A Query Language and Ranking Algorithm for News Items in the Hermes News Processing Framework

Frederik Hogenboom*, Damir Vandic, Flavius Frasincar, Arnout Verheij, Allard Kleijn

Erasmus University Rotterdam, P.O. Box 1738, NL-3000 DR, Rotterdam, The Netherlands

Abstract

Hermes is a Web-based framework that makes use of many Semantic Web technologies for building personalized news services. Ontologies are employed for knowledge representation, natural language processing techniques are used for semantic text analysis, and semantic query languages enable the specification of the desired information. To accommodate for the need for an intuitive way to create complex queries for news information, we present the Hermes Graphical Query Language (HGQL). The language enables users to create structured queries that use disjunctive, conjunctive, negation, and pattern operators. In addition, this paper presents a ranking algorithm based on the queries made using our graphical query language. Results show that our proposed ranking algorithm significantly outperforms three state-of-the-art ranking algorithms and that users prefer our graphical query language over a text-based alternative.

Keywords: query languages, news ranking, ontology-based querying

1. Introduction

One of the major problems that arise as a result of today's unstoppable growth of the Web, is the information overload daily users are confronted with. This calls for a methodical approach to information filtering, for example news recommendation [1, 2, 3], in such a way that the presented subset of results actually represents the individual user's preferences. Many information retrieval techniques are already available, of which keyword matching is the most common one. In such approaches, user-specified keywords are matched to the available textual data, resulting in a relevant selection of information matching these keywords. However, a problem with this approach is the lack of semantics, i.e., the meaning of words is not taken into account. For example, the keyword 'apple' could refer to fruit, the company, or even a person's name.

In order to deal with the previously identified semantic issue while processing (news) text, in earlier work we have introduced the Hermes news personalization framework [4, 5] which is built upon Semantic Web technologies. The news items are gathered from RSS feeds provided by the user. Hermes is composed of multiple natural language processing resources that enable the processing (and querying) of news. Also, by employing a plug-in-based software architecture, the Hermes framework allows for the addition of user-created specialized information processing plug-ins. By default, the framework stores lexicalized domain concepts and relations (i.e., properties that relate concepts to each other or to data types) in a domain ontology. The ontology also stores synonyms (string representations) of domain-specific entities like companies, persons, etc., as well as their relations, such as subsidiary and competitor relations. The domain ontology is used to index news as well as to retrieve relevant news items in a semantically-enhanced way. In addition, we have proposed ontology-based recommendation plug-ins that also benefit from Hermes' ontology and the news processing framework [2, 6].

However, semantics-based matching alone is not enough in order to provide a good personalized news service, as user preferences also need to be

*Corresponding author; tel: +31 (0)10 408 8907; fax: +31 (0)10 408 9031

Email addresses: fhogenboom@ese.eur.nl (Frederik Hogenboom), vandic@ese.eur.nl (Damir Vandic), frasincar@ese.eur.nl (Flavius Frasincar), 308057av@student.eur.nl (Arnout Verheij), 303118ak@student.eur.nl (Allard Kleijn)

elicited. This could be achieved by letting the user specify queries that express the concepts of interest. Therefore, in previous work we have devised a text-based query language that makes use of linguistic patterns that incorporate lexical, syntactic, and semantic elements [7], and we have successfully implemented the language into Hermes as a plug-in. However, in order to aid the average, day-to-day user, who is not an expert in information technology, with creating a query, graphical query languages can prove to be useful here. These structured graphical languages aim to minimize the amount of effort the user has to put into formulating queries, by providing a less complex syntax, and by using only Boolean (AND, OR, and NOT) and sequence operators (in the form of a sentence, i.e., they have a subject, a predicate, and an object). Based on our previous experience with creating a graphical query language for RDF, RDF-GL [8], in recent work [9], we have introduced a graphical query language designed for Hermes: the Hermes Graphical Query Language (HGQL).

Although semantics-based queries generated through a graphical query language provide the user with a subset of potentially interesting news items, these items need to be ranked according to their relevance to the user query. Several weighting schemes for concept importance have been proposed in the literature. Most of the schemes can cope with AND and/or OR operators for queries, but few solutions have been devised to use these operators together with the NOT operator. Hence, we propose to enhance the extended Boolean model [10] with the negation operator.

This paper builds on recent work [9] and has four main contributions with respect to the state-of-the-art. First, we describe a graphical query language for searching news that goes beyond current keyword-based approaches, and which is additionally supported by an implementation in Hermes. Compared to our earlier work, we provide more details on the main language elements and grammar, as well as the Hermes framework and the implementation of HGQL. Second, we devise a ranking algorithm for sorting news that effectively supports the negation operator. Third, in contrast to our recent work, in our current endeavours, we provide a quantitative and qualitative evaluation of the proposed query language. Last, we provide an extensive, more detailed, evaluation of our ranking algorithm with respect to several vector space model weighting schemes.

The rest of the paper is organized as follows. First, Section 2 provides a more thorough description of the Hermes framework. Next, Section 3 describes related work on graphical query languages and relevance ranking algorithms. Section 4 proposes the Hermes Graphical Query Language (HGQL). Section 5 devises a ranking algorithm for HGQL, and Section 6 discusses the implementation of the language and its ranking algorithm. The query language and algorithm are evaluated against other approaches in Section 7. Last, Section 8 presents our conclusions and suggests future work.

2. Hermes

The Hermes framework [1, 4], as depicted in Figure 1, is comprised of a sequence of steps for building a personalized news service. The system's inputs are RSS news feeds, whereas its outputs are filtered (relevant) news items. The core of the Hermes framework is a domain ontology developed by domain experts, employed for indexing news items and for formulating user queries. In addition, the user can specify temporal constraints that news items need to satisfy. The resulting news items are sorted based on their relevance for the user queries.

After removing the duplicate news items from the input RSS feeds through heuristics by lexically comparing news message titles, the news is processed in three subsequent steps, i.e., news classification, news querying, and results presentation. News classification is responsible for indexing the news items based on ontology concepts. News querying consists of two substeps: query formulation, i.e., helping the user build the query that expresses the items of interest, and query execution, i.e., computing the results of query evaluation. In the last processing step, the resulting news items are presented based on their relevance to the user interests.

2.1. News Classification

The ontology concepts that are employed in the news item classification stage are classes and individuals from the domain ontology. Concepts are linked to sets of synonyms (synsets) from a semantic lexicon such as WordNet [11], identifying their unique meaning. The synonyms contained by a synset are treated as lexical representations of the associated ontology concept. General semantic lexicons are domain independent. Hence, for domain

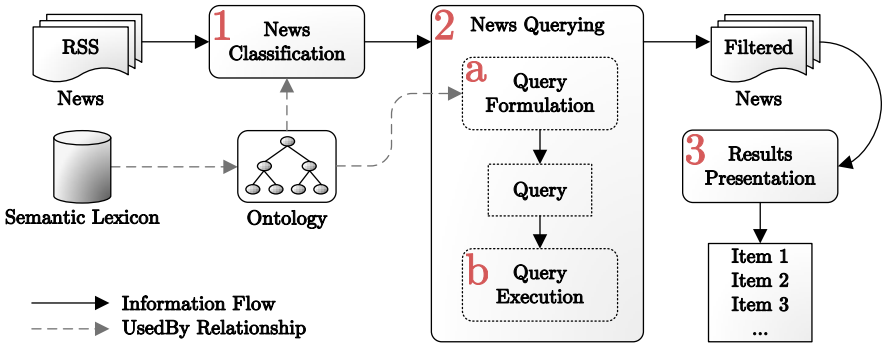

Figure 1: Overview of the Hermes framework.

dependent concepts in our ontology, we associate additional domain specific lexical representations to their corresponding concepts. These lexical representations are composed only of word lemmas (the canonical word form appearing in dictionaries).

The classification approach is ontology-centric, implying that during news item processing, ontology concepts are loaded one at a time and their lexical representations are matched against the news items. This approach is more efficient than a news item-centric algorithm, as the number of concepts in the ontology is considerably larger than the number of words in the news items (for batch processing of news items).

Before ontology concepts can be matched, however, a few basic processing tasks have to be performed in order to prepare the plain text to be machine interpretable. First tokenization, sentence splitting, part-of-speech tagging, and morphological analysis are performed. The tokenization precedes sentence splitting as sentence splitting requires punctuation tokens stemming from tokenization. Morphological analysis follows part-of-speech tagging because the lemma of a word depends on its part-of-speech tag. In this way, all words in a news item are reduced to their canonical form, a form shared also by the lexical representations of concepts stored in the domain ontology.

After these processing steps, lexical representations of the ontology concepts are matched with identified (groups of) lemmas in the news messages. As the same lexical representation can belong to different concepts, for each match, the word sense of the lemma identified in the text is disambiguated using the adapted Lesk algorithm [12]. This algorithm retrieves the glossary of a lemma sense, as well as the glossaries of related senses from a semantic lexicon, and calculates similarity scores between each considered sense's glossary words and the context words (from the sense's sentence). After disambiguating the senses of matching lemmas using the previously computed highest similarity scores, news items are linked to the corresponding concepts in the domain ontology.

2.2. News Querying

Within the Hermes framework, the user is able to express the topics of interest through queries that make use of concepts from the domain ontology. In addition, the user can express time constraints that need to be satisfied. In order to assist the query construction process, the user is presented a conceptual graph representation of the ontology, which gives insight into the overall structure of the domain at hand. Within this graph, the user is able to select the concepts of interest, which are added to a search graph. This graph has generalized disjunctive semantics with respect to the included concepts, indicating that the user is interested in any of the search graph concepts to be found in news items, but has a stronger preference for more of the specified concepts appearing in a top ranked news item in the result list. For each of the concepts in the search graph, all lexical representations are taken into consideration by the framework when filtering news messages.

Additionally, the user can employ time comparison or arithmetic operators and retrieve the current time in order to build complex time expressions. Moreover, the system provides predefined temporal constraints such as: last day, last week, last two weeks, last three months, last quarter, last half year, and last year. The temporal conditions that model these constraints have conjunctive semantics as they need to be fulfilled in the same time.

The specified time constraints and the concepts of interest contained in the search graph are converted into a query formulated in a text-based semantic query language, e.g., SPARQL [13], or tSPARQL [4, 5] (a SPARQL extension with time-specific features). Subsequently, news items linked to the concepts of interest are retrieved.

2.3. Results Presentation

As a last step in the framework, the retrieved news items (matching the user-defined criteria) are ordered descendingly according to their relevance and are displayed in a visually appealing manner explaining their relevance to the user query.

News items can be ranked in many ways. In Hermes, news ranking is generally performed by calculating a relevance agree, which is defined as a weighted sum of the number of hits, where the weights depend on the hits location (either the title or the body of a news item). News items that have the same degree of relevance are sorted descendingly, based on their associated time stamps. In this paper we propose and evaluate a more advanced ranking algorithm, and compare its performance to other ranking algorithms.

Last, the results are presented to the user as a list of summaries, containing title, source, date, and a few lines from the news item. Also, the relevance degrees are shown. Moreover, for each returned news item, the identified lexical representations are emphasized in the news item text, thereby offering to the user an explanation of why a certain news item is considered to be relevant.

