travel | 1.17 | 2.70 | 4.52*** | 4.62 | 103 |
| | Movie | 6.72 | 4.08 | <u>7.20</u> | 3.92 | 50 |
| | Music | 5.96 | 4.64 | <u>6.36</u> | 4.51 | 53 |
| | Book | 4.59 | 4.50 | <u>5.00</u> | 4.51 | 51 |
| Accuracy (<i>mrs</i>) | Travel | <u>66.73</u> | 20.51 | 65.34 | 22.36 | 23 |
| | Movie | <u>60.95</u> | 22.98 | 60.29 | 22.79 | 43 |
| | Music | <u>63.77</u> | 16.05 | 62.96 | 16.44 | 34 |
| | Book | 56.17 | 18.18 | <u>56.81</u> | 16.81 | 37 |
| Novelty (<i>nv</i>) | Travel | 0.48 | 0.41 | <u>0.61</u> | 0.49 | 21 |
| | Movie | 0.44 | 0.36 | <u>0.55</u> | 0.80 | 42 |
| | Music | <u>0.55</u> | 0.39 | 0.52 | 0.37 | 38 |
| | Book | <u>0.77</u> | 0.34 | 0.76 | 0.33 | 33 |
| Diversity (<i>divU</i>) | Travel | 3 | - | <u>3.5</u> | - | 24 |
| | Movie | 4 | - | 4 | - | 43 |
| | Music | 4 | - | 4 | - | 38 |
| | Book | 3 | - | 3 | - | 37 |
| Diversity (<i>divC</i>) | Travel | 0.68 | 0.18 | <u>0.72</u> | 0.21 | 19 |
| | Movie | 0.70 | 0.24 | <u>0.73</u> | 0.23 | 41 |
| | Music | 0.86 | 0.13 | <u>0.87</u> | 0.12 | 34 |
| | Book | 0.84 | 0.24 | <u>0.95</u> | 0.63 | 29 |
| Usefulness | Travel | 3.33 | 0.93 | <u>3.50</u> | 0.77 | 24 |
| | Movie | <u>3.32</u> | 0.85 | 3.31 | 0.86 | 43 |
| | Music | 3.55 | 0.81 | <u>3.64</u> | 0.78 | 38 |
| | Book | 3.39 | 0.83 | <u>3.62</u> | 1.21 | 37 |

According to mean scores, expressive requests generated more comprehensive recommendation lists (*size*). This finding is in line with the increased response rates shown in Table 12. However, subsequently conducted t-tests confirmed significant differences only for the travel experiment ($t(102) = -7.89, p < 0.001$). In contrast, precision scores were higher in most of the domains (travel, movie, music), when the SKOSRec engine processed a simple constraint-based query. However, statistical tests did not confirm any systematic differences. The author applied Wilcoxon tests for *prec* scores and the remaining performance metrics, because of the small sample sizes resulting from the low response rates of the simple constraint-based approach.

Regarding novelty, none of the approaches was superior. In the diversity dimension, expressive filters generated suggestions that were at least as topically diversified as the results from simple *constraint-based retrieval*. Even though no significant differences were identified between the two approaches, increased mean scores (*divU* in the travel domain and *divC* in each domain) indicate a slight superiority of expanded filter requests in this quality dimension. The same applies to usefulness scores (mean values were higher in 3 out

of 4 domains, when expressive filtering was applied). However, these statements are speculative at best, because statistical tests were not significant.

In summary, it can be concluded that expressive filters improve recall, potentially leading to a slight loss in precision scores (see Tab. 13). It may also be the case that expanded user constraints diversify as well as increase the usefulness of recommendation lists. However, these claims are unverified. In contrast, it is safe to say that expressive query patterns generate results of at least the same quality as simple patterns while increasing the number of relevant suggestions. The few differences may be explained by the high Jaccard scores (JI) (see Tab. 14) of recommendation lists. Throughout the domains, result sets were much alike. Given these similarities and the increases in list *sizes* for expressive filters, it is supposed that an expanded constraint produces an enhanced version of the recommendation list resulting from simple filtering. Therefore, expanded filters should be the default setting in a constraint-based retrieval context.

