INTEGRATION OF YAGO ONTOLOGY
IN THE IQP QUERY CONSTRUCTION SYSTEM
TO SUPPORT EFFICIENT QUERY CONSTRUCTION
OVER A LARGE-SCALE RELATIONAL DATABASE

Von der Fakultät für Elektrotechnik und Informatik
der Gottfried Wilhelm Leibniz Universität Hannover
zur Erlangung des Grades

Master of Science

M.Sc.

Thesis
von

Iryna Oelze

Erstprüfer : Prof. Dr. techn. Wolfgang Nejdl
Zweitprüfer : Prof. Dr. rer. nat. Udo Lipeck
Betreuer : M.Sc. Dipl.-Inf. (FH) Elena Demidova
ABSTRACT

IQP query construction system empowers naïve database users to create their own structured queries in an interactive way, starting from simple keywords and refining the initial query using options automatically suggested by the system. In order to generate such query construction options, IQP makes use of the database internal structures such as database schema and keyword occurrence statistics. The efficiency and usability of IQP was experimentally confirmed for the middle-sized datasets such as IMDB and Lyrics.

Freebase is a large-scale open-world database where users collaboratively create content over an open platform. Deployment of the IQP system on Freebase faces additional challenges as the schema of Freebase is big and the query construction options derived based on the Freebase schema only are not informative enough to enable an efficient query construction process. Recently, IQP was extended to enable more general ontology-based query construction options that summarize the database schema and speed up the query construction process. Currently, the implementation of IQP uses Freebase taxonomy, that includes about 100 Freebase-specific domains and several top-level categories.

In order to further improve efficiency of the query construction process of IQP and to enable deployment of the system over large-scale databases that do not have a pre-defined ontology layer, we make use of YAGO ontology. YAGO can provide a basis to summarize database schema and to create an optimal query construction process for large-scale databases. In the first step, we develop schema mappings between the Freebase dataset and the YAGO ontology. Then, we present algorithms that enable efficient integration of the YAGO ontology in the query construction process of IQP. Moreover, we generalize implementation of IQP to enable integration of generic ontologies. Finally, we evaluate the effect of YAGO integration on the efficiency of the query construction process. Our evaluation results confirm that YAGO-based options improve efficiency of the query construction process.

Keywords: Keyword Search in relational Databases, Incremental Query Construction, YAGO
Table of Contents

List of Figures

1 Introduction

2 Background

  2.1 Keyword Search in Databases

  2.1.1 Data Modeling

  2.1.2 Structural Ambiguity of the Results

  2.1.3 Ranking Strategies

  2.2 Incremental Query Construction

  2.3 Ontology

  2.3.1 WordNet

  2.3.2 Wikipedia

  2.3.3 YAGO

  2.3.4 Freebase

  2.4 Ontology Matching Strategies

  2.4.1 Element-Based Matching

  2.4.2 Structure Similarity

  2.4.3 Instance-Based Techniques

3 Problem Analysis

  3.1 Motivation
3.2 Introduction to the IQP System ........................................... 27
3.3 Problem Statement ......................................................... 30

4 Conceptual Design 33
  4.1 Definitions ............................................................... 33
  4.2 Overview of Preprocessing and Search Steps ...................... 35
  4.3 Schema Matching ....................................................... 36
    4.3.1 Concept Identification ........................................... 37
    4.3.2 Matching .......................................................... 38
    4.3.3 Similarity Score Computation .................................. 39
  4.4 IQP Hierarchy Construction ........................................... 44

5 Implementation 53

6 Evaluation 57
  6.1 Experiment Setup ...................................................... 57
  6.2 Experimental Results ................................................ 58

7 Conclusions 61

Bibliography 63
2.1 Common components of ontology ........................................... 13
2.2 Hierarchical structure of YAGO concepts ............................... 18
2.3 Hierarchical structure of Freebase concepts ............................... 19
2.4 Matching process. Ontologies \( o, o' \), alignments \( A, A' \), matching parameters \( p \), external resources \( r \) ........................................... 20
3.1 Overview of the IQP System .................................................... 28
4.1 Overview of preprocessing and search steps ............................... 36
4.2 Concepts and instances in YAGO ............................................. 37
4.3 Concepts and instances in Freebase ........................................... 38
4.4 Matching of YAGO and Freebase concepts ............................... 39
4.5 Number of Freebase concepts with a given percentage of Wikipedia instances ........................................... 41
4.6 Number of Freebase concepts with a given percentage of Wikipedia and YAGO instances ........................................... 42
4.7 An example of IQP hierarchy for a keyword \textit{king} ........................ 45
4.8 IQP hierarchy structure using YAGO ontology .......................... 46
4.9 Hierarchy construction algorithm ............................................. 49
5.1 Hierarchy class diagram ......................................................... 54
6.1 Number of query construction options in a 1-keyword query set in three different scenarios ........................................... 59
6.2 Number of query construction options in a 2-keyword query set in three different scenarios .......................... 60
6.3 Number of query construction options in a 3-keyword query set in three different scenarios ........................................ 60
With the growth of structured information available on the Web and in online databases, it becomes increasingly difficult for users to find the exact data they seek for. One example of a large-scale online database is Freebase [BEP+08]. Freebase currently contains about 22 millions entities and more than 350 millions facts in about 100 domains [fre]. Freebase data is organized as a relational database with more than 7500 tables, mostly containing textual data. The users of Freebase can collaboratively create, structure and maintain database content. In addition, automatic imports from external data sources such as Wikipedia [Wik], MusicBrainz [mus] and others enable further growth of the data and schema size. Given the amount of the available structured information on the Web, it is extremely important for users to be in a possession of an efficient and effective search function.