3. Related Work

This section gives an overview of related research in the field of information retrieval, more specifically on graphical representations of queries and relevance ranking algorithms.

3.1. Graphical Query Languages

One of the merits of graphical query representations is that they are considered to be easier to comprehend by humans than their textual counterparts, under the assumption that the representations are not too complex. Additionally, textual representations often require the user to be familiar with a certain complex syntax, steepening the learning curve. These factors continue to drive the development of graphical query languages.

In [14], a technique for building graphical queries for RDF is introduced. The RDF data model is based on triples, which are composed of a subject, a predicate, and an object. The language discussed in [14] uses rounded rectangles with a predicate and optionally an object to visualize queries. Multiple of these rounded rectangles can be interconnected, creating conjunctive queries (i.e., by using a logical AND). Object nesting is also supported, enabling resource linking. Hence, it is possible to create a query, for which the object is defined by another query. A disadvantage of this language is that it does not provide support for disjunctive queries.

RDF-GL [8] is a graphical query language for RDF and is based on the standard query language SPARQL. This language does support a wider set of operators than [14], such as conjunction, disjunction, and negation. It uses boxes, circles, and arrows in different colors to represent its elements. RDF-GL covers the SELECT statement from SPARQL, yet this wide coverage of the SPARQL language makes the graphical language too complex for our intended use.

Another graphical query language is GLOO [15], in which the user can specify concepts, individuals, relations, and logical operators (AND/OR). Ovals, squares, and arrows are the building blocks for GLOO. GLOO is a powerful graphical query language, allowing the user to make relatively complex queries in an intuitive way. However, there is no method discussed for the implementation of filters, such as those in RDF-GL, nor is there built-in support for negation.

The authors of [16] propose an approach to query databases in a graphical way using the 'Query by Diagram' system. The system uses the Entity-Relationship Model as a conceptual model. Although one could adapt the Entity-Relationship Model to support RDF, the semi-structured nature of RDF data would cause many issues when dealing with open world representations.

In [17], a graphical query language is proposed that avoids the explicit use of Boolean operators. It is designed specifically for non-technical users that require an easy-to-use language. Our approach differs from this approach, as we explicitly focus on Boolean operators. The group of users that we are targeting need to be using such operators in an effective and efficient way. The only assumption that we make about the users of HGQL is that they are not familiar with existing graph query languages, such as SPARQL.

3.2. Relevance Sorting Algorithms

In order to sort query results, many algorithms have been developed. Most of these approaches employ a model which relies on additional term weighting procedures.

3.2.1. Ranking Models

The most elementary ranking model is the Boolean model [18], which uses the structured query operators AND representing the logical product, OR representing the logical sum, and NOT representing the logical difference. However, this model does not provide any ranking mechanism for text querying.

One of the earliest models for ranked retrieval is the vector space model [19]. The vector space model is based on measuring the similarity between a query and a document. The document vector is $d = (d_1, d_2, \ldots, d_m)$ of which each component d_k $(1 \leq k \leq m)$ is associated with an index term (i.e., a word, a keyword, or a longer phrase). An index term is a term occurring in the documents or the query (the vocabulary). For the query there is a similar vector $q = (q_1, q_2, \ldots, q_m)$ of which the components are associated with the same terms. The vectors contain binary values indicating whether a term occurs in the document or query, or numerical values to indicate the importance of a term. The vector space model maps every term to a different dimension. The query and document are considered vectors in the high dimensional Euclidean space determined by the terms. The similarity is measured by taking the cosine of the angle between the query vector and the document vector. If the vectors are normalized, the cosine is merely a vector inner product [20]. There is a need for an additional term weighting model, because the vector space model does not describe the values of the vector components. Also, the model is only able to cope with conjunctive queries.

The vector space model can be extended to the p-norm extended Boolean model so that it is able to additionally support disjunctive queries. Let us consider a query of 2 terms and that the vectors are normalized to unit length. Point (1,1) represents the situation that both query terms are present with weight 1. Point (0,0) represents the situation that both query terms are not present. Documents in point (1,1) have the highest relevance for conjunctive queries while documents in point (0,0) have the lowest relevance if the query is disjunctive. Therefore, the documents should be ranked in order of increasing distance from the point (1,1) for conjunctive queries and in order of decreasing distance from point (0,0) for disjunctive queries. This reasoning gives the definition of the following scores [19]:

$$score\,(d, a \text{ OR } b) = \sqrt{\frac{(d_a - 0)^2 + (d_b - 0)^2}{2}}\,, \tag{1}$$

$$score\,(d, a \text{ AND } b) = 1 - \sqrt{\frac{(1 - d_a)^2 + (1 - d_b)^2}{2}}\,. \tag{2}$$

Generalizing these formulas and extending them with term weights introduces a p-norm that provides a certain softness to Boolean operators [19]:

$$score\left(d, q \text{ OR}_{(p)}\right) = \left(\frac{\sum_{k=1}^{m}(q_k)^p (d_k)^p}{\sum_{k=1}^{m}(q_k)^p}\right)^{1/p}\,, \tag{3}$$

$$score\left(d, q \text{ AND}_{(p)}\right) = 1 - \left(\frac{\sum_{k=1}^{m}(q_k)^p (1 - d_k)^p}{\sum_{k=1}^{m}(q_k)^p}\right)^{1/p}\,. \tag{4}$$

Salton et al. propose a recursive algorithm in order to cope with queries that contain both conjunctive and disjunctive operators [10]. The use of '-1' for query terms with the NOT-operator is mentioned in literature [21] in order to cope with negation queries. Terms that do not appear in the document are represented as '-1' in the document vector. This approach has, nevertheless, neither theoretical nor empirical support so far. The vector space model and the p-norm extended Boolean model both work with all the terms appearing in all documents and in the query. In Hermes, concepts are used instead of terms, so with this adjustment these models can be reused for Hermes.

Another solution aiming at incorporating negation in vector spaces is based on linear algebra [22]. A negated concept is represented as its orthogonal complement under the scalar product. Compared to this approach, our current endeavours aim to build on the (computational) simplicity of the extended Boolean model proposed by Salton [10] by enhancing it to be able to deal with the negation operator.

Fuzzy set-based information retrieval models rank the documents based on the degree of membership of the document to the terms in the query. Paice's model [23] extends the basic fuzzy models

by adding different computations of membership for disjunctive and conjunctive queries, but it does not support negation queries. Like the vector space model, this model needs an additional term weighting algorithm.

Other approaches focus on ranking complex relationship search results, employing the concept relationship between the appearing concepts, on the Semantic Web. The work done in [24] is an example of such an approach. The authors employ a ranking model that takes the relationship structure between entities into account. Stojanovic et al. [25] apply inference on ontologies for ranking entities on the Semantic Web. Because we focus on news retrieval with a relatively simple annotation structure, such approaches are out of the scope of the paper.

3.2.2. Term Weighting

Most of the earlier discussed relevance ranking algorithms rely on additional term weighting procedures, of which many have been developed in the past decades [26]. Essentially, term weighting models treat the query like a document. Both the document and the query terms are assigned a weight, and subsequently the similarity of these weights is calculated using one of the models described in the previous section.

Most modern weighting algorithms are based on the Term Frequency – Inverse Document Frequency (TF-IDF) [27] concept. Term Frequency (TF) weighting counts how often a term occurs in the document and query. The more often a term occurs in a document, the more relevant that term is considered to be to that particular document. Inversed Document Frequency (IDF) weighting is the inverse of the number of documents a term occurs in. A term that occurs in a low number of documents is considered to be specific and therefore documents with this term should have a high weight. By multiplying TF and IDF values, one obtains the TF-IDF weights, which can be normalized using cosine normalization:

$$d_k = q_k = \frac{TF_k \cdot \log \frac{N}{df_k}}{\sqrt{\sum_{i=1}^{m} (TF_i \cdot \log \frac{N}{df_i})^2}} \,. \tag{5}$$

Here, d represents the document vector, q represents the query vector, TF is the term frequency, N is the total number of documents in the document collection, m is the total number of index terms, and df is the document frequency.

In 1988, Salton and Buckley proposed that the document and query weights should be mapped differently [28]. Consequently, many variations of TF-IDF have been proposed for this purpose, which have been assigned names according to a naming convention where two three-letter combinations are used. The first group represents the document term weight, and the second combination represents the query term weight. The first letter of such a combination indicates the TF component, the second letter the IDF component, and the third letter represents the normalization. An example of this is the tfc.nfc algorithm, which uses a normalized TF factor for the query weights. The original TF-IDF algorithm as previously described is called tfc.tfc.

An important discovery is that weights that are logarithmic in TF outperform weighting algorithms that are linear in TF [29]. An example of this is the lxc.ltc formula, where the 'l' stands for weights with a logarithmic TF or IDF component. The normalization part of these formulas is left out here because it follows the usual cosine normalization. For lxc.ltc, the weights are calculated as follows:

$$d_k = 1 + \log TF_k \,, \tag{6}$$

$$q_k = (1 + \log TF_k) \cdot \log \frac{N+1}{df_k} \,. \tag{7}$$

A more recent algorithm that claims to be outperforming the cosine normalization is the Lnu.ltu algorithm [30]. This algorithm uses a combination of the document length and the average document length for normalization:

$$d_k = \frac{1 + \log TF_k}{1 + \log TF_{avg}} \cdot \frac{1}{(1-s) + s \, \frac{uw}{uw_{avg}}} \,, \tag{8}$$

$$q_k = (1 + \log TF_k) \cdot \log \frac{N+1}{df_k} \cdot \frac{1}{(1-s) + s \, \frac{uw}{uw_{avg}}} \,, \tag{9}$$

where TF_{avg} is the average term frequency of all terms in document d_k, uw denotes the number of unique words in d_k, uw_{avg} represents the average number of unique words (taken over all documents d), and s is the slope factor that is dependent on the number of unique terms in the document and therefore experimentally determined. Slope s is optimized to 0.25 when using pivoted unique normalization in [31].

4. Hermes Graphical Query Language

This section discusses the theoretical framework for the Hermes Graphical Query Language (HGQL). First, we introduce the basic elements of an HGQL query, which is in essence a directed graph consisting of nodes (colored rounded rectangles) and edges (arrows). Then, we discuss the main forms of HGQL queries, followed by a presentation of the grammar of HGQL.

4.1. HGQL Elements

Elements in HGQL can be divided into two categories: nodes and edges. Nodes can represent concepts, wild cards, operators, and relations and take the form of rounded rectangles in different colors. Edges represent connections between nodes and take the form of lines with arrows at one end.

Concepts. The concepts that are used in HGQL are domain classes and individuals stemming from a domain ontology. These concepts can represent people, countries, companies, etc. Concepts are depicted as purple boxes and are denoted by noun phrases.

Relations. The relations allow the user to employ HGQL's triple-based pattern querying. By creating a query with the structure $Concept \rightarrow Relation \rightarrow Concept$, news items matching that pattern will be returned. The possible instances of a relation are extracted from the knowledge base. Relations are represented by verb phrases. Instances of the relation category are denoted as green nodes.