In TC3 (i.e., evaluation of roll-up query patterns), the SKOSRec engine almost always generated non-empty recommendation lists (see Tab. 15). In the travel and the book

Table 14
Jaccard Indices

| Domain | Mean Jaccard Index (JI) | SD | N |
|--------|-------------------------|------|----|
| Travel | 0.57 | 0.47 | 25 |
| Movie | 0.71 | 0.38 | 43 |
| Music | 0.82 | 0.35 | 37 |
| Book | 0.92 | 0.26 | 35 |

Table 15
Response Rates

| Domain | Regular | Aggr.-based | N |
|--------|---------|-------------|-----|
| Travel | 99% | 100% | 103 |
| Movie | 96% | 92% | 50 |
| Music | 100% | 100% | 53 |
| Book | 96% | 100% | 51 |

experiment, the response rate was higher for aggregation-based queries, whereas in the movie domain, the engine more often provided recommendations for regular requests. However, these differences are only marginal and do not indicate a clear superiority of one method.

Figure 11 depicts participants' general agreement to Likert statements concerning the *Perceived Usefulness* of the approaches in TC3. The diagram shows similar results for regular as well as advanced requests. On average, satisfaction scores reached levels, which lay slightly above neutral agreement. A subsequent t-test confirmed no significant differences between the two methods. In fact, the approach with the highest average score of *Perceived Usefulness* was different in each experiment. While mean values were higher for regular requests in the studies on travel and music RS, advanced queries produced higher scores in the movie and book experiments.

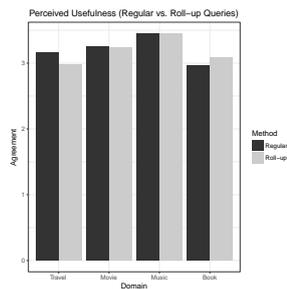


Fig. 11. Participant's overall satisfaction with regular and roll-up recommendation retrieval

Content-based diversity (*divC*) was the only metric with systematic differences. Here, scores went up for advanced retrieval patterns in the travel ($t(98) = -5.50, p < 0.001$), the music ($t(52) = -4.51, p < 0.001$) and the book domain ($t(47) = -4.90, p < 0.001$) (see Table 16). This finding is not surprising, given that the aggregation-based approach

explores the wider semantic network of each item in the profile. In spite of significantly improved content-based diversity scores (*divC*).

Hence, it can be assumed that both approaches produce recommendation lists of similar quality and can provide benefits to potential consumers. It depends on the underlying information need, the respective domain and the available data in the LOD repository, whether or not a particular approach is better suited for a given retrieval task, since neither approach outperformed the other one. This conclusion is especially impressive, given the mean Jaccard scores for recommendation lists resulting from regular and advanced retrieval (see Tab. 17).

The mean score never exceeded 0.1 throughout the domains. It indicates a low concordance between result sets. It is remarkable that participants perceived the recommendations as equally good, given the high dissimilarity of the lists. Thus, advanced queries have an added-value, as they provide additional interesting results. They can help to explore LOD repositories in different ways than a regular retrieval approach. Therefore, it is worthwhile to offer aggregation-based query patterns as an alternative retrieval strategy, when the regular approach has not produced any helpful recommendations.

In addition to TC3, user assessments for TC4 (cross-domain recommendations) were also evaluated. However, since the author conducted experiments for TC4 only in the multimedia domains, the samples were considerably smaller. Nevertheless, it is assumed that the amount of completed user sessions is only just enough to conduct further statistical analyses. Table 18 shows the response rates for the two retrieval methods. Cross-domain patterns had a higher success rate of generating non-zero result sets throughout the domains. Participants almost always received a recommendation for this query type, whereas in case a regular SPARQL query was executed, often they did not receive a single suggestion at all.

Figure 12 gives an overview of participants' general satisfaction with cross-domain recommendation results as opposed to suggestions resulting from a regular SPARQL query. The diagram indicates a tendency of participants to prefer recommendations from the SPARQL query. However, a Wilcoxon test did not confirm this assumption.