On the one hand, a structured query put in a structured query language specifically designed for a database is very expressive as it can describe the users’ information need precisely. To this end, Freebase offers a structured query language named MQL [mql]. However, formulation of a structured query is a challenging task for a naive user as it requires precise knowledge of the database schema and the query language. On the other hand, a keyword query is easy to put and it is more customary for the users. Keyword search allows ordinary users to search for information without any expert knowledge of the database schema. However, as a single keyword can occur in nearly any textural
attribute of a database, the number of possible query interpretations grows sharply with number of textual attributes and the size of the schema. As a consequence, database search applications can encounter difficulties in detecting the desired information quickly and accurately and may return irrelevant or incomplete search results.

A novel system IQP [DZN10, DZN11] is designed to fill the gap between usability of keyword search and expressiveness of database queries. IQP provides a query construction interface, that enables users to create their own structured queries in an interactive way based on the query construction options automatically suggested by the system. If the user accepts or rejects an option, the system can use this information to automatically reduce the interpretation space of the user’s keyword query and suggest new options. The interaction process between the user and IQP continues until the user finds the desired interpretation of the keyword query and retrieves the corresponding search results. Initially, in order to generate query construction options, IQP made use of the database schema and keyword occurrence statistics. Efficiency and usability of IQP was experimentally confirmed for the middle-sized datasets such as IMDB [IMD] and Lyrics [LYMC06].

In a case of a large-scale database such as Freebase the number of query construction options based on the keyword occurrences can rapidly increase. At the same time, these options are not informative enough to enable efficient query construction as each option subsumes only a small portion of the dataset. That’s why querying of such database as Freebase opens additional challenges. A recent extension of IQP enables generation of more general ontology-based query construction options. These options summarize the database schema and speed up the query construction process. Currently, the implementation of IQP uses options derived from the Freebase taxonomy that includes about 100 Freebase-specific domains and several top-level categories. In order to further improve efficiency of the query construction process of IQP and to enable deployment of the system over large-scale databases that are not associated with an ontology a-priori, we make use of YAGO [SKW07].
A YAGO ontology is a lexical resource that contains facts, their relations and categories automatically extracted from Wikipedia [Wik]. The extracted categories from Wikipedia are unified with the concepts of the WordNet thesaurus [Fel98] and arranged into a taxonomic hierarchy. The semantic categories of YAGO can be used to add a hierarchical structure to the database schema and enable an optimal query construction process. For example, if the user issues a keyword query “Munich” and accepts an option “Munich: Geographical Area”, the database search can focus on the tables subsumed by this YAGO category and ignore other keyword occurrences such as e.g. “Munich: Person”.

In this thesis we perform integration of YAGO ontology in the IQP system. First, we define a mapping between the concepts of the YAGO ontology with the Freebase tables. This mapping does not depend on particular query and is performed in an offline pre-processing step. Next, at the query time, we identify Freebase tables that have keyword occurrences and group the corresponding database tables using the YAGO hierarchy. As the taxonomic hierarchy of YAGO doesn’t feature a balanced structure, we propose an algorithm for an efficient construction of the hierarchical tree at the query time. In our experiments we evaluate effect of YAGO integration in the query construction process. To this end we compare efficiency of the query construction process performed with different options: YAGO-based options, Freebase-based options and options generated without an involvement of any ontology. Our evaluation results confirm that YAGO-based options improve the efficiency of the query construction process significantly.

This thesis is organized as follows. Chapter 2 provides background on the related work in database keyword search, incremental query construction, ontologies and schema matching. A motivating example, an overview of the IQP framework and the problem statement are presented in Chapter 3. In Chapter 4 we introduce the basic concepts used in the thesis, our development of the schema mapping between the YAGO ontology and the Freebase database as well as the hierarchy construction algorithm. Chapter 5 specifies the implemented ontology-based extension to IQP. The evaluation of the effect of YAGO
integration on the resulting query construction process is demonstrated in Chapter 6. Finally, Chapter 7 provides a conclusion.
Chapter 2 analyses the problem area and offers an overview of the related work. We start with the description of the existing approaches in the domain of keyword database search in Section 2.1 and demonstrate that they differ in the modeling of the data, result structure and ranking strategies. The new approach of incremental query construction process is presented Section 2.2.

Along with a database, an ontology plays a growing role in the gathering and conceptualization of different objects, concepts, their properties and relationships. How the existing applications profit from this knowledge is showed in Section 2.3. We introduce some of the largest and mostly used knowledge bases available today, such as Wikipedia, WordNet, YAGO and Freebase. Not all ontologies are designed in the same fashion, that’s why in order to exploit all useful information from different resources the matching of data from two or more ontologies is necessary. Section 2.4 gives the deeper insights into this problem.

2.1 Keyword Search in Databases

The continually growing amount of structured information has little use without the availability of an efficient search function. The most relational databases provide users with a full-text search capability restricted to one table attribute and enabling the execution of a simple
query that affects only one data entry of table column. But due to the complexity of the database and its schema, each attempt to put an intricate query becomes a challenging task. One should still have knowledge of the database schema and use a structured query language (e.g. SQL or XQuery) in order to satisfy his information need.

In the age of internet and search engines ordinary users are more acquainted with a keyword search. This kind of search is very intuitive and convenient, as they only need to type some keywords in the search window and become a list with ranked suggestions for their request.

But what happens behind the keyword search process? What is the magic behind the transformation of the simple keyword query into the relevant individual results? The following aspects are important during this process:

1. data modeling
2. structural ambiguity of the results
3. ranking strategies

In order to enable the database search, first the properties and peculiarities of database data and schema are reflected in a data model. In the second step, on the basis of a selected model the number of candidates as possible results are generated. Thereby there exists a vast variety on their structural presentation. As users are interested only in a limited number of relevant results, the disambiguation and ranking of the candidates is done.

2.1.1 Data Modeling

A lot of attempts have been made in order to support keyword search on relational databases. The first of them to mention are the Discover, DBXplorer, BANKS systems [HP02, Agr02, BHN+02], a framework proposed by Hristidis et al. [HGP03] and research done by Yu et al. [LYMC06]. As the keywords can be present in different attributes and tables, these systems were the pioneers generating the matching
rows obtained by joining several tables. For this purpose a database is modeled as a graph with tuples as nodes and edges between the nodes are primary to foreign key relationships. The BANKS system [BHN+02] models database as a directed graph and uses a prestige weight, similar to PageRank, where a node with the most in-pointers gets the highest prestige.