Wild Cards. A wild card is a construct in the HGQL that stands for a group of nodes, that represents either concepts or relations (partly) unknown to the user, and hence wild cards are essentially existential quantifiers over either concepts or relations. The wild card can be either 'unknown' or a text match. For example, the HGQL query 'Google UNKNOWN Yahoo' will return any news items in which a relation between *Google* and *Yahoo* exists. The text match provides the user with the ability to find concepts or relations for which he does not know the exact name. For example, in case the user is interested in Apple's CEO Steve Jobs, yet only remembers the last name *Jobs*, the user can use a text match node for *Jobs* and the concept with the closest lexical match will be used in the query. From now on, when we refer to concepts, this includes concept wild cards, and similarly, when referring to relations, this includes relation wild cards. Instances of the wild card category are represented in the color of the type of node they match, i.e., purple for concepts and green for relations, but with a different label (e.g., UNKNOWN, "Jobs", etc.).

Operators. The operators are a group of nodes in HGQL that allows the user to add logic to queries. There are two types of operators: intra-query and inter-query. Intra-query operators support logic within queries and we distinguish between conjunction (AND), disjunction (OR), and negation (NOT). The reason for using these specific operators is that the combination of them can be used to create any possible propositional logic formula. Although combining the conjunction and negation or the disjunction and negation logical operators is already enough to obtain logical completeness, we offer all logical operators in order to better support the user by giving a higher degree of freedom in expressiveness. Instances of the intra-query operators category are denoted as light blue nodes. Inter-query operators support logic (conjunction, disjunction, and negation) between different queries, allowing the user to process multiple queries simultaneously. Inter-query operators connect to the top node (i.e., the node that has no incoming edges) of the query; if the query is a (chained) triple-based pattern query, the inter-query operator connects to the first predicate in the predicate chain (root of predicates). Instances of inter-query operators are denoted as yellow nodes.

Edges. Edges provide connectivity between nodes (i.e., concepts, relations, wildcards, and operators) and consist of two types: logical connectivity edges and pattern connectivity edges. Logical connectivity edges allow for the connection of operators to concepts and relations. The arrows for logical connectivity edges always point from the operator node to concepts, wild cards, or relations and are colored light gray.

Pattern connectivity edges (depicted as black arrows) allow for the creation $Subject \rightarrow Predicate \rightarrow Object$ pattern structures by connecting the concepts, wild cards, and relations. If multiple subjects exist, logical connectivity edges connect these subjects to an operator, which in turn can be connected to a relation node or a node that is the root to multiple relations. The direction of a pattern connectivity edge determines the pat-

tern order of elements. Relations require incoming edges which connect to their subject(s) and outgoing edges which connect to their object(s).

4.2. HGQL Structure

A query in HGQL is represented as a directed graph. Three types of queries can be identified, i.e., queries consisting only of concepts, queries following the triple paradigm, and chained queries.

Concepts-only Queries. The most simple type of query one could create in HGQL is the concept-only query, which consists of one or more concept nodes connected by at least one operator, allowing for the construction of conjunctive, disjunctive, and negation queries (and their combinations). The root node of such a query is an operator, and all concepts (and other operators) are connected to this node. Edges run from the local root node to the non-root nodes and operator nesting is allowed. An example of a complex concepts-only query is shown in Figure 2, which represents a query for retrieving all news items in which *Dell* is mentioned, as well as either *Cisco* or *Mac OS*, but not *AMD*.

Triple-based Pattern Queries. In HGQL, the user is also allowed to query for specific patterns within news items by means of triple-based pattern queries. HGQL supports the *Subject* $\rightarrow$ *Predicate* $\rightarrow$ *Object* pattern, which is also employed in RDF. A triple-based pattern mimics the structure of a simple English sentence (subject-verb-object). A simple query of this type would be *Concept* $\rightarrow$ *Relation* $\rightarrow$ *Concept*, which can be extended by the use of logical operators in any of these three fields. In order to maintain the *Subject* $\rightarrow$ *Predicate* $\rightarrow$ *Object* structure, a triple based pattern query always has pattern connectivity edges pointing from the subject(s) towards the relation(s) and from there towards the object(s). When a triple-based pattern query has multiple subjects, relations, and objects, it can be divided into three trees that are interconnected at the root nodes by pattern connectivity edges. Hence, the root of the subjects connects to the root of the relations, which in turn connects to the root of the objects. An example can be seen in Figure 3, which expresses a query that retrieves all news items matching the patterns $AMD \rightarrow NewCompetitor/Buys \rightarrow Dell/MacOS$ and $CISCO \rightarrow NewCompetitor/Buys \rightarrow Dell/MacOS$. A news item needs to contain both of these patterns in order to be returned.

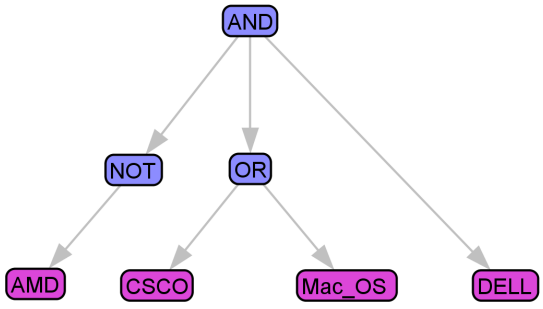

Figure 2: An example of a concepts-only query.

Chained Queries. A specific type of triple-based pattern queries are chained queries. Chained queries have the same properties as triple-based pattern queries, but additionally, it is possible to use the object of one triple pattern as the subject of another triple pattern. For example, the chained query in Figure 4 joins two triples. The query represented here retrieves all news items that match the pattern $AMD \rightarrow NewCompetitor \rightarrow Dell$ and the pattern $Dell \rightarrow Buys \rightarrow MacOS$.

4.3. HGQL Grammar

In this section, we provide the EBNF grammar of the HGQL language. First, we present the query types that are used in the grammar specification and how these relate to the HGQL queries. Second, for each of these query types, we explain how HGQL queries can be serialized to text. Last, we discuss the EBNF grammar that completely covers HGQL, which is used for checking the validity of HGQL queries.

In the previous section, we learned that HGQL queries can be one of three types: concepts-only queries, triple-based pattern queries, or chained queries. Furthermore, in Section 4.1, we distinguished between intra-query inter-query operators. Consequently, in our grammar specification, we describe the following three types of queries: concepts-only queries, pattern queries (describing triple-based and chained queries), and compound queries (queries that contain an inter-query operator).

For concepts-only queries, we use a recursive tree representation for the translation of HGQL queries to text (a query is a node with a list of children nodes). In fact, this translation is trivial because concepts-only queries are strict trees that have no

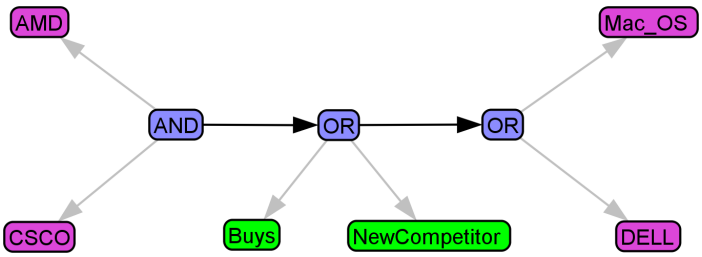

Figure 3: An example of a triple-based pattern query.

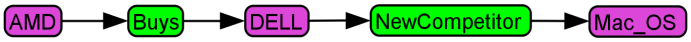

Figure 4: An example of a chained query.

floating nodes. For pattern-queries, we use a sequence of nodes separated by special characters (as shortly defined). For the last type, i.e, the compound queries, we use a representation that is a tree of query nodes. The translation of concepts-only and pattern-based queries to HGQL queries is trivial, as there is a direct connection between the two. For compound queries, we construct the HGQL based on the rules described in Section 4.1, i.e., inter-query operators connect to the root node of a concepts-only query and to the root of the first predicate of a triple-based or chained query.

The EBNF grammar of HGQL is given in Figure 5. A query can be a concept query, a pattern query, or a compound query. A concepts-only query (i.e., `concept_query`) consists of at least one AND, OR, or NOT operator. We further introduce the notion of a concept cover (i.e., `c_cover`). This is a recursive definition of a node that covers one or more concepts. The pattern query description uses this notion of a concept cover, as well as the notion of a relation cover (i.e., `r_cover`). The relation cover is a node that can represent one or more predicates. Next, a compound query consists of at least one AND, OR, or NOT operator that is applied to a query (i.e., either a concepts-only query or a pattern query). The EBNF further describes the way we serialize concepts and predicates from the knowledge base. Concepts start with an underscore ('_'), while predicates start with a hash sign ('#'). Additionally, the match node is described, which can be used to match concepts and relations using arbitrary text. The last part of the EBNF grammar describes the terminals for the HGQL nodes (AND, OR, NOT, and the UNKNOWN node).

Using this EBNF grammar, we can represent the HGQL query in Figure 2 as follows:

```
AND(
    NOT(_AMD),
    OR(
        _CSCO,
        _Mac_OS
    ),
    _DELL
)
```

The order of the nodes does not affect the result. Note that the nodes from the knowledge base (AMD, CSCO, Mac_OS, and DELL) start with an underscore. In this case, we have used new lines to make the query more readable, which is perfectly fine as newline characters (and other whitespace characters) can be ignored by the parser.

The HGQL query in Figure 3 can be serialized as follows:

```
AND(
    _AMD,
    _CSCO
)
->
OR(
    #buys,
    #is_competitor
)
->
OR(
    _Mac_OS,
    _DELL
)
```

Also in this case, we have made use of newline characters to make the query more readable. In the

```
query           : concept_query | pattern_query | compound_query

concept_query   : (AND | OR) '(' c_cover (',' c_cover)* ')'
                | NOT '(' c_cover ')'
c_cover         : concept
                | match
                | UNKNW
                | (AND | OR) '(' c_cover (',' c_cover)* ')'
                | NOT '(' c_cover ')'

pattern_query   : c_cover '->' r_cover '->' c_cover ('->' r_cover '->' c_cover)*
r_cover         : rel
                | (AND | OR) '(' r_cover (',' r_cover)* ')'
                | NOT '(' r_cover ')'

compound_query  : (AND | OR) '(' query (',' query)* ')'
                | NOT '(' query ')'

concept         : '_' ([0-9] | [a-z] | [A-Z]) ([a-z] | [A-Z] | '_')*
rel             : '#' ([0-9] | [a-z] | [A-Z]) ([a-z] | [A-Z] | '_')*
match           : 'MATCH("' ([0-9] | [a-z] | [A-Z] | '_')+ '")'

AND             : 'AND'
OR              : 'OR'
NOT             : 'NOT'
UNKNW           : 'UNKNW'
```

Figure 5: The specification of the HGQL grammar in EBNF format.

OR node we can see the two predicates 'buys' and 'is_competitor', which start with a '#' character to indicate that these nodes are predicate nodes.