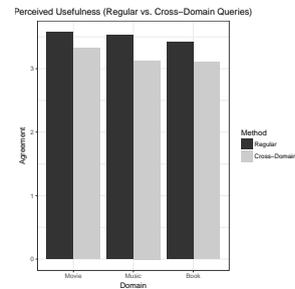


Fig. 12. Participant's overall satisfaction with regular and cross-domain recommendation retrieval

Table 16
Performance results of regular and roll-up recommendation retrieval

| Dimension (Metric) | Domain | Regular | | Aggr.-based | | N |
|---------------------------|--------|--------------|-------|----------------|-------------|-----|
| | | M | SD | M | SD | |
| Accuracy (<i>size</i>) | Travel | <u>9.90</u> | 0.99 | 9.42 | 2.07 | 103 |
| | Movie | <u>2.88</u> | 0.59 | 2.76 | 0.82 | 50 |
| | Music | 10.00 | 0.00 | 10.00 | 0.00 | 53 |
| | Book | 2.88 | 0.59 | <u>3.00</u> | 0.00 | 51 |
| Accuracy (<i>mrs</i>) | Travel | <u>48.21</u> | 24.98 | 44.93 | 25.59 | 102 |
| | Movie | 54.58 | 22.88 | <u>56.69</u> | 25.42 | 44 |
| | Music | 53.95 | 18.31 | <u>58.24</u> | 16.19 | 52 |
| | Book | <u>51.53</u> | 23.90 | 49.25 | 22.30 | 49 |
| Novelty (<i>nv</i>) | Travel | <u>0.43</u> | 0.39 | 0.38 | 0.40 | 98 |
| | Movie | <u>0.35</u> | 0.38 | 0.28 | 0.41 | 41 |
| | Music | <u>0.49</u> | 0.36 | 0.46 | 0.39 | 50 |
| | Book | 0.61 | 0.42 | <u>0.62</u> | 0.44 | 47 |
| Diversity (<i>divU</i>) | Travel | <u>3.5</u> | - | 3 | - | 100 |
| | Movie | <u>4</u> | - | 3 | - | 43 |
| | Music | <u>4</u> | - | 3 | - | 50 |
| | Book | 3 | - | 3 | - | 48 |
| Diversity (<i>divC</i>) | Travel | 0.79 | 0.17 | 0.89*** | 0.13 | 99 |
| | Movie | <u>0.75</u> | 0.20 | 0.66 | 0.27 | 44 |
| | Music | 0.83 | 0.17 | 0.93*** | 0.07 | 53 |
| | Book | 0.74 | 0.27 | 0.93*** | 0.06 | 48 |
| Usefulness | Travel | <u>3.36</u> | 0.77 | 3.22 | 0.74 | 101 |
| | Movie | 3.20 | 0.92 | <u>3.26</u> | 0.94 | 44 |
| | Music | <u>3.45</u> | 0.84 | 3.44 | 0.85 | 51 |
| | Book | 2.95 | 0.87 | <u>3.03</u> | 0.96 | 48 |

Table 17
Jaccard Indices (TC3)

| Domain | Mean Jaccard Index (JI) | SD | N |
|--------|-------------------------|------|-----|
| Travel | 0.07 | 0.11 | 101 |
| Movie | 0.03 | 0.07 | 44 |
| Music | 0.02 | 0.05 | 53 |
| Book | 0.02 | 0.06 | 50 |

Table 18
Response rates for TC4

| Domain | Regular (SPARQL) | Cross-Domain | N |
|--------|------------------|--------------|----|
| Movie | 32% | 96% | 50 |
| Music | 62% | 98% | 53 |
| Book | 53% | 100% | 51 |

The author applied non-parametric tests for the performance metrics *mrs*, *nv*, *divU*, *divC*, and *Perceived Usefulness*, due to the low response rates of the SPARQL-based approach, which led to a low number of comparable data points accordingly. Overall the FDR-adjusted pairwise Wilcoxon tests detected no significant differences, except for user-

based diversity scores (*divU*) in the movie ($Z = -2.69, p < 0.05$) and the book domain ($Z = -2.56, p < 0.05$). The results for the content-based diversity (*divC*) metric seem to point in the same direction, because of the high mean values for cross-domain queries in each domain. But the significance of these results was not verified. The *mrs* scores indicate a better performance of regular SPARQL queries, but the within-subjects test did not confirm this assumption statistically (see Tab. 19).