This modeling is sufficient in case of a small or middle-sized relational database. But with the growing amount of information the size of the databases increases and the relationships between the tables become more complex. That’s why IQP and SUITS systems [DZN10, DZZN09] go further and try to reduce the interpretation space of a keyword query by using a set of pre-computed query templates. Three possibilities are proposed to generate them: automatically by exploiting the join paths between the tables within a preset radius, by searching for common patterns in a query log or manually adjusted by a database administrator to meet the needs of an application.

The similar concept is used by Qunits system [NJ09], where the database is represented as a collection of qunits, each of which is an independent piece of information in a database and represents the desired result for some query. In order to extract these concepts there are three sources that can be taken into account: the database itself with its data and schema, query logs and the resources with the structured information such as e.g. Wikipedia.

2.1.2 Structural Ambiguity of the Results

During the search step the suggestions that meet the users’ information need are created. As modelings of the dataset and extraction algorithms used by the research community is very diverse, the structure of results also vary. One possibility is to represent an answer to a query as a minimal Steiner tree connecting tuples that contain keywords of the query [HP02, Agr02, BHN+02, HGP03, LYMC06]. The first three systems support conjunctive keyword queries, while the system proposed by Hristidis et al. [HGP03] also returns the suggestions containing not
necessarily all keywords in the query.

The SQAK system [TL08] is equipped with an analyzer that, based on the query input, produces a set of Candidate Interpretations (a set of attributes from a database) and a builder that builds a Simple Query Network (SQN) tree which represents a structured query. Same as some other approaches SQAK models database as a graph with the nodes representing tables, and edges representing relationships between the tables. SQN is then connected subgraph of this schema graph.

Other approach [NJ09] simplifies a search step to the choosing of the most appropriate qunits and displaying these concepts in a ranked order to a user. In such a way there is no need to search for a match in the database or to determine the appropriate number of results to return.

IQP query construction system [DZN10] uses query templates and during a search phase fills them with keywords. First the smallest templates are used to generate the partial interpretations. Next during the expansion step the more complex queries up to complete ones are created. With the help of such query hierarchy there is no need to generate the whole interpretation space. The appropriate options help to construct the query incrementally and ensure the scalability of the system.

2.1.3 Ranking Strategies

As to the analysis of the results, there exist two approaches. The traditional methods generate and explore all possibilities, whereas the newer strategy is to rank structures. Let us first have a look how the traditional methods proceed. At the beginning tuple trees are extracted as the possible answers for a keyword query. In the next step a ranking strategy is used in order to decide which results are the most relevant ones. The user can then explore the ranked list to identify the desired interpretations.

DBXplorer [Agr02] and Discover [HP02] systems rank join sequences according to the number of joins they require. In such a way
the highest rank is assigned to the matching rows of a single tuple that contain all the keywords. BANKS exploats the links in the directed graph and defines the final relevance score of a tuple tree as a combination of prestige weights of tuples and edge weights that measure how related the two tuples are. As we can see the above systems exploat the peculiarities of a database schema but don’t make use of the IR methods for ranking the results. This aspect was touched in Hristidis et al. [HGP03]. They treat each tuple as a collection of documents and a row is considered to be a document. A state-of-the-art IR ranking function estimates the document relevance to the given query and assigns a score for each text column value in the tuple tree. The final score of the tuple tree is the sum of these scores divided by the number of tuples involved.

Recently the attention of database research community has been attracted by the structured information available in a database and new query ranking approaches emerged [KKR⁺06, ZWX⁺07, ZZDN08, TL08, BTMI10, BDG⁺11]. Given a keyword query, these frameworks convert it into a list of candidate structured queries. Then different strategies are used to decide which query suit user’s information need.

IQP query construction system [DZN10] presents a query interpretation probabilistic model that estimates the rank of a structural query on the basis of the keyword interpretations and the probability of the query template.

The small number of results is one of the central points in the ranking of the structured queries by Suits [DZZN09] as one of the assumptions states that a query should return a reasonable number of results. The heuristic used is similar to inverse document frequency. Another two factors take into account whether most query keywords are matched and how completely each attribute in the predicates is covered by query terms.

The approach in [BTMI10] also identifies query patterns in order to minimize the amount of information that will be presented to the user. But differently from IQP and SUITS, the structure of the generated queries is used here to extract common query patterns which
lead to a compact representation of the queries. Likewise a QUnit system [NJ09] proceeds. During a search phase the large pool of candidate qunits is built. Further a top-ranking set of the most relevant concepts is retrieved based on the utility score of a qunit. The calculation of this score is similar to measuring document relevance in information retrieval. Additionally references like links and joins are resolved to be used for network analysis.

Bergamaschi et al. [BDG+11] propose two different types of weights to measure the probability that a keyword in the query has the intended semantics of a database term: intrinsic and contextual. An intrinsic weight is calculated based on syntactic, semantic and structural factors present in a database or other auxiliary external sources, such as vocabularies, ontologies, etc. A contextual weight is used to measure the same likelihood but considering the mappings of the remaining query keywords.

Also the authors of [LYMC06] concentrate their attention on the effectiveness of the keyword search in a database. They present an improved a new ranking strategy that takes into account such factors as tuple tree size, document length, document frequency normalization and an inter-document weight. The further improvement is to take into account the schema elements and process them differently from row values. As the databases usually describe various entities from real world, a phrase-based model was proposed.