5. Hermes Ranked Results

Within the Hermes news personalization framework, a list of news items that are likely to be interesting for a user is retrieved. For this purpose, a ranking mechanism is required that orders the items based on their relevance with respect to user-created queries. Hermes Ranked Results is an extension to the Hermes framework that ranks the news on relevance based on queries created in HGQL, the previously introduced graphical query language for Hermes.

5.1. Normal Forms

As it is unfeasible to define different ranking methods for each possible query, we need to normalize the query form. An example of a normal form is the negation normal form (NNF). A query is in negation normal form if negation is only applied to single concepts or patterns, and if only disjunctive, conjunctive, and negation operators are used. Two possible extensions of NNF are the disjunctive normal form (DNF) and the conjunctive normal form (CNF). A query takes on the disjunctive normal form if it is a disjunction of clauses, where a clause is a conjunction of possibly negated concepts or patterns. All rules for NNF also apply to DNF. A query is written in the conjunctive normal form if it is a conjunction of clauses, where a clause consists of a disjunction of concepts or patterns that are possibly negated. Technically, it is equivalent to DNF, except the disjunction and conjunction operators are swapped. Each query can be converted to NNF, DNF, or CNF [32].

In Hermes, DNF is used as normal form, interpreting a query as a disjunction of its conjunctive subqueries. The queries made in HGQL are converted to DNF using double negation elimination, De Morgan's laws, and the distributive law [33]. In order to be able to also rank the news for queries that besides concepts, also contain patterns, pat-

terns are treated just like concepts. For this, we first need to convert pattern queries to a form that is compatible with the ranking algorithm. In HGQL, one can create a pattern based query with logical operators within a single pattern. The ranking algorithm can only process queries that consist of simple patterns with operators between them. Such a simple pattern has only one subject, one predicate, and one object, and represents a so-called complex concept. A complex pattern, which is a pattern that uses logical operators in a pattern, is reduced to simple patterns (complex concepts) that are connected by logical operators, by outer moving the logical operators.

Chained queries are not an explicit part of HGQL. This type of query is simulated by two complete patterns connected by a conjunctive operator, where the first pattern's object is the same as the second pattern's subject.

5.2. Ranking Algorithm

After converting a regular HGQL query to a query in DNF, a ranking algorithm can be applied that makes use of the converted queries in order to sort news items based on their relevance to the user.

5.2.1. Document and Query Representation

In order to be able to use a relevance ranking algorithm, we first need to represent query and documents as weight vectors. In this study, we calculate term weights by means of four different algorithms – discussed Section 3.2.2 – to check which one returns the best results.

The first method is a simple binary weight, i.e., the extended Boolean method. The document weight is 0 if the concept does not occur and 1 if it does occur at least one time in the document. The same holds for the query weights. This method is an extension of the original extended Boolean model by Salton et al. [10], which does not support negation operators. Second, the basic TF-IDF weighting algorithm with cosine normalization (tfc.tfc) is used. The third method is lxc.ltc, which is a TF-IDF variation with logarithmic TF and IDF weights, as described in Equations (6) and (7). The last method used is the Lnu.ltu algorithm as described in Equations (8) and (9), which is said to be outperforming the previous algorithms using cosine normalization. This algorithm uses a combination of the document length and the average document length for normalization. As slope we use a value of 0.25, since it has been shown that this value provides the best results [31].

5.2.2. Disjunctive & Conjunctive Queries

Hermes uses a combination of the two formulas of the p-norm extended Boolean model, described in Equations (3) and (4), with p = 2. The latter formula addresses conjunctive queries, whereas the former formula copes with disjunctive queries. The input queries for this model are in DNF, implying that there are two parts, i.e., a disjunctive query and a collection of conjunctive subqueries. First, all the weights of the conjunctive subqueries are calculated by Equation (4). Then, using Equation (3), the total relevance of the document using the obtained weights of the subqueries is computed. The query weights in the formula that calculates the total score of the document are 1 because all subqueries are present one time. This results in the following relevance score calculation:

$$score(w_i \text{ OR}_{(p)}) = \left(\frac{\sum_{k=1}^{n} (w_i)^p}{n} \right)^{1/p}, \quad (10)$$

$$w_i \left(d, q \text{ AND}_{(p)}\right) = 1 - \left(\frac{\sum_{k=1}^{m} (q_k)^p (1 - d_k)^p}{\sum_{k=1}^{m} (q_k)^p} \right)^{1/p}, \quad (11)$$

where d is the document vector, q is the query vector, m is the total number of terms in the documents and query, n is the total number of conjunctive clauses, and w_i is the weight of clause i.

5.2.3. Negation

Up until now, the relevance scoring mechanism does not take in consideration the negation operator. Therefore, we make an adjustment to the query and document weights to cope with this operator. The query weights for concepts that are not part of the query are 0, and the query weights for concepts that occur in the query and are not negated are calculated with the four different methods described above. The weights of query concepts that are negated are computed in a similar way as before, although they are multiplied by -1. Document weights remain the same for concepts that occur in the document, while concepts that do not occur in the document are given weight -1 instead of 0. One can note the asymmetry in dealing with negations in documents and queries, as queries get the modulus of the term weight from the whole set of documents.

With the original vector inner product, this approach is correct with respect to the intended query semantics. The query and document weights are simply pairwise multiplied in this approach. Therefore, documents get a negative relevance when the query concept is negated while the concept does occur in the document, or when the query concept is not negated but the concept does not occur in the document. The relevance will be positive when a non-negated concept of the query occurs in the document, or a negated query concept does not occur in the document. Concepts that do not occur in the query will not affect the relevance score, as the assigned weight to these concepts in the query is 0, resulting in a relevance score of 0 when multiplied with the document weight.

Due to the changes in term weighting caused by also considering the negation operator, the ranking model described earlier needs to be updated accordingly, as it was assuming a Boolean weighting model. In some cases, the original ranking algorithm results in the same scores for negated and non-negated queries. If the document contains a term ($d_k = 1$), the query weight is multiplied with 0 according to Equation (11). This results in a value of 0 for the last term, independent from the value of the query weight. However, the equation provides incorrect relevance scores when a document does not contain a term ($d_k = -1$). In this case, the query weight q_k will be multiplied with a value higher than 1 ($1 - d_k = 1 - -1 = 2$), which may result in values below 0 (for the clause weight). As for the p-norm extended Boolean model, we aim for relevance schemes that are between 0 and 1, where high values denote high relevance.

In order to be able to calculate the relevance score of news items based on structured queries consisting of disjunctive, conjunctive, and negation operators, we need to adjust Equations (1) and (2). The OR part is based on the distance to the worst case (0,0). With negation we have a new worst case, i.e., $(-q_a, -q_b)$ for the specific case of two query terms a and b. In general, the worst case is $-q_k$. The AND part of this formula is based on best case (1,1). The new best case for negation queries in case of two query terms a and b is (q_a, q_b), so in general, the best case is q_k. The old formulas adjusted with this knowledge to make them compatible with negation queries result in Equations (12) and (13). For OR queries, $(d + q)^2$ is maximal when $d = q = 1$ or $d = q = -1$ and for AND queries $(q - d)^2$ is maximal when $q = -1$ and $d = 1$ or $q = 1$ and $d = -1$.

$$score\,(d, Q_{or}) = \sqrt{\frac{(d_a + q_a)^2 + (d_b + q_b)^2}{(2 \cdot q_a)^2 + (2 \cdot q_b)^2}}\,, \quad (12)$$

$$score\,(d, Q_{and}) = 1 - \sqrt{\frac{(q_a - d_a)^2 + (q_b - d_b)^2}{(2 \cdot q_a)^2 + (2 \cdot q_b)^2}}\,. \quad (13)$$

Q_{or} is either $a \vee b$, $a \vee \neg b$, $\neg a \vee b$, or $\neg a \vee \neg b$, and Q_{and} is either $a \wedge b$, $a \wedge \neg b$, $\neg a \wedge b$, or $\neg a \wedge \neg b$. From these formulas we can deduce the generalization for queries able to deal also with negations, resulting in Equations (14) and (15):

$$score\left(d, q\ \mathrm{OR}_{(p)}\right) = \left(\frac{\sum_{k=1}^{m}(q_k)^p(d_k + q_k)^p}{\sum_{k=1}^{m}(2 \cdot q_k)^p}\right)^{1/p}, \quad (14)$$

$$score\left(d, q\ \mathrm{AND}_{(p)}\right) = 1 - \left(\frac{\sum_{k=1}^{m}(q_k)^p(q_k - d_k)^p}{\sum_{k=1}^{m}(2 \cdot q_k)^p}\right)^{1/p}. \quad (15)$$

Based on these formulas, we can instantiate the formula for combined disjunctive and conjunctive queries, compatible with negation queries. Because the range of the weights from the AND part of the formula is now from 0 to 1, we do not have to change the OR part of the formula which calculates the disjunction of the subquery weights. Hence, we obtain:

$$score(w_i\ \mathrm{OR}_{(p)}) = \left(\frac{\sum_{k=1}^{n}(w_i)^p}{n}\right)^{1/p}, \quad (16)$$

$$w_i\left(d, q\ \mathrm{AND}_{(p)}\right) = 1 - \left(\frac{\sum_{k=1}^{m}(q_k)^p(q_k - d_k)^p}{\sum_{k=1}^{m}(2 * q_k)^p}\right)^{1/p}, \quad (17)$$

where d is the document vector, q represents the query vector, m is the total number of terms in documents and query, n is the total number of conjunctive clauses, and w_i is the weight of clause i.

6. Implementation

Both the query language and the ranking algorithm are implemented in the Hermes News Portal (HNP), i.e., our implementation of the Hermes news personalization framework, which has been introduced earlier in Section 2. Our implementation of the HGQL-enabled result ranking HNP can be downloaded from `http://people.few.eur.nl/fhogenboom/hermes.html`. The HNP allows

users to formulate queries and execute them on the domain ontology in order to retrieve relevant news items. The HNP application is a stand-alone, Java-based tool which makes use of Semantic Web technologies, and supports extendability through the support for user-created plug-ins (e.g., HGQL-related components). Its internal knowledge base is an expert-created domain ontology with 66 classes, 18 object properties, 11 data properties, and 1173 individuals, represented in OWL [34], offering useful features for information representation in Hermes, e.g., the ability to describe disjoint classes, role cardinality, and role symmetry. Concepts that appear also in the WordNet [11] semantic lexicon have associated the corresponding WordNet synset. For manipulating OWL representations, the Jena [35] library is employed. While populated ontologies are typically queried by the Semantic Web's standard query language SPARQL [13], querying within HNP is performed through tSPARQL queries [4, 5], an extension of SPARQL with time-specific features. The classification of the news articles is done using GATE [36, 37] and synset information made available by the WordNet semantic lexicon.