However, the most important finding of the analysis was that cross-domain queries generated significantly larger lists throughout the domains. (movie: $t(49) = -17.92, p < 0.001$, music: $t(52) = -25.73, p < 0.001$, book: $t(50) = -12.75, p < 0.001$) (see Tab. 19). Figure 13 illustrates this graphically.

In the case of a successfully answered SPARQL query, the result set almost always contained only a few items, due to data sparsity. In contrast to that, the SKOSRec engine was not only able to generate recommendations for practically each cross-domain request, it also mostly provided the required number of 10 recommendations. The verified increase in result set size is a remarkable outcome, given the fact that no significant differences were identified between the two

Table 19
Performance results of cross-domain recommendation retrieval

| Dimension (Metric) | Domain | Regular (SPARQL) | | Cross-Domain | | N |
|---------------------------|--------|------------------|--------|-----------------|-------------|------|
| | | M | SD | M | SD | |
| Accuracy (<i>size</i>) | Movie | 0.82 | 2.09 | 8.66*** | 2.73 | 50 |
| | Music | 0.89 | 1.59 | 9.30*** | 2.03 | 53 |
| | Book | 2.90 | 3.98 | 10.00*** | 0.00 | 51 |
| Accuracy (<i>mrs</i>) | Movie | <u>67.41</u> | 27.62 | 53.65 | 14.71 | 16 |
| | Music | <u>62.36</u> | 31.76 | 48.32 | 22.49 | 20 |
| | Book | <u>66.42</u> | 21.28 | 55.59 | 16.84 | 27 |
| Accuracy (<i>prec</i>) | Movie | <u>0.84</u> | 0.32 | 0.68 | 0.30 | 16 |
| | Music | <u>0.77</u> | 0.41 | 0.63 | 0.37 | 20 |
| | Book | <u>0.85</u> | 0.26 | 0.74 | 0.29 | 27 |
| Accuracy (<i>ndcg</i>) | Movie | <u>0.91</u> | 0.25 | 0.88 | 0.09 | 16 |
| | Music | <u>0.94</u> | 0.22 | 0.83 | 0.24 | 20 |
| | Book | <u>0.94</u> | 0.09 | 0.91 | 0.10 | 27 |
| Novelty (<i>nv</i>) | Movie | <u>0.59</u> | 0.49 | 0.41 | 0.34 | 16 |
| | Music | <u>0.72</u> | 0.44 | 0.59 | 0.40 | 19 |
| | Book | 0.54 | 0.41 | <u>0.63</u> | 0.30 | 27 |
| Diversity (<i>divU</i>) | Movie | 3 | - | 4* | - | 16 |
| | Music | 3 | - | 3 | - | 20 |
| | Book | 3 | - | 4* | - | 27 |
| Diversity (<i>divC</i>) | Movie | (0.64) | (0.35) | (0.99) | (0.02) | (6) |
| | Music | (0.90) | (0.13) | (0.98) | (0.03) | (10) |
| | Book | (0.84) | (0.12) | (0.87) | (0.1) | (8) |
| Usefulness | Movie | <u>3.57</u> | 0.83 | 3.35 | 1.57 | 15 |
| | Music | <u>3.53</u> | 1.17 | 3.11 | 0.98 | 20 |
| | Book | 3.41 | 1.04 | 3.41 | 0.94 | 27 |

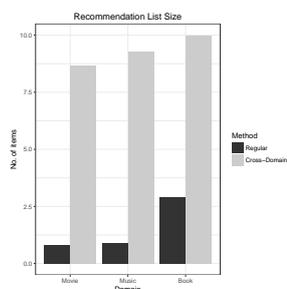


Fig. 13. Mean recommendation list sizes for regular and cross-domain recommendation retrieval

approaches regarding *Perceived Usefulness*. Therefore, it is reasonable to assume that the capability of the SKOSRec engine to flexibly switch between processing steps of similarity calculation and pattern matching, can facilitate an improved exploration of LOD repositories.