As we can see there are many different ways to guide ordinary users on their way from keyword query input to desired results. The approaches differ not only in the modeling of the database and ranking strategies, but also in a way users interact with the system. And while most existing systems function well on small and middle-sized databases, the scalability and effectiveness problems will arise while dealing with the large-scaled databases with a complex flat schema.
2.2 Incremental Query Construction

Nowadays in the age of web search machines ordinary users let himself be spoilt with the ease of use of a keyword query. Unfortunately, this simplicity doesn’t mean high expressiveness and quality of the retrieved results. The less knowledge about the intent behind a query is provided, the more effort is needed to extract the satisfying information from database. On the contrary, databases are equipped with a powerful query language, that allows to ask even trickiest and unusual questions but is too complex and hard to understand for an ordinary user.

Many efforts have been made in order to combine the high expressiveness of a database query language and the simplicity of a keyword query. Some of them were presented in Section 2.1. The recent approaches make use of the database structure and provide users with a possibilty to refine their information need and actively participate in the information retrieval process.

Users start their information search by entering a keyword query and are then guided through an incremental construction process. First, a system generates a number of elementary partial queries that reflect the semantic meaning of the keywords. Then in each step users accept one of the options and restrict the query interpretation space. In such a way they are able to refine their choice and construct a more complex structured query. In an interactive fashion the system leads them to the relevant semantic query and thus to the desired results.

A QUICK framework [ZZM+09] is used for querying domain-specific ontologies. All what is expected from a user is some basic domain knowledge, he doesn’t need any understanding how a ontology is organized or a query language functions. In order to reduce a query interpretation space, QUICK generates a number of precomputed templates. Each template is a subgraph of the schema graph and represents a possible complete query. At the moment it is limited to acyclic conjunctions of triple patterns.

A search interface of SUITS system [DZZN09] empowers a user with the possibility of incremental query construction in relational databases
without the knowledge of SQL language. It generates the query construction options to help users to find the fragments of a desired structured query. As the number of structured queries for a keyword query grows with the number of keywords in the query and the size of database, the templates and top-k query approach are used, so that the evaluating and executing of all structured queries is not done at once. The number of nodes in templates is estimated due to the analysis of the queries of a real-world query log.

Furthermore, an IQP framework [DZN10] unifies the advantages of the previously mentioned systems. It also provides a formal definition of the process of incremental query construction and presents a probabilistic model to estimate the probabilities of structural query interpretations. The central aim of IQP is to minimize the number of interactions between the system and a user. For this purpose an algorithm for generating an optimal query construction plan based on Information Gain is introduced.

The incremental query construction approach is already a great aid in the reducing of the query interpretation space, but there is still a need for improvement in generation, structuring and ranking of the query construction options in order to make an information search faster, more efficiently and attractive for ordinary users.

2.3 Ontology

An ontology is another way to organize the information about the real objects from various domains, their properties and relationships. Sowa [Sow00] defines an ontology as a collection of names for concept and relation types organized in a partial ordering by the type/subtype relation. An ontology can be viewed as a set of assertions that are meant to model some particular domain [ES07].

Common components of ontologies include:

- Concepts are the main entities of an ontology. For example, in Fig 2.1, Book, Person, Writer are concepts.
2.3 Ontology

- Relations describe the ways the concepts can be related to one another. e.g. \( \text{isA} \), \( \text{writtenBy} \).
- Axioms describe additional constraints on the ontology, e.g. \( \forall x, \text{Autobiography}(x) \Rightarrow \exists y : \text{Person}(y) \land \text{Author}(x) \land \text{topic}(x,y) \).
- Instances are individuals of a concept, for example \( \text{It} \), \( \text{Stephen King} \).
- Attributes are the properties, features, characteristics or parameters that instances and concepts can possess, such as e.g. \( \text{name} \), \( \text{age} \).
- Datatypes specify values of the attributes, e.g. \( \text{String} \) and \( \text{Integer} \) are datatypes.

In various areas of domain knowledge there are different data and conceptual models that can be thought of as ontologies:

- glossaries and data dictionary (compare extraction methods proposed by Navigli et al. [NV08])
- thesauri and taxonomies, e.g. a taxonomic hierarchy of WordNet synsets [Fel98] or the categorical structure of Wikipedia [Wik]
Chapter 2 Background

- metadata and data models, e.g. entity-relationship model, UML class diagram
- formal ontologies, like SUMO [BBEK11] or BFO [GSG04]

Many applications in modern information technology profit from ontological knowledge. It serves as a powerful disambiguator for named entities and aids in detecting them in an unstructured text [Bun06, Coh04]. Also the properties of the objects and the relations between them can be utilized. For example, a SPARK system [LLWZ07] uses ontologies to identify classes, instances and properties for each term in the keyword query. This information is used for the construction of candidate query graphs and choosing the most suggestive ones.

It can also be helpful in overcoming the language barriers and in facilitating cross-language retrieval and machine translation [KCSZ10]. While combining two or more resources of information it is important to detect the different representation of the same object. The rich domain information of an ontology thus helps to resolve this data cleaning problems [CGM05] and plays an important role in information integration [NDH05]. A semantic data model and the constraints over the sets of objects and their relationships serve as valuable recognizers for constraints in free-form service requests [AME07].

Ontologies have also gained popularity among database and web research community as a good support for entity-oriented data integration, data mining and search. A new method for answering relationship queries on two entities is proposed in [LTIT07] and a system for combined full-text and ontology search in [BCSW07]. The NAGA project [KSI*08] presents not only a possibility to extract and organize the information available on the web, but also a query language suitable to answer simple keyword queries of an ordinary user as well as giving an expert an ability to put the more complex ones.

Furthermore, taxonomies are used in numerous other applications for word sense disambiguation and query expansion [LLYM04, BMS07, BV10], document classification [IW06], question answering and information retrieval [FBCC*10].
Next, we introduce the knowledge bases relevant for our research: a
lexicon WordNet with the taxonomic hierarchy of synsets, an encyclopediac
Wikipedia with the hierarchical structure of categories, ontologies
YAGO and Freebase.

2.3.1 WordNet

WordNet [Fel98] is a large lexicon that organizes English verbs, nouns,
adojectives and adverbs by their semantic relations.