In order for the HNP plug-ins to operate, a few processing tasks need to be performed first. For this, GATE provides a pipeline consisting of different components, which are in order of usage: *Document Reset*, *ANNIE English Tokenizer*, *ANNIE Gazetteer*, *ANNIE Sentence Splitter*, *ANNIE Part-Of-Speech Tagger*, *Word Sense Disambiguator*, and *OntoGazetteer*. These components clean each document (news item) from annotations, after which text is tokenized into numbers, words, punctuation, etc. Then, the *ANNIE Gazetteer* looks up words from gazetteer lists (i.e., lists with names of, for example, cities, countries, companies, days of the week, world leaders, etc.) for annotation. Subsequently, the pipeline identifies sentences, required for part-of-speech tagging that is performed through a modified version of the Brill tagger [38]. A word sense disambiguation procedure based on the adapted Lesk algorithm [12] implementation made available in [39] is performed to determine the meaning of words in the news text. The WordNet semantic lexicon is used to provide the word senses and their definitions. Last, the *OntoGazetteer* component annotates tokens with ontological concepts.

In the default configuration of the HNP, the user searches for news by selecting the concepts of interest by using a graph visualization of the knowledge base (depicted in Figure 6), from which the user can choose either single concepts or concepts which are related to a certain concept. These concepts are stored in a search graph, which is depicted in Figure 7. After selecting all concepts of interest, news can be queried using some additional time constraints, if desired. For querying, the selected concepts are interpreted as a disjunctive set of concepts, favoring results with many of the requested concepts. The HNP presents a ranking of news items along with their computed relevance scores. Concepts of interest are highlighted in the result list, as depicted in Figure 8.

Within the HNP, user profiles can be created as well, which can store search queries. In future releases, these user profiles can be exploited by allowing the user to assign importance weights to the selected concepts of interest. At the moment, the weights given by the users are not considered in the implementation.

Our HGQL plug-in for the HNP represents ontologies using a graph visualization generated through the Prefuse [40] Java API, while the OWL2Prefuse [41] Java API is used for the visualization of the OWL knowledge base. Also, queries are transformed to SPARQL, which are executed by means of ARQ [42]. Additionally, SPARQL/Update statements supported by the ARQ Java API are used in order to update ontologies. The conversion of queries to DNF is done by employing the Orbital library [43].

In the HGQL tab, depicted in Figure 9, the user is able to build queries using the Boolean operators AND, OR, and NOT. HGQL also offers the ability to create pattern queries, consisting of a subject, a predicate, and an object. The user can add concepts and relations on the fly to the HGQL query canvas by using searchable dropdown boxes. Unknown concepts and relations are added through dedicated buttons. The text match fields allow the user to search for concepts and relations in the knowledge base. Intra-query operators are added by clicking the AND, OR, and NOT buttons, whereas logical edges are created through the designated button for connecting selected concepts or relations (the order of selection determines the edge direction). Selected items (and their associated edges) can be deleted by a click of a button.

Users can select items in the HGQL drawing by clicking them while pressing the control key (they will turn red if selected). Clicking them again will deselect the items. Note that an item can be dragged around by click-hold-moving it, and

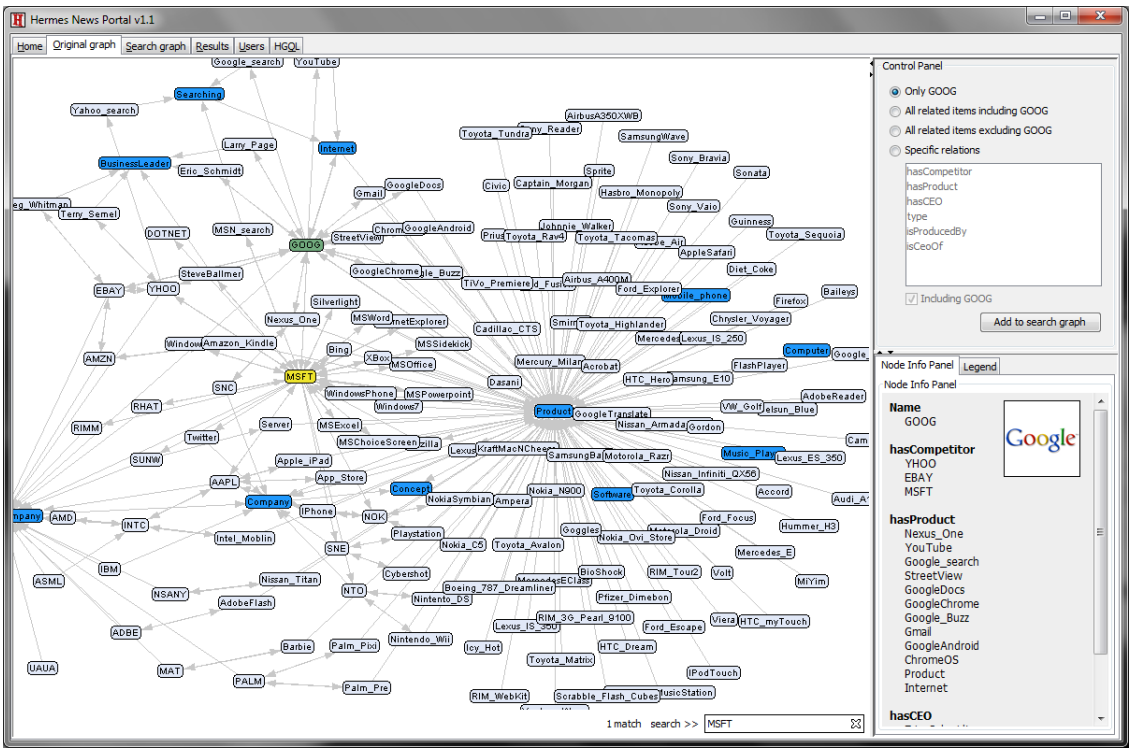

Figure 6: Selecting concepts for a search query in the HNP.

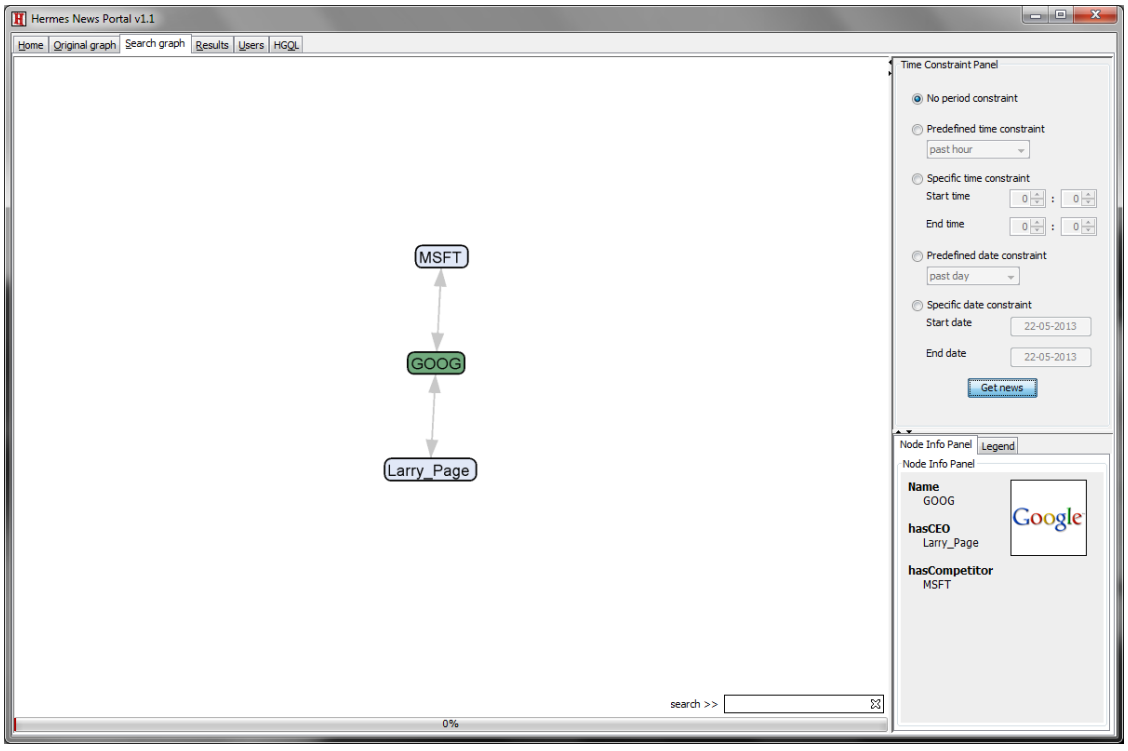

Figure 7: A search graph in the HNP.

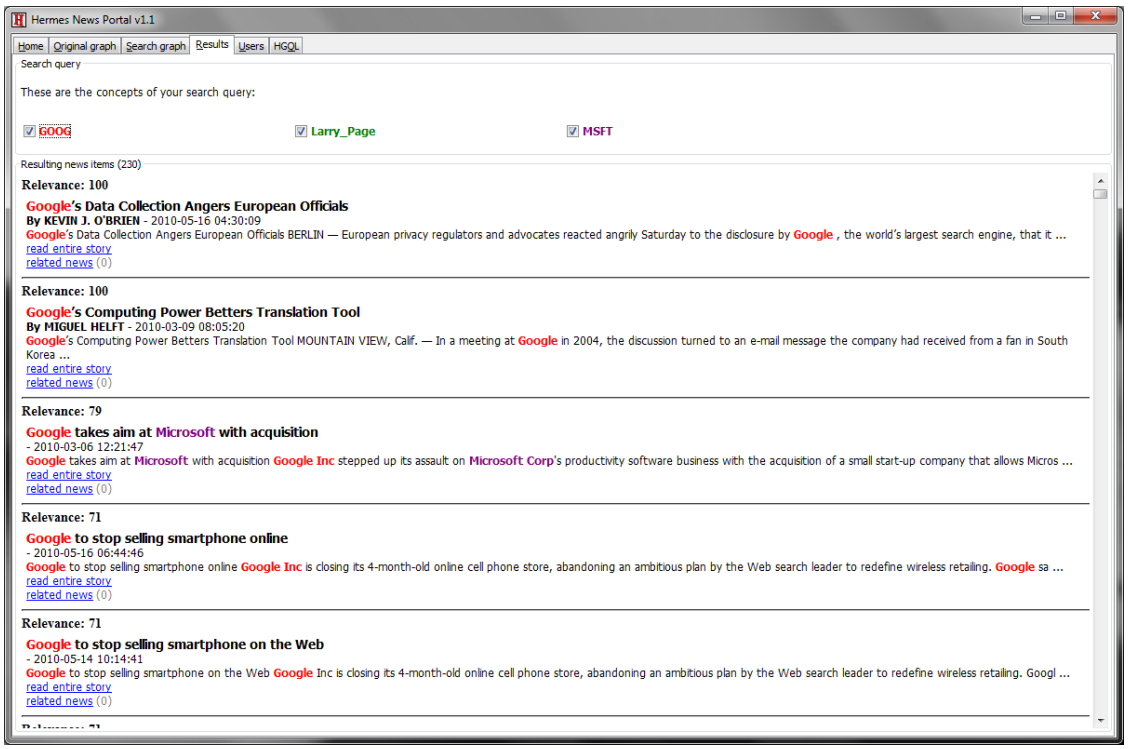

Figure 8: Ranked query results in the HNP.