Table 20 also shows that the items in the two sets often did not match, as is indicated by low Jaccard indices. This finding further demonstrates that the execution of an additional step of similarity calculation can help to retrieve other relevant items. Regarding practical implications, these findings

Table 20
Jaccard Indices

| Domain | Mean Jaccard Index (JI) | SD | N |
|--------|-------------------------|------|----|
| Movie | 0.03 | 0.07 | 44 |
| Music | 0.01 | 0.02 | 21 |
| Book | 0.16 | 0.27 | 33 |

suggest that the two approaches could be combined into a single SKOSRec query pattern to achieve the best outcome.

5. Discussion & Future Work

This paper has presented the query-based SKOSRec engine that facilitates new types of recommendation requests for LOD repositories. The evaluations have shown that the SKOSRec system can often generate relevant and useful suggestions when certain query templates are utilized. In many cases, the application of expressive or advanced query patterns (i.e., in the context of constraint-based retrieval or cross-domain requests) helped to significantly improve recall values, while still providing the same level of quality in the

remaining performance dimensions. Additionally, advanced recommendation requests can be used to generate diversified recommendation lists, in case users are not satisfied with the results of regular recommendation requests (i.e., roll-up or cross-domain retrieval vs. regular requests). While not being superior in each domain and test case, is at least safe to say that the novel recommendation approaches of the SKOSRec engine considerably extend common retrieval methods and at least provide an alternative search strategy when conventional methods fail to provide useful results.

It is a strength of the developed approaches that they facilitate combinations of graph-based and similarity-based retrieval at different stages of the recommendation workflow. This feature extends general search capabilities for semantic networks. It requires future research to investigate whether, in addition to SKOS, further RDF vocabularies can be used for ad-hoc item-to-item similarity computation to increase the range of potential usage scenarios. In this context, it also needs to be determined whether the representation of the user profile in the SKOSRec query language can be substituted with free-text expressions to enable search-like functionalities. LOD researchers have already developed engines that can process natural language queries over RDF data [56], [34], [63]. It will have to be investigated how the SKOSRec engine profitably fits into this landscape of existing retrieval tools.

Another open research question concerns the aspect of interfaces to other applications. In its standard configuration, the system is not an end-user retrieval engine, but a backend application, with which an administrator, who has domain knowledge (e.g. of RDF vocabularies and data models of a particular usage scenario), creates the respective query patterns. Although the websites of the online experiments represent possible implementations of suitable end-user applications for the SKOSRec engine, there are still many other possibilities and extensions to design such a user interface (UI). For instance, the selection of LOD items for the profile was made possible by previously storing the data in an index that could then be accessed and searched before issuing a recommendation query. However, this runs against the idea of ad-hoc retrieval. In this context, future research will have to determine whether it is feasible to generate an on-the-fly mapping from items to the respective LOD resources. In the case of a commercial application, the engine needs to match items with the corresponding entries in the product catalog. Even if it is not yet clear which UI components can make the best possible use of the SKOSRec engine's features, the results from the user experiments can give at least a few suggestions.

The following retrieval options might be the default setting in an interface that contains a LOD-enabled RS component to assist users in finding interesting items: Since the regular approach of on-the-fly retrieval achieved good results throughout the domains, the display of simple recommendations generated from previously stated user preferences

represents a suitable starting point for retrieval. On-the-fly queries could be refined by providing filter options. Since the evaluations have shown that expressive constraints often lead to increased recall values, the application of such a filter might be the best option for assisted retrieval. As in the preparations for the experiments, appropriate graph-based query patterns for the usage scenario in question would also have to be determined by a domain expert before setting up the application. The same applies to advanced retrieval patterns, which should be available to the end user to refine the query when both the simple as well as the constraint-based approach have failed to provide relevant items. This suggestion is made because roll-up queries were competitive with regular recommendations in the web-based experiments. Therefore, they represent a viable alternative retrieval strategy. For some domains (e.g., for multimedia retrieval) it is also possible to use an optional selection field to enable the formulation of cross-domain requests. Evaluations for this query type suggest that users might receive interesting results (i.e., diversified recommendation lists) from such a request.