The most important relation is synonymy (synset), it denotes the
similarity of meaning and that the words belong to the same concept.
For example, *jump, leap, bound, spring* belong to the synset *move* and
*spring, springtime* to *season, time of year*. The notions in a synset are
grouped such that they are interchangeable in some context.

At the moment WordNet distinguishes among 117 000 synsets and
links them by means of conceptual relations. WordNet can be regared
as an ontology. The synsets represent the concepts, the words grouped
by the concepts are instances.

Verbs and nouns are arranged into the hierarchy with the help of
a subordination/superordination also called hyperonymy/hyponymy or
IsA relation. A hyperonym represents a more general synset and a hy-
ponym a specific one, e.g. *time period* is a superset of *season, time of
year* and *run* is a subset of *travel rapidly*. With these kinds of relations
it is possible to build a noun hierarchy with a root node *entity*. The hy-
peronymy/hyponymy relations arrange the concepts into a taxonomic
hierarchy.

Supplementary, verbs have a troponym relation that expresses a par-
ticular degree of acting, doing something or a happening as in *move-
run-fly*. Another relation between synsets is meronymy/holonomy. It
reflects the part-whole relation between the objects, e.g. *car* has parts
*automobile engine, car seat, air bag* and *leg* is part of *body*. Antonomy
describes the contraposition of some properties of the objects like *early*
is opposite to *late* and is used to organize adjectives and adverbs as the
majority of them derive from adjectives via morphological affixation.
Troponym, meronymy/holonymy and antonomy relations describe the relations between the concepts.

WordNet is free and publicly available for download. The applications appreciate its knowledge, it is a useful tool in diverse fields, for example in information retrieval [LHM08, SJ10, LLYM04], word sense disambiguation [Voo93, KB09], machine translation [OGW07, KCZ02, KZK01].

2.3.2 Wikipedia

Wikipedia [Wik] is an open source multilingual encyclopedia project, it is written and continually updated by volunteers from all over the world. The main elements of Wikipedia can be regarded as the components of a huge ontology:

1. Each article represents one topic (concept) is identifiable by the article title (concept’s attribute).

2. Since 2004 Wikipedia possesses its own category system (folksonomy), created collaboratively by contributors. The majority of Wikipedia articles belong to one or more categories and are grouped together on similar subjects. Encyclopedia users can access the knowledge base by exploring the articles within a category. (the hierarchical structure of concepts)

3. Article can link to other articles using hyperlinks, so that the users can navigate following the links. (relations between the concepts)

4. Articles may contain an infobox - a relational concise summary of an article, a set of attribute/value pairs describing the articles subject. (attributes of a concept)

5. Wikipedia has a redirect system. A Wikipedia redirect page is a virtual link that redirects a user to the correct Wikipedia article.

As an open source project, the entire content of Wikipedia can be easily obtained and is very popular among the research community [MMLW09, NPE]. For example, its categorisation structure and
links to other articles are used whether to build an association web thesaurus [NHN07] or to automatically cross-reference the documents and enrich them with links to the appropriate Wikipedia articles [MW08]. Schnhofen [Sch06] exploits only the titles and categories of Wikipedia articles in order to determine the most characteristic category of a document. There are also tries to create a large-scaled ontology [LCT11, HKVV06] or to use Wikipedias category tree as an ontology [CINZ06, SKW07].

2.3.3 YAGO

The aim of YAGO ontology [SKW07] is to capture the world in terms of entities and statements about them. It is one of the largest resources available today and comprises more than 1.7 million entities and millions of facts about them.

YAGO, in contrast, exploits profound knowledge of Wikipedia to extract named entities like people, films, organizations, spatial data, books, etc. Then it goes further and aims to include not only concrete individual objects but also their classes and relations.

The YAGO model was introduced to express relations between the objects, called facts. A fact is a triple \((x, r, y)\), where \(x\) and \(y\) are entities and \(r\) a relation between them, such as \((Zidane, \text{bornInYear}, 1972)\), \((Einstein, \text{familyNameOf}, Albert\\; Einstein)\). These triples are extracted from the relational Wikipedia categories. An additional relation \textit{means} aids to deal with synonymy like in \((metropolis, \text{means}, city)\). The synsets from WordNet and Wikipedia redirects are used to find the similar objects that describe the same concept. The structural organization of WordNet and category system of Wikipedia help to identify and hold the semantic categories of the objects. E.g. \textit{Stephen King} is a \textit{Writer} (see 2.2). Each concept is treated as an entity and can be connected to another concept by using of \textit{subClassOf} relation, e.g. \((Book, \text{subClassOf}, Object)\). In such a way, the lower classes extracted from Wikipedia are connected to the higher classes extracted from WordNet. All concepts are arranged into a taxonomic hierarchy.
with the root concept *Entity*. The hierarchy can be materialized as an unbalanced hierarchical tree.

![Diagram of YAGO concepts hierarchy](image)

**Figure 2.2**: Hierarchical structure of YAGO concepts

Due to its structure, high precision and large coverage YAGO has already gained the popularity among the research community. It forms a taxonomic backbone to DBpedia [BLK+09] and has been merged with another ontology in the YAGO-SUMO project [dMSP08].

### 2.3.4 Freebase

The Freebase project [BEP+08] aims to conceptualize the world’s knowledge by extracting information from existing sources and then edited by volunteers in a similar fashion as Wikipedia. In such a way, it tries to unify the scalability of structured databases and the advantages of collaborative work. Nowadays it is a huge online database that contains information about almost 22 million entities, each of them having a unique id. The input data for the building of the knowledge base came from numerous sources such as e.g. Wikipedia, Internet Movie Database [IMD].
2.4 Ontology Matching Strategies

The main components of Freebase are topics, that represent physical entities, locations, abstract notions, artistic/media creations, etc. E.g. the *Ulysses*, *James Joyce* are topics (see Figure 2.3). A topic can be assigned to several types, namely *Stephen King* is a *Film writer*, *Film actor*, *Film producer*, *Film director*, *Musical Artist*, *Person*, *Author*. The types are grouped into domains and the domains themselves are associated with the top level categories. In our example, the types *Book* and *Author* belong to the domain *Books* which is associated with the top-level category *Arts & Entertainment*. The hierarchical structure of the Freebase elements has a well-defined structure. Although Freebase has defined a number of topic types so far, but still large amounts of topics lack class membership information.