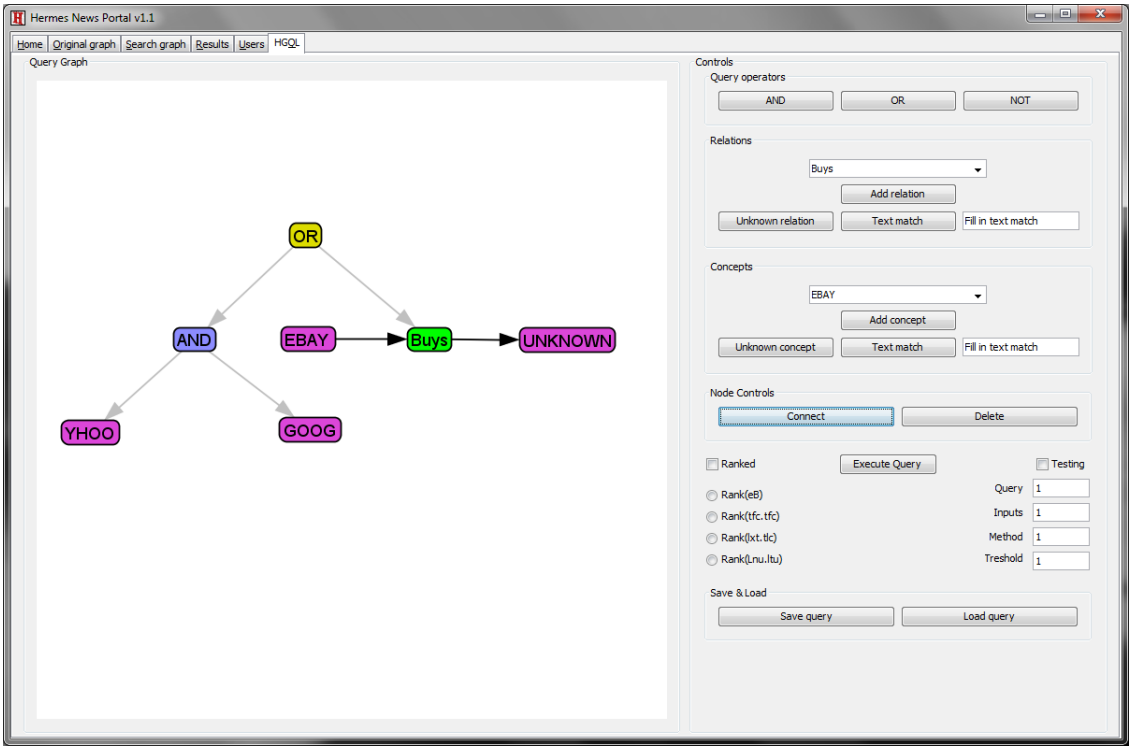

Figure 9: An overview of the HGQL interface.

panning the drawing is supported through click-hold-moving the canvas. Zooming can be done by scrolling up (zoom out) or down (zoom in). Edges can be changed to logical or pattern edges by right-clicking them and by selecting the appropriate type. Operators can be changed into inter-query or intra-query operators by using their right-click menu. Queries can be saved or loaded using the buttons at the bottom of the screen.

Additionally, users are able to specify a weighting algorithm using radio buttons. A checkbox can be ticked if ranked results are desired and testing parameters can be fine-tuned. Once the query is finished and execution preferences have been specified, the user can search the news repository by clicking the 'Execute Query' button. After query validation (using the rules described in Section 4.3) and an HGQL to tSPARQL conversion, the query is executed. After query execution, the system switches to the Results tab. When converting an HGQL query to tSPARQL, HNP splits the graph in order to produce a list of subqueries which can be processed individually. For every subquery, a tSPARQL query is produced, executed, and – depending on the inter-query operators used – added to the list of results, which is presented to the user after all sub-queries have been processed.

We illustrate the translation from HGQL to tSPARQL by means of two examples, where we give the HGQL queries and their tSPARQL equivalents.

If we consider the concepts-only HGQL query depicted in Figure 10, which fetches anything related to companies and either the United States or Japan, we obtain the following tSPARQL equivalent:

```
PREFIX news: <http://www.hermes.com/news.owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/
              22-rdf-syntax-ns#>
PREFIX kb:   <http://www.hermes.com/
              knowledgebase.owl#>
SELECT DISTINCT ?news
WHERE
{
   ?news rdf:type news:News.
   ?news news:relation ?relation1.
   ?news news:relation ?relation2.
   ?relation1 news:relatedTo ?concept1.
   ?relation2 news:relatedTo ?concept2.
   FILTER(?concept1 = kb:Company &&
         (?concept2 = kb:Japan ||
          ?concept2 = kb:United_States))
}
```

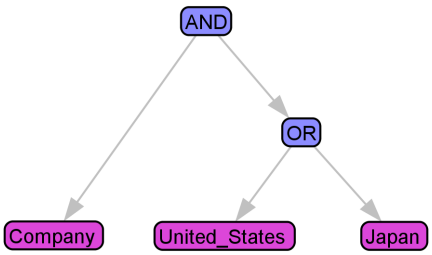

Figure 10: A concepts-only query in HGQL.

The 'PREFIX' commands allow for the use of namespaces. 'SELECT DISTINCT ?news' indicates that only unique news items will be selected as a result, which is desirable in a news recommender. The 'WHERE' clause specifies the conditions for the news items to be selected. The most important part of this translation is the 'FILTER' statement; this allows the query to filter the results for certain conditions. In this case we want one concept to be *Company* and the other either *Japan* or *United_States*. Two examples of returned news items from our news ontology are:

Panasonic Slumps to $4 Billion Yearly Loss
AP - Panasonic Corp. slumped deep into the red last fiscal year, joining the expanding club of big Japanese brands shellshocked by their rapid descent from cash cow to money loser. Panasonic reports earnings based on U.S. accounting standards. In trading Friday, shares of Panasonic jumped 4.8% to 1,455 yen on the TokyoStock Exchange, outpacing the benchmark Nikkei 225 index's 1.9% rise. The results were released after trading closed.

Advertising: A Tech Company's Campaign to Burnish its Brand
Intel's new campaign beginning Monday in the U.S. is the company's first to focus on the amusingly weird, technology-focused culture of Intel and celebrates the company's role in the future, rather than the present. The tagline is "Sponsors of Tomorrow," and the ads highlight achievements of Intel engineers in a humorous way. The campaign is Intel's first that focuses on its brand rather than its products and it is Intel's most expensive campaign since 2006.

Figure 11: A triple-based pattern query in HGQL.

Our second example is depicted in Figure 11, which is a triple-based pattern query. For this type of query, a new class was added to the news ontology, i.e., 'Pattern', which holds the subject, object, and predicate of a pattern.

When this HGQL query is translated to tSPARQL, this results in the following query:

```
PREFIX news: <http://www.hermes.com/news.owl#>
PREFIX rdf:  <http://www.w3.org/1999/02/
              22-rdf-syntax-ns#>
PREFIX kb:   <http://www.hermes.com/
              knowledgebase.owl#>
SELECT DISTINCT ?news
WHERE
{
   ?news rdf:type news:News.
   ?news news:relation ?relation.
   ?relation news:relatedTo ?pattern.
   ?pattern news:subject ?subject.
   ?patern news:predicate ?predicate.
   ?pattern news:object ?object.
   FILTER(?subject = kb:EBAY &&
          ?predicate = kb:Buys &&
          ?object = kb:PayPal)
}
```

Again distinct news items are returned, but this time only the relations that relate to a pattern are selected. For these items, the subject, predicate, and object are selected and compared against the subject, predicate and object from the query. As can be seen this filter has conjunctive semantics (&&), because the pattern has to match on all three fields. An example of a returned news item is:

EBay to Buy its Rival PayPal
The online auction giant eBay announced plans today to acquire PayPal, a rapidly growing start-up that lets people make payments via e-mail. The move will let eBay take a bigger share of many sales made on its own site, and to expand its business to earn money on tens of thousands of transactions made elsewhere on the Internet.

7. Evaluation

In this section we evaluate the HGQL graphical query language and the proposed ranking algorithms. First, we discuss the experimental setup, after which the experimental results are presented.

7.1. Experimental Setup

For our experiments, we make use of a database of 927 news items about various subjects in our domain which is of financial nature. This database is contained in the implementation available at `http://people.few.eur.nl/fhogenboom/hermes.html`. Moreover, the expert-created OWL ontology mentioned earlier serves as a basis for concepts and relations. In order to evaluate our query language and the performance of the proposed ranking algorithms, we consider 10 different queries, as depicted in Table 1.

We evaluate the graphical query language both on quantitative and qualitative aspects against a text-based alternative, i.e., SPARQL. First, we let a small group of 4 test users with a background in Computer Science and with experience in query languages create the 10 queries described in Table 1 both in SPARQL and in HGQL and we measure their creation times. Subsequently, we ask the users about their experiences with both languages and determine their preferences. The users are asked to rate the languages on a 5-point Likert scale with respect to their clarity, preciseness, conciseness, ease of use, intuitiveness, expressiveness, interpretability, insensitivity for mistakes, and on their confidence in the generated results. Additionally, the users are encouraged to rate the implementation of HGQL in terms of clarity, ease of use, intuitiveness, and efficiency.

The proposed ranking algorithm, i.e., the extended Boolean model (Rank ($e\mathbb{B}$)), is also evaluated based on the query list (displayed in Table 1) using a small but different group of test users. We consulted 5 test users who have a Computer Science background and who are familiar with Semantic Web technologies and Information Retrieval techniques in order to create a golden standard using a majority voting scheme. As the query topics are rather general, no domain experts are required, and the choice of test users yields a high quality data set due to their former experiences in annotating documents. It should be noted that while we have used only simple concepts in our queries, we do not loose in generality for our experiments, because complex concepts are merely sequences of three simple concepts that behave as any other concept.

The ranking algorithm is compared to 3 other methods for document and query weight calculation, i.e., the traditional TF-IDF weights (Rank (tfc.tfc)), the TF-IDF model with logarithmic

ID	Query
Q1	(Microsoft ∨ (Yahoo ∧ Google)) ∧ ¬United States
Q2	(Software ∨ DOTNET ∨ Programming ∨ Networking) ∧ (Facebook ∨ Twitter ∨ Yahoo)
Q3	(Steve Ballmer ∨ Steve Jobs ∨ Meg Whitman ∨ Terry Semel ∨ Bruce Chizen ∨ Eric Schmidt ∨ Larry Page) ∧ (Google ∨ Microsoft ∨ Yahoo ∨ Apple)
Q4	European Central Bank ∨ BankOfAmerica ∨ (Europe ∧ NASDAQ ∧ ¬Google) ∧ ¬Software
Q5	(AOL ∧ ¬United Kingdom) ∨ ((Microsoft ∨ Ebay) ∧ Google ∧ ¬iPhone ∧ ¬Adobe)
Q6	(Microprocessor ∨ Networking ∨ Computer hardware) ∧ ¬(United States ∨ Google ∨ Apple ∨ Microsoft)
Q7	(Google ∧ Microsoft ∧ ¬(Computer ∨ Europe)) ∨ (Ebay ∧ ¬United States)
Q8	(Europe ∨ United States) ∧ (Saab ∨ Fiat ∨ Volkswagen ∨ Audi ∨ MercedesBenz ∨ Mercedes ∨ Lexus ∨ Ford ∨ Cadillac) ∧ ¬Africa
Q9	(Bank ∨ Federal Bank ∨ Commercial Bank) ∧ New York Stock Exchange ∧ ¬Asia ∧ ¬Europe
Q10	(TiVo ∨ Adobe ∨ RedHat ∨ Sun ∨ Dell ∨ Cisco ∨ Oracle) ∧ (Microsoft ∨ Apple) ∧ ¬iPhone

Table 1: Evaluation queries.

weights (Rank (lxc.ltc)) as given in Equations (6) and (7), and the TF-IDF model with a combination of the document length and the average document length for length normalization (Rank (Lnu.ltu)), as given in Equations (8) and (9). This way, we compare our proposed method against a classic model (TF-IDF) as a baseline reference, and 2 extensions from the literature that have proven to outperform TF-IDF [28, 30]. The calculations for the different possibilities of concept presence in documents and queries are shown in Table 2.