The prototypical implementation and the evaluation of the SKOSRec engine in different usage scenarios have demonstrated that the freely available knowledge sources on the LOD cloud can be successfully applied to advance LOD-enabled retrieval approaches. The novel SKOS-based recommendation methods proposed by this work, namely expressive graph-based filters and advanced queries are promising alternatives to existing retrieval strategies for semantic networks.

References

- [1] G. Adomavicius, A. Tuzhilin.: Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. In: *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.
- [2] G. Adomavicius, A. Tuzhilin, R. Zheng: REQUEST: A query language for customizing recommendations. In: *Information Systems Research*, vol. 22, no. 1, pp. 99–117, 2011.
- [3] C. Aggarwal: *Recommender systems*, Springer, 2016.
- [4] M. Arenas, B. Cuenca Grau, E. Kharlamov, S. Marcuska, D. Zheleznyakov: Faceted search over ontology-enhanced RDF data. In: *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management*, pp. 939–948, 2014.
- [5] V. Ayala, M. Przyjaciół-Zablocki, T. Hornung, A. Schätzle, G. Lausen: Extending SPARQL for recommendations. In: *Semantic Web Information Management*, 2004.
- [6] J. Beel, C. Breiting, S. Langer, A. Lommatzsch, B. Gipp: Towards reproducibility in recommender systems research. In: *User Modeling and User-adapted Interaction*. vol. 26, no. 1, pp. 69–101, 2016.
- [7] C. Boutilier, R. Brafman, C. Domshlak, H. Hoos, D. Poole: CP-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. In: *Journal of Artificial Intelligence Research (JAIR)*, vol. 21, pp. 135–191, 2004.