### Figure 2.3: Hierarchical structure of Freebase concepts

![Hierarchical structure of Freebase concepts](image)

The rapid growth of the semantic web leads to the rise of a number of diverse ontologies. Various applications make use of their profound knowledge. But as the aims and the requirements of these applications may vary, it is often not enough to use only one ontology. Therefore
one have to identify semantically same or similar elements in several different sources of information.

As soon as different knowledge bases that describe the objects from the same domain are identified, it is necessary to find equal parts and establish the semantic correspondences between their elements. This process is called matching. Pavel Shvaiko and Jerome Euzenat [Shv05] state that matching operation (Figure 2.4) takes as input two ontologies $o$ and $o'$. The output of the matching is called alignment $A'$ and represents a set of correspondences between the input sources.

![Figure 2.4: Matching process. Ontologies $o$, $o'$, alignments $A$, $A'$, matching parameters $p$, external resources $r$](image)

Matching process can also use some additional parameters, namely an input alignment $A$, the matching parameters $p$ (e.g. thresholds, boosting parameters) and some external resources $r$ (e.g. WordNet).

To find the correspondence between ontology objects and their relationships, the similarity between concepts need to be calculated. It can be done whether on element or structure level [ES07].

### 2.4.1 Element-Based Matching

Element-level matching techniques analyse concepts in isolation and ignore their relations with other concepts or their instances.

*String similarity* is often used to compare the entity names. A string is treated as a sequence of letters (edit distances) or a set of tokens
2.4 Ontology Matching Strategies

(token-based distances). String distance is a function that maps a pair of strings \( s \) and \( t \) to a real number \( r \). The smaller the value of \( r \) is, the greater is a similarity between the strings \( s \) and \( t \). Some examples of string comparison techniques are Levenstein distance, Monger-Elkan distance function, Jaro metric, Jaccard similarity, SoftTFIDF, N-Gram [CRF03, sim, BKKB05, sec, Kon05].

Before comparing two strings the language-based techniques [DR02, MBR01, GSY04] perform some preprocessing of the words. For example, the frequent words called stopwords, like the, this, any, have little semantic weight and thus can be omitted. Another method is stemming, the reducing of inflected words to their root form, e.g. forms, forming to form.

Another approach [Pat06] is to consider the names of the entities as words of the natural language and use Linguistic resources such as lexicons or domain specific thesauri in order to find the synonymous concepts or polysemy.

2.4.2 Structure Similarity

Structure-level matching techniques put the accent on the relations of the ontology concepts with other concepts or their instances. They are based on such structures as graph nodes, is-a links, property, multiplicity comparisons, etc.

Graph-based techniques [MGMR01, EV04] view an ontology as a labelled graph and analyses the positions of the nodes within it. The hypothesis here is that the nodes from two ontologies are similar if the nodes they are connected to are the same.

Along with purely graph-based techniques, there are other more specific structure-based techniques, for instance, involving trees. Taxonomy-based techniques are based on the is-a hierarchy of the ontology. The intuition behind these techniques is that two concepts are similar if their super or subconcepts are similar. This feature was exploited in [LDKG04, ES04b].
2.4.3 Instance-Based Techniques

The idea behind this approach is to find instances representing the same concepts rather than finding equivalent concepts in different ontologies. Two concepts match if they contain the same set of instances. The individual representations, like a film title, an auto brand, an employee id or url, facilitate this comparison, as they oft coincide across ontologies.

There are various approaches to compute the similarity between instances, e.g. string similarity methods (see Section 2.4.1), FCA [VJ11] or cluster-based methods [AMR11, DFSB11]. One benefit most from the instance information when a set of instances characterised in both ontologies is available. The easiest way is then to compare the overlap of instances between two classes, i.e., to examine the intersection of their instance sets (Hamming distance, Jaccard distance).

Many research on the matching problem have been done in the last decade, good overviews are provided in [Lin07, CSH06, ES07] and a number of systems for efficient ontology matching [DMDH02, SM01, ELTV04, ES04a, NM00], that use the matching methods whether in isolation or in a combination, have emerged.
In Chapter 2 we discussed the existing approaches in the domain of keyword search in databases and gave an overview of the matching strategies whose aim is to find correspondences between the schemas of two or more repositories. This chapter starts with the clarification of the reasons for our research and the exposition of our motivation in Section 3.1. Further we give the overview of the IQP system in Section 3.2. Proceeding from the motivation and the environment of our work, Section 3.3 presents the statement of the problems to be solved.

3.1 Motivation

To motivate our ontology-based approach, let us exemplify one of the possible situations which arise, for example, while searching for an information in a large-scale database with a flat schema like Freebase [BEP+08].

Suppose Alice saw an advertisement of a film “Munich” and decided to get more details about it. For this purpose she uses a database search system and types in a search window a keyword munich. As a keyword can appear in many rows of different tables, the number of returned results can be enormous. Table 3.1 contains some of the suggestions from Freebase for a query munich together with the tables to which
they belong.