The ranking algorithms are evaluated by means of two measures. The first measure is the precision for the first 10 documents in our results list. For n different queries we calculate the number of relevant news items in the top ten ranked results. This is referred to as the 'Mean Precision @ 10' (MP@10), and is computed as

$$MP@10 = \frac{1}{n}\sum_{i=1}^{n}\frac{n_i}{10}\,, \tag{18}$$

where n is the number of queries and n_i is the number of relevant documents in the first 10 results of the ranked list for query i.

The second measure that is employed for evaluating the ranking algorithms is the Mean Average Precision (MAP), which provides a single-figure measure of quality across all recall levels. The Average Precision (AP) is the average of the precision values obtained for the set of top k documents in the search results before each relevant document is retrieved for a certain query. We take the mean of this value from n different queries. The MAP can be computed as follows:

$$MAP = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{m_i}\sum_{k=1}^{m_j}\frac{k}{r_{d_{m_k}}}\,, \tag{19}$$

where $\{d_1, ..., d_{m_i}\}$ is the set of relevant documents for query i, m_i represents the number of relevant documents for query i, and $r_{d_{m_k}}$ denotes the rank of document d_{m_k} in the ranked list.

Next, we analyze the precision/recall graph averaged over the different queries. Here, for fixed recall values, we determine the precision for each of the ranking algorithms. Significance of these results is assessed through a one-tailed two-sample paired Student t-test, where we measure whether the one

	Document Weight		Query Weight		
	Concept present	Concept absent	Concept present	Concept absent	Concept negated
Rank($e\mathbb{B}$)	1	-1	1	0	-1
Rank(tfc.tfc)	tfc.tfc	-1	tfc.tfc	0	-1 × tfc.tfc
Rank(lxc.ltc)	lxc.ltc	-1	lxc.ltc	0	-1 × lxc.ltc
Rank(Lnu.ltu)	Lnu.ltu	-1	Lnu.ltu	0	-1 × Lnu.ltu

Table 2: Document weight calculation.

	Language	
Criteria	SPARQL	HGQL
Clear	3	5
Precise	5	5
Concise	3	4
Easy to use	3	4
Intuitive	3	4
Expressive	4	4
Quickly interpretable	3	5
Insensitive for mistakes	2	4
Result confidence	3	4
Overall	3	4

Table 3: Qualitative evaluation of SPARQL and HGQL using a 5-point Likert scale, where 1 represents disagree and 5 represents agree.

	Tool
Criteria	HGQL
Clear	3
Easy to use	2
Intuitive	3
Efficient	3
Overall	3

Table 4: Qualitative evaluation of HGQL implementation in the HNP using a 5-point Likert scale, where 1 represents disagree and 5 represents agree.

method has a larger mean precision than the other, using a significance level α of 0.05.

Last, we analyze the scalability of our approach by measuring the average processing times (in milliseconds) for the evaluation queries (excluding indexing). News indexing is not relevant here, as this can be done before the user poses his/her query. We distinguish between the average processing time for validating the HGQL queries, the average conversion time from valid HGQL queries to tSPARQL queries, and the average time it takes the software to execute the generated tSPARQL queries and fetching the results. Execution times are measured on a user system running an Intel Core i7-3770 CPU at stock speed (3.40GHz) using 16.0GB of RAM.

7.2. Experimental Results

When comparing HGQL against its text-based alternative, SPARQL, we asked the user to rate both languages with respect to various criteria. The results are displayed in Table 3. Overall, the test users clearly favour HGQL over SPARQL. According to the users, HGQL is much clearer and easier to interpret. Also, the language is less sensitive for mistakes, resulting in users having a higher confidence in the returned results. With respect to expressiveness and preciseness, the users make no distinction between SPARQL and HGQL. Last, when it comes to conciseness, ease of use, intuitiveness, HGQL performs slightly better than SPARQL.

After constructing their queries, when asked about their preference, 3 out of 4 users indicated they preferred using HGQL over SPARQL, whereas 1 user did not have a specific preference for either of the languages, mainly due to the HGQL implementation. As shown in Table 4, the users are less enthusiastic about the implementation. Especially the button-oriented design makes the tool less easy to use, for instance when connecting nodes. On the other hand, searching and adding concepts and relations was intuitive and easy for all users.

	Time (seconds)	
Query	SPARQL	HGQL
Q1	449	155
Q2	170	125
Q3	229	165
Q4	283	194
Q5	220	199
Q6	189	133
Q7	171	166
Q8	202	175
Q9	144	112
Q10	150	146

Table 5: Creation times in seconds per query for their SPARQL and HGQL variants.

Considering the creation times, it is evident that creating HGQL queries with our current implementation takes less time than writing their equivalent SPARQL queries. Table 5 shows the creation times in seconds, and illustrates that for all queries, HGQL outperforms SPARQL. On average, the performance gain is 29%.

The results of the ranking algorithms are depicted in Table 6. This table demonstrates that the extended Boolean model performs best on both measures. The performance of Rank(lxc.ltc) is a bit lower than the measured performance of the extended Boolean model. Rank(tfc.tfc) and Rank(Lnu.ltu) have the lowest scores, with the latter performing slightly better than the former.

Figure 12 shows the 11-point precision/recall graph averaged over the 10 given queries. This

Measure	Rank($e\mathbb{B}$)	Rank(tfc.tfc)	Rank(lxc.ltc)	Rank(Lnu.ltu)
MP@10	0.850	0.470	0.730	0.480
MAP	0.874	0.467	0.694	0.572

Table 6: Evaluation results using user list benchmark.

	Rank($e\mathbb{B}$)	Rank(tfc.tfc)	Rank(lxc.ltc)	Rank(Lnu.ltu)
Rank($e\mathbb{B}$)		1.00	1.00	1.00
Rank(tfc.tfc)	0.00		0.00	0.00
Rank(lxc.ltc)	0.00	1.00		0.94
Rank(Lnu.ltu)	0.00	1.00	0.06	

Table 7: One-tailed two-sample paired Student t-test p-values for the precision across all recall values for the Rank($e\mathbb{B}$), Rank(tfc.tfc), Rank(lxc.ltc), and Rank(Lnu.ltu) ranking algorithms ($H_0 : \mu_{column} = \mu_{row}$, $H_1 : \mu_{column} > \mu_{row}$, $\alpha = 0.05$).

graph underlines that the extended Boolean model performs best across all recall levels. From the term weighting methods, Rank(lxc.ltc) performs best for most recall levels, while Rank(Lnu.ltu) performs best on high recall levels. Rank(tfc.tfc) performs poorly on all recall levels, which matches our expectations (as suggested in the literature).

When assessing the significance of precision on all recall values, we obtain the p-values as denoted in Table 7. From these values, we can deduce that on average, the extended Boolean model significantly outperforms the other ranking models. Also, the TF-IDF model with logarithmic weights (Rank (lxc.ltc)) significantly outperforms the basic TF-IDF model, yet it does not significantly outperform the TF-IDF model with a combination of the document length and the average document length for length normalization (Rank (Lnu.ltu)). Last, Rank (Lnu.ltu) significantly outperforms the basic TF-IDF model, but fails to significantly outperform the other ranking algorithms.

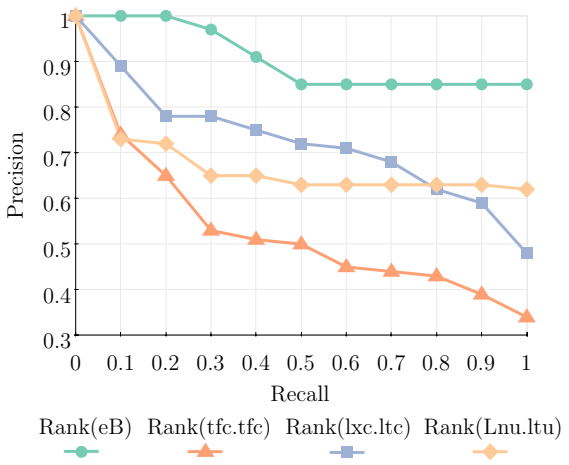

Figure 12: The 11-point precision/recall graph with the user list as benchmark.

We have used different vector space model weighting schemes for our ranking algorithm. The previous results [28] that showed that it is better to map the document and query vectors differently in the vector space have been confirmed here, because both Rank(lxc.ltc) and Rank(Lnu.ltu) perform better than the basic TF-IDF model, Rank(tfc.tfc), where the query and document vectors are mapped equally. However, our experiments do not confirm earlier results regarding algorithms that use a combination of the document length and the average document length for normalization compared to algorithms using cosine normalization [30], as Rank(Lnu.ltu) does not outperform Rank(lxc.ltc).

Last, in the 10 evaluated queries, the average processing time (excluding indexing) was 0.2ms for HGQL validation, 3.4ms for HGQL to tSPARQL conversion, and 2826.4ms for tSPARQL execution. From this, we can conclude that the process of validating and converting HGQL queries is not computationally intensive, and hence is expected to be scalable to larger systems. In contrast, the tSPARQL execution times can be improved by employing other, more scalable SPARQL engines, but this is outside the scope of our paper. A further evaluation of the scalability is subject to future work.

8. Conclusion

In this paper we have presented, implemented, and evaluated the Hermes Graphical Query Language (HGQL), which is a query language for news items that makes use of graphical elements. The

language is intended for use in the Hermes news personalization framework, which is implemented as a plug-in extendable Web-based tool, the Hermes News Portal (HNP). The HGQL plug-in in the HNP enables users to define structured queries without requiring any knowledge of the standard (textual) query language, SPARQL, and makes use of several building blocks, i.e., nodes (representing concepts, relations, operators, and wildcards) and edges (interconnecting nodes). The language distinguishes between queries consisting of only concepts and logical operators, triple-based query patterns, and combinations of these two types. HGQL queries result in either a list of news items that completely match the query (the Boolean model), or a ranked list of news items based on their relevance to the specified query.