- [8] P. Castells, S. Vargas, J. Wang: Novelty and diversity metrics for recommender systems: choice, discovery and relevance, 2011.
- [9] G. Cheng, W. Ge, Y. Qu: Falcons: searching and browsing entities on the Semantic Web. In: *Proceedings of the 17th ACM International Conference on World Wide Web*, pp. 1101–1102, 2008.
- [10] J. Davies, R. Weeks: QuizRDF: Search technology for the Semantic Web. In: *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, 2004.
- [11] DBpedia, URL: <https://wiki.dbpedia.org/>, 2017.
- [12] T. Di Noia, R. Mirizzi, V. Ostuni, D. Romito: Exploiting the web of data in model-based recommender systems. In: *Proceedings of the 6th ACM Conference on Recommender Systems (RecSys)*, pp. 253–256, 2012.
- [13] T. Di Noia, R. Mirizzi, V. Ostuni, D. Romito, M. Zanker: Linked Open Data to support content-based recommender systems. In: *Proceedings of the 8th International Conference on Semantic Systems*, 2012.
- [14] D. Dillman, D. Robert, D. Bowker: Principles for constructing web surveys. In: *Joint Meetings of the American Statistical Association*, 1998.
- [15] DCMI metadata terms, URL: <http://dublincore.org/documents/dcmi-terms/>, 2004
- [16] I. Fernandez-Tobias, I. Cantador, M. Kaminskas, F. Ricci: A generic semantic-based framework for cross-domain recommendation. In: *Proceedings of the 2nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems*, pp. 25–32, 2011.
- [17] A. Freitas, J. Oliveira, S. O’Riain, E. Curry, J. Da Silva, C. Pereira: Querying Linked Data using semantic relatedness: a vocabulary independent approach. In: *Natural Language Processing and Information Systems*, pp. 40–51, 2011.
- [18] T. Grainger, T. Potter, Y. Seeley: Solr in action. Cherry Hill: Manning, 2014.
- [19] A. Gunawardana, G. Shani: Evaluating recommender systems. In: *Recommender Systems Handbook*. Springer, pp. 265–308, 2015.
- [20] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, U. Scheel: Faceted Wikipedia search. In: *International Conference on Business Information Systems*, pp. 1–11, 2010.
- [21] S. Harris, A. Seaborne, E. Prud’hommeaux: SPARQL 1.1 query language. In: *W3C recommendation*, 2013.
- [22] B. Heitmann, C. Hayes: Using Linked Data to build open, collaborative recommender systems. In: *AAAI Spring Symposium: Linked Data Meets Artificial Intelligence*, pp. 76–81, 2010.
- [23] J. Herlocker, J. Konstan, L. Terveen, J. Riedl: Evaluating collaborative filtering recommender systems. In: *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.
- [24] Y. Hu, Z. Wang, W. Wu, J. Guo, M. Zhang: Recommendation for movies and stars using YAGO and IMDB. In: *12th International Asia-Pacific Web Conference (APWEB)*, pp. 123–129, 2010.
- [25] D. Huynh, D. Karger: Parallax and companion: Set-based browsing for the data web. In: *Proceedings of the 18th ACM International Conference on World Wide Web*, 2009.
- [26] D. Jannach, M. Zanker, A. Felfernig, G. Friedrich: Recommender systems: an introduction. Cambridge University Press, 2010.
- [27] Y. Kabutoya, R. Sumi, T. Iwata, T. Uchiyama, T. Uchiyama. In: *IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, pp. 625–630, 2012.
- [28] M. Kaminskas, I. Fernandez-Tobias, F. Ricci, I. Cantador: Knowledge-based music retrieval for places of interest. In: *Proceedings of the 2nd International ACM Workshop on Music Information Retrieval with User-centered and Multimodal Strategies*, pp. 19–24, 2012.
- [29] H. Khrouf, R. Troncy: Hybrid event recommendation using Linked Data and user diversity. In: *Proceedings of the 7th ACM Conference on Recommender Systems*, pp. 185–192, 2010.
- [30] P. Kirk: Experimental design. John Wiley & Sons, 1982.
- [31] B. Knijnenburg, M. Willemsen, Z. Gantner, H. Soncu, C. Newell: Explaining the user experience of recommender systems. In: *User Modeling and User-Adapted Interaction*, vol. 22, no. 4-5, pp. 441–504, 2012.
- [32] G. Koutrika, B. Bercovitz, H. Garcia-Molina: FlexRecs: expressing and combining flexible recommendations. *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, 2009.
- [33] M. Lee, W. Kim: Semantic association search and rank method based on spreading activation for the Semantic Web. In: *IEEE International Conference on Industrial Engineering and Engineering Management*, pp. 1523–1527, 2009.
- [34] J. Lehmann, L. Bühmann: Autosparql: Let users query your knowledge base. In: *8th Extended Semantic Web Conference (ESWC)*, 2011.
- [35] N. Marie, F. Gandon, M. Ribiere, F. Rodio: Discovery hub: on-the-fly Linked Data exploratory search. In: *Proceedings of the 9th International Conference on Semantic Systems*, pp. 17–24, 2013.
- [36] S. McNeel, J. Riedl, J. Konstan: Being accurate is not enough: how accuracy metrics have hurt recommender systems. In: *CHI’06 Extended Abstracts on Human Factors in Computing Systems*, 2006.
- [37] R. Meymandpour, J. Davis: Recommendations using Linked Data. In: *Proceedings of the 5th PhD Workshop on Information and Knowledge*, 2012.
- [38] A. Miles, S. Bechhofer: SKOS Simple Knowledge Organization System Reference. In: *W3C Recommendation*, 2009.
- [39] B. Mobasher, X. Jin, Y. Zhou: Semantically enhanced collaborative filtering on the web. In: *Web Mining: From Web to Semantic Web*, pp. 57–76, 2004.
- [40] A. Musetti, A. Nuzzolese, F. Draicchio, V. Presutti, E. Blomqvist, A. Gangemi, P. Ciancarini: Aemoo: Exploratory search based on knowledge patterns over the Semantic Web. In: *Semantic Web Challenge*, 2012.
- [41] T. Neumann, G. Weikum: The RDF-3X engine for scalable management of RDF data. In: *The VLDB Journal*, vol. 19 no. 1, pp. 91–113, 2010.
- [42] J. Pérez, M. Arenas, C. Gutierrez: Semantics and Complexity of SPARQL. In: *5th International Semantic Web Conference*, pp. 30–43, 2006.
- [43] S. Policarpio, S. Brunk, G. Tummarello, G.: Implementation of a SPARQL integrated recommendation engine for Linked Data with hybrid capabilities. In: *AIMWD*, 2012.
- [44] Pu, P., Chen, L., Hu, R.: A user-centric evaluation framework for recommender systems. In: *Proceedings of the 5th ACM Conference on Recommender Systems*, 2011.