Table 3.1: Suggestions for a query munich (taken from [fre])

<table>
<thead>
<tr>
<th>Suggestion</th>
<th>Freebase Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Munich</td>
<td>City</td>
</tr>
<tr>
<td>Munich</td>
<td>Film</td>
</tr>
<tr>
<td>Munich</td>
<td>Composition</td>
</tr>
<tr>
<td>Munich Philharmonic Orchestra</td>
<td>Conducted ensemble</td>
</tr>
<tr>
<td>Munich Massacre</td>
<td>Event</td>
</tr>
<tr>
<td>FC Bayern Munich</td>
<td>Football team</td>
</tr>
<tr>
<td>Munich International Airport</td>
<td>Airport</td>
</tr>
<tr>
<td>Munich</td>
<td>Band</td>
</tr>
<tr>
<td>Munich Mashine</td>
<td>Band</td>
</tr>
<tr>
<td>Munich Records</td>
<td>Record label</td>
</tr>
<tr>
<td>Munich Syndrom</td>
<td>Musical artist</td>
</tr>
<tr>
<td>Technical University of Munich</td>
<td>Colledge/University</td>
</tr>
<tr>
<td>Munich Waldfriedhof</td>
<td>Cemetery</td>
</tr>
<tr>
<td>Munich</td>
<td>Book</td>
</tr>
<tr>
<td>Munich</td>
<td>Soundtrack</td>
</tr>
<tr>
<td>Munich National Theater</td>
<td>Building</td>
</tr>
<tr>
<td>New Munich</td>
<td>City</td>
</tr>
<tr>
<td>Munich Barons</td>
<td>Ice hockey team</td>
</tr>
<tr>
<td>Munich</td>
<td>TV episode</td>
</tr>
<tr>
<td>Karl Munich</td>
<td>Athlete</td>
</tr>
<tr>
<td>The Munich Mannequins</td>
<td>Poem</td>
</tr>
<tr>
<td>Munich</td>
<td>Place of birth</td>
</tr>
<tr>
<td>William Munich</td>
<td>Author</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Let us have a closer look at the results. Among them are people with a family name Munich, events that took place in Munich, Munich as a city and an urban district, a film, a poem, several bands along with organisations, buildings, companies in Munich. As we can see the suggestions are not only numerous, but they are also very diverse, as their categories show. How should Alice get an overview over all these suggestions and find the only one she really needs? One possibility is to use some ranking in order to guess what was actually meant by the user. But due to the size and structure of the database, it is still of little use in this situation. In a worst case, ranking may fail to identify the desired results, so Alice will still need to scroll all results in order
to find the only one she wants.

Table 3.2: Categorized suggestions for a query munich (taken from [fre])

<table>
<thead>
<tr>
<th>Suggestion</th>
<th>Table</th>
<th>Domain</th>
<th>Top-Level Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Munich</td>
<td>City</td>
<td>Location</td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>Place of birth</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>City</td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Munich</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich Massacre</td>
<td>Event</td>
<td>Event</td>
<td>Time and Space</td>
</tr>
<tr>
<td>Munich International Airport</td>
<td>Airport</td>
<td>Building</td>
<td></td>
</tr>
<tr>
<td>Munich Waldfriedhof</td>
<td>Cemetery</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technical University of Munich</td>
<td>Colledge/University</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich National Theater</td>
<td>Building</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>Film</td>
<td>Mass Media</td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>TV episode</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich The Munich Mannequins</td>
<td>Book</td>
<td>Periodicals</td>
<td>Arts and Entertain-</td>
</tr>
<tr>
<td></td>
<td>Poem</td>
<td></td>
<td>ment</td>
</tr>
<tr>
<td>Munich Philharmonic Orchestra</td>
<td>Composition</td>
<td>Music</td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>Conducted ensemble</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>Band</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich Machine</td>
<td>Band</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich Records</td>
<td>Record label</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich Syndrom</td>
<td>Musical artist</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Munich</td>
<td>Soundtrack</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FC Bayern Munich</td>
<td>Football team</td>
<td>Soccer</td>
<td>Sports</td>
</tr>
<tr>
<td>Munich Barons</td>
<td>Ice hockey team</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ice hockey</td>
<td></td>
</tr>
<tr>
<td>Karl Munich</td>
<td>Athlete</td>
<td>Person</td>
<td>Person</td>
</tr>
<tr>
<td>William Munich</td>
<td>Author</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
hockey team to the domain of Sports. Further the domains are grouped into top-level categories, e.g. Mass media, Periodicals, Music are united to Arts and Entertainment.

Having a possibility to refine search results according to their categories e.g. in a faceted search interface, let us look how the same search would look like if Alice had the categories for a film “Munich”. First she chooses the category Arts and Entertainment, then proceeds to Mass Media and Film. Finally she becomes a ranked list with the films that contain a keyword munich. In this manner, not only the number of suggestions will be drastically reduced, but also the returned result are the most relevant for the user’s information need. So why don’t enable Alice to decide from the very beginning what category to choose? In such a way a database query becomes more like a structured search where a user comfortably navigates through the categories.

As we can see the Freebase categories act like a filter for the results, letting through only the relevant ones. But is this filter also effective? Due to the large size of the Freebase dataset, there arises a situation that a single category may cover a great number of match entries. E.g. for the Alice’s query munich in a category Film it will return the following results:

- several films that contain a keyword munich in title, e.g. Munich: Secrets of a City, Night Train to Munich, Munich Mambo, Munich, The Ghost of Munich,
- some films with a notable filming location munich, e.g. Hannibal Brooks, Port of Memory, Vicky the Viking,
- a film Celibidache: Ravel and Debussy with the participation of Munich Philharmonic Orchestra,
- more than 100 person (actors, directors, music contributor) born in Munich.

As we can see the Freebase categories are not specific enough. On the other hand, if we take YAGO categories, that were extracted from Wikipedia(see Section 2.3.3), they are less general and cover less entities. In our case, the categories Films about the Israeli-Palestinian
conflict, Films based on real events or Films produced by Steven Spielberg are very effective as the only result returned for Alice’s query will be a film “Munich”. Same as Frebase, the categories of YAGO are arranged into the the taxonomic hierarchy, but differ in the grade of granularity.