Our qualitative and quantitative evaluation of the query language on a small group of users provided various insights. First, overall, the users preferred HGQL over its text-based alternative, SPARQL, mainly because of its clarity and interpretability. The users were more moderate in their judgement of the HGQL implementation, which was not always easy to use. However, when comparing query creation times, composing HGQL queries still took roughly 29% less time than writing their SPARQL equivalents.

Moreover, we addressed the performance of a proposed ranking algorithm supporting negation operators for queries consisting of conjunctive and disjunctive operators. The ranking algorithm of HGQL is based on a combination of the different formulas for disjunctive and conjunctive queries of the p-norm extended Boolean model. The negation operator is addressed by using negative weights for query and document terms. Four different vector space model weighting algorithms have been tested in this study, i.e., the extended Boolean model Rank($e\mathbb{B}$), which uses document weights of 1, 0 and -1, and three different TF-IDF weighting algorithms, Rank(tfc.tfc), Rank(lxc.ltc), and Rank(Lnu.ltu).

Our study showed that the extended Boolean model is performing best with a Mean Precision at 10 (MP@10) of 0.85 and a Mean Average Precision (MAP) of 0.874. The lxc.ltc algorithm provides the second best results with an MP@10 of 0.730 and a MAP of 0.694. The Lnu.ltu algorithm and the tfc.tfc algorithm perform relatively poor with an MP@10 of 0.480 and 0.470, respectively, and an MAP of 0.572 and 0.467, respectively.

In future work, we would like to extend the ranking algorithm by employing user-defined weights for query concepts (already supported at the query language level), by treating them in conjunction with query weights (by for example scaling the original weights with the user-defined weights). Another research direction that we would like to pursue is to fine-tune the HGQL implementation and to perform a user-based evaluation on a larger scale and using a wider range of prospected user groups with respect to the ease-of-use of the query language. Last, we would like to assess the scalability of our approach, and more specifically the query times of our approach on larger ontologies on the one hand, and for query sets of different complexities on the other hand.

Acknowledgment

The authors are partially sponsored by the NWO Physical Sciences Free Competition project 612.001.009: Financial Events Recognition in News for Algorithmic Trading (FERNAT) and the Dutch national program COMMIT.

References

[1] F. Frasincar, J. Borsje, F. Hogenboom, E-Business Applications for Product Development and Competitive Growth: Emerging Technologies, IGI Global, 2011, Ch. Personalizing News Services Using Semantic Web Technologies, pp. 261–289.

[2] W. IJntema, F. Goossen, F. Frasincar, F. Hogenboom, Ontology-Based News Recommendation, in: F. Daniel, L. M. L. Delcambre, F. Fotouhi, I. Garrigós, G. Guerrini, J.-N. Mazón, M. Mesiti, S. Müller-Feuerstein, J. Trujillo, T. M. Truta, B. Volz, E. Waller, L. Xiong, E. Zimányi (Eds.), International Workshop on Business intelligencE and the WEB (BEWEB 2010) at 13th International Conference on Extending Database Technology and 13th International Conference on Database Theory (EDBT/ICDT 2010), Vol. 426 of ACM International Conference Proceeding Series, ACM, 2010.

[3] L. Zheng, L. Li, W. Hong, T. Li, PENETRATE: Personalized News Recommendation Using Ensemble Hierarchical Clustering, Expert Systems with Applications 40 (6) (2013) 2127–2136.

[4] F. Frasincar, J. Borsje, L. Levering, A Semantic Web-Based Approach for Building Personalized News Services, International Journal of E-Business Research 5 (3) (2009) 35–53.

[5] K. Schouten, P. Ruijgrok, J. Borsje, F. Frasincar, L. Levering, F. Hogenboom, A Semantic Web-Based Approach for Personalizing News, in: M. J. Palakal, C. Hung (Eds.), 25th Symposium On Applied Computing (SAC 2010), Web Technologies Track, ACM, 2010, pp. 854–861.

[6] F. Frasincar, W. IJntema, F. Goossen, F. Hogenboom, Business Intelligence Applications and the Web: Models, Systems and Technologies, IGI Global, 2011, Ch. A Semantic Approach for News Recommendation, pp. 102–121.

[7] W. IJntema, J. Sangers, F. Hogenboom, F. Frasincar, A Lexico-Semantic Pattern Language for Learning Ontology Instances from Text, Journal of Web Semantics: Science, Services and Agents on the World Wide Web 15 (1) (2012) 37–50.

[8] F. Hogenboom, V. Milea, F. Frasincar, U. Kaymak, Emergent Web Intelligence: Advanced Information Retrieval, Advanced Information and Knowledge Processing, Springer London, 2010, Ch. RDF-GL: A SPARQL-Based Graphical Query Language for RDF, pp. 87–116.

[9] A. Verheij, A. Kleijn, F. Frasincar, D. Vandic, F. Hogenboom, Querying and Ranking News Items in the Hermes Framework, in: S. Ossowski, P. Lecca (Eds.), 27th Symposium on Applied Computing (SAC 2012), Web Technologies Track, ACM, 2012, pp. 672–679.

[10] G. Salton, E. A. Fox, H. Wu, Extended Boolean Information Retrieval, Communications of the ACM 26 (11) (1983) 1022–1036.

[11] C. Fellbaum, WordNet: An Electronic Lexical Database, MIT Press, 1998.

[12] S. Banerjee, T. Pedersen, An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet, in: A. F. Gelbukh (Ed.), 4th International Conference on Computational Linguistics and Intelligent Text Processing (CICLING 2002), Springer-Verlag, 2002, pp. 136–145.

[13] E. Prud'hommeaux, A. Seaborne, SPARQL Query Language for RDF - W3C Recommendation 15 January 2008, From: `http://www.w3.org/TR/rdf-sparql-query/` (2008).

[14] A. Harth, S. R. Kruk, S. Decker, Graphical Representation of RDF Queries, in: 15th International Conference on World Wide Web (WWW 2006), ACM Press, New York, NY, USA, 2006, pp. 859–860.

[15] A. Fadhil, V. Haarslev, GLOO: A Graphical Query Language for OWL Ontologies, in: OWL: Experience and Directions (OWLED 2006), CEUR-WS, 2006.

[16] M. Angelaccio, T. Catarci, G. Santucci, QBD*: A Graphical Query Language with Recursion, IEEE Transactions on Software Engineering 16 (10) (1990) 1150–1163.

[17] A. Michard, Graphical Presentation of Boolean Expressions in a Database Query Language: Design Notes and an Ergonomic Evaluation, Behaviour & Information Technology 1 (3) (1982) 279–288.

[18] G. G. Chowdhury, Introduction to Modern Information Retrieval, John Wiley and Sons, 1998.

[19] G. Salton, M. J. McGill, Introduction to Modern Information Retrieval, McGraw-Hill, 1983.

[20] H. P. Luhn, A Statistical Approach to the Mechanized Encoding and Searching of Literary Information, IBM Journal of Research and Development 1 (4) (1957) 309–317.

[21] J. Hynek, Document Classification in a Digital Library, Tech. rep., University of West Bohemia in Pilsen, from: `http://www.kiv.zcu.cz/vyzkum/publikace/technicke-zpravy/2002/tr-2002-04.pdf` (2002).

[22] D. Widdows, Orthogonal Negation in Vector Spaces for Modelling Word-Meanining and Document Retrieval, in: 41st Annual Meeting on Association for Computational Linguistics (ACL 2003), Association for Computational Linguistics, 2003, pp. 136–143.

[23] C. P. Paice, Soft Evaluation of Boolean Search Queries in Information Retrieval Systems, Information Technology: Research and Development 3 (1) (1984) 33–42.

[24] K. Anyanwu, A. Maduko, A. Sheth, SemRank: Ranking Complex Relationship Search Results on the Semantic Web, in: A. Ellis, T. Hagino (Eds.), 14th International conference on World Wide Web (WWW 2005), ACM, 2005, pp. 117–127.

[25] N. Stojanovic, R. Studer, L. Stojanovic, An Approach for the Ranking of Query Results in the Semantic Web, in: D. Fensel, K. P. Sycara, J. Mylopoulos (Eds.), 2nd International Semantic Web Conference (ISWC 2003), Vol. 2870 of Lecture Notes in Computer Science, Springer, 2003, pp. 500–516.

[26] D. Hiemstra, Using Language Models for Information Retrieval, CTIT Ph.D. Thesis Series No. 01-32, University of Twente, from: `http://wwwhome.cs.utwente.nl/~hiemstra/papers/thesis.pdf` (2001).

[27] G. Salton, C. S. Yang, On the Specification of Term Values in Automatic Indexing, Journal of Documentation 29 (4) (1973) 351–372.

[28] G. Salton, C. Buckley, Term-Weighting Approaches in Automatic Text Retrieval, Information Processing and Management: an International Journal 24 (5) (1988) 513–523.

[29] C. Buckley, J. Allan, G. Salton, Automatic Routing and Retrieval Using Smart: TREC-2, Information Processing and Management 31 (3) (1995) 315–326.

[30] A. Singhal, C. Buckley, M. Mitra, Pivoted Document Length Normalization, Information Processing and Management 32 (5) (1996) 619–633.

[31] F. Oroumchian, A. Aleahmad, P. Hakimian, F. Mahdikhani, N-Gram and Local Context Analysis for Persian Text Retrieval, in: 9th International Symposium on Signal Processing and its Applications (ISSPA 2007), IEEE Computer Society, 2007, pp. 1–4.

[32] H. K. Büning, T. Lettmann, Propositional Logic: Deduction and Algorithms, Cambridge University Press, 1999.

[33] J. E. Whitesitt, Introduction to Boolean Algebras, Courier Dover Publications, 1995.

[34] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein, OWL Web Ontology Language Reference - W3C Recommendation 10 February 2004, From: `http://www.w3.org/TR/owl-ref/` (2004).

[35] Apache Software Foundation, Apache Jena - A Semantic Web Framework for Java, from: `http://jena.sourceforge.net/` (2012).

[36] H. Cunningham, GATE, a General Architecture for Text Engineering, Computers and the Humanities 36 (2) (2002) 223–254.

[37] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications, in: 40th Anniversary Meeting of the Association for Computational Linguistics (ACL 2002), Association for Computational Linguistics, 2002, pp. 168–175.

[38] E. Brill, A Simple Rule-Based Part of Speech Tagger, in: 3rd Conference on Applied Natural Language Processing (ANLP 1992), Association for Computational Linguistics, 1992, pp. 152–155.

[39] A. S. Jensen, N. S. Boss, Textual Similarity: Comparing Texts in Order to Discover How Closely They Discuss the Same Topics, Bachelor's Thesis, Technical University of Denmark (2008).
[40] Berkeley Institute of Design, Prefuse - Information Visualization Toolkit, from: http://prefuse.org/ (2012).
[41] J. Borsje, J. Giles, OWL2Prefuse - An OWL to Prefuse Converter, from: http://owl2prefuse.sourceforge.net/ (2012).
[42] Apache Software Foundation, Apache ARQ - A SPARQL Processor for Jena, from: http://jena.apache.org/documentation/query/ (2012).
[43] A. Platzer, Orbital Library, from: http://symbolaris.com/orbital/ (2012).