- [45] E. Oren, R. Delbru, S. Decker: Extending faceted navigation for RDF data. In: *5th International Semantic Web Conference*, pp. 559–572, 2006.
- [46] J. Rosati, T. Di Noia, T. Lukasiewicz, R. De Leone, A. Maurino: Preference queries with Ceteris Paribus semantics for Linked Data. In: *OTM Confederated International Conferences*, pp. 423–442, 2015.
- [47] T. Ruotsalo, K. Haav, A. Stoyanov, S. Roche, E. Fani, R. Deliai, E. Mäkelä, T. Kauppinen, E. Hyvönen: SMARTMUSEUM: A mobile recommender system for the Web of Data. In: *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 20, pp. 50–67, 2013.
- [48] I. Ruthven, M. Lalmas, K. van Rijsbergen: Incorporating user search behavior into relevance feedback. In: *Journal of the Association for Information Science and Technology*, vol. 54, no. 6, pp. 529–549, 2003.
- [49] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, J. Item-based collaborative filtering recommendation algorithms. In: *Proceedings of the 10th International Conference on World Wide Web*, 2001.
- [50] M. Schmachtenberg, C. Bizer, H. Paulheim: Adoption of the Linked Data best practices in different topical domains. In: *International Semantic Web Conference (ISWC)*, pp. 245–260, 2014.
- [51] H. Schütze, C. Manning, P. Raghavan: Introduction to information retrieval, Cambridge University Press, 2008.
- [52] SparqlImplementations. URL: <https://www.w3.org/wiki/SparqlImplementations>, 2016.
- [53] M. Stankovic, W. Breitfuss, P. Laublet: Discovering relevant topics using DBPedia. In: *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, pp. 219–222, 2011.
- [54] K. Swearingen, R. Sinha: Beyond algorithms: An HCI perspective on recommender systems. In: *ACM SIGIR 2001 Workshop on Recommender Systems*, vol. 13, no. 5-6, 2001.
- [55] D. Tunkelang: Faceted search. In: *Synthesis Lectures on Information Concepts, Retrieval, and Services*, pp. 1–80, 2009.
- [56] C. Unger, L. Bühmann, J. Lehmann, A. Ngonga Ngomo, D. Gerber, P. Cimiano: Template-based question answering over RDF data. In: *Proceedings of the 21st ACM International Conference on World Wide Web*, 2012.
- [57] C. Varnhagen, M. Gushta, J. Daniels, T. Peters, N. Parmar, D. Law, T. Johnson: How informed is online informed consent? In: *Ethics & Behavior*, vol. 15, no. 1, pp. 37–48, 2005.
- [58] J. Waitelonis, H. Sack, H. Towards exploratory video search using Linked Data. In: *11th IEEE International Symposium on Multimedia*, pp. 540–545, 2009.
- [59] L. Wenige: Knowledge Organization Systems for Linked Data-enabled on-the-fly recommendations.
- [60] L. Wenige, J. Ruhland: Retrieval by recommendation: using LOD technologies to improve digital library search. In: *International Journal on Digital Libraries*, 2017.
- [61] F. Zarrinkalam, M. Kahani: A multi-criteria hybrid citation recommendation system based on Linked Data. In: *2nd International EConference on Computer and Knowledge Engineering (ICCKE)*, pp. 283–288, 2012.
- [62] C. Ziegler, S. McNee, J. Konstan, G. Lausen: Improving recommendation lists through topic diversification. In: *Proceedings of the 14th International Conference on World Wide Web*, pp. 22–32, 2005.
- [63] L. Zou, R. Huang, H. Wang, J. Yu, W. He, D. Zhao: Natural language question answering over RDF: a graph data driven approach. In: *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 313–324, 2014.