This example shows us how big the size of the query interpretation space can be just for one keyword. And it gets much bigger when a query contains more keywords because of the large number of possible connections between the tables in a database, e.g. for a query stephen king some of the possible interpretations are an author Stephen King, a football player Stephen King, a king Stephen of England, Stephen Mallinder also known as Tennessee King, a musical composition King Crack produced by Stephen Street, etc. As we can see the simplicity of keyword search comes along with the drastic reduction of expressiveness and quality of the results. Having only a bunch of words to interpret and without any semantic intent specified, it is difficult for an application to decide what should be treated as the desired result. Our intent is thereby to give an ordinary user with only little domain knowledge a possibility while searching for a piece of knowledge in a large-scale database to specify her information need in an interactive and understandable way.

3.2 Introduction to the IQP System

The IQP system [DZN10] is designed to enable users with an interactive, intuitive and quick way to construct a structured query and find a desired result that satisfies their information need. It allows users to refine their search iteratively and with each successive step to reduce a query interpretation space and thus to decrease the total search process.

The major components of the IQP user interface are presented in Fig. 3.1:

1. a search field
2. query construction options
3. structured queries
4. search results

Figure 3.1: Overview of the IQP System

A search field(1) is used for a keyword query input. Users type a word or a bunch of words and click on a search button near the search window to execute the query. Given a keyword query, a system interprets the keywords, generates the query construction options(2), ranks them based on their information gain and presents those with the highest score to users. At the same time a list of structured queries(3) relevant to the options is built and displayed in the ranked order. Once
users detect the intended structured query, they can pick it and the results, i.e. the corresponding data rows from database, will appear in the results window(4). If users cannot find the desired structured queries or results right away, the additional query construction options give them an opportunity to define the meaning of the keywords more precisely and reduce the interpretation space. Once an option is chosen, the refinement of the structured queries is triggered. In such a way these queries reflect both the keyword query and the chosen option.

For example, in order to find the books written by Stephen King, Alice types a keyword query *stephen king* in a search field. Based on the available information about the database schema IQP system generates a list of structured queries that are presented in a window(3) on the right. The suggested queries reflect different semantic interpretations of the keywords. Among them are an author Stephen King, a football player Stephen King, whereas the other query represents the first king of Hungary Stephen I. At the same time the number of query construction options are built and presented in the options window. The options are intended to specify the meaning of the query and reduce the space of the structured queries.

If a query Alice searches for, in our case an author Stephen King, is among the top generated structured queries, she can choose it from the list(3) and after double-click on it, the corresponding results will be shown in the result window(4).

In case neither query nor result window contains the desired result, Alice can use the query construction options to refine the list of structured queries. For example, the selecting of the options *person* and *author* will lead Alice to the intended result. Whenever Alice selects an option, only the structured queries and results satisfying the selected options will remain. The options are the generalization of several structured queries and denying or accepting of an option triggers the denying or accepting of the queries subsumed by this option. The interaction between Alice and the system lasts until either the desired structured query or results are found or there are no options left, meaning that the desired interpretation doesn’t exist in the database.
Chapter 3 Problem Analysis

3.3 Problem Statement

Keyword search allows ordinary users to search for information without any knowledge about the database schema or query language. But on the other hand, without any known semantic intent of the query, the number of possible interpretations can be enormous and the applications have difficulties in detecting the desired search results.

The IQP query construction system aims at filling the gap between the ease of usage of keyword search and expressivity of structured query languages. It empowers naïve database users to create their own structured queries in an interactive way, starting from simple keywords and refining the initial query using options automatically suggested by the system. If the user selects an informative option, IQP can reduce the search space efficiently. In order to generate such query construction options, IQP can make use of the database internal structures such as database schema and keyword occurrence statistics. The efficiency and usability of IQP using these options was experimentally confirmed for the middle-sized datasets such as IMDB and Lyrics.

As the number of tables and entries in a database increases and its schema becomes more complex. At the same time, the number of possible query construction options generated by system explodes, whereas their informativeness drops. That’s why in order to support an efficient query construction process on large-scale databases with flat schemas, we need a new approach that will be able to organize the database schema information and to generate informative options. Recently, IQP was extended to integrate ontological level to enable users to reduce the query interpretation space efficiently in the interaction process and to speed up the query construction process in case of flat and big database schemas.

In the current implementation of IQP, Freebase ontology, which comprises about 100 domains and several top-level categories, is used. However, in a general case of a large-scale database, ontological knowledge may not be available. Even on Freebase, the categories may not be specific enough to optimally divide the search space. Our aim is to further
3.3 Problem Statement

improve efficiency of the query construction process and to make the IQP system portable on large-scale databases that do not possess any predefined ontological level.

We intend to use YAGO ontology for the achievement of this goal. The hierarchical structure of YAGO’s semantic categories is a good basis for generalizing of the database schema of Freebase. We expect that YAGO hierarchy can optimally divide the search space and reduce the interaction cost in the query construction process. While integrating YAGO into the IQP system and using the ontology for the Freebase dataset we have to solve the following problems:

1. Mapping of the Freebase tables to the semantic concepts of the YAGO ontology

2. Efficient hierarchy construction at the query time

We treat a database table as a grouping of the entities that belong to the semantic concepts. In order to automatically assign semantic categories from YAGO to the tables of the Freebase dataset the matching of two schemas should be done, i.e. to identify that Stephen I of Hungary from Freebase and king stephen from YAGO denote the same person. After the correspondencies between the entities are retrieved, the most suitable category has to be identified, e.g. a YAGO category writer for a Freebase table author. This is the problem to be solved during the preprocessing step.

During a search step we intend to enable users with a possibility, starting with the simple and general categories, like object or person, to refine their information need and navigate to more specific categories, like book or tv actor. For this purpose we have to organize the database schema, i.e. to build a category hierarchy for the interpretations of a given keyword query. The current solution based on the Freebase taxonomy supports construction of the hierarchy with a small and well-defined balanced structure. In contrast, the YAGO categories are arranged in an unbalanced hierarchical tree. That’s why a new solution to the hierarchy construction is needed. As we deal
with a large-scale database, the generation of such hierarchy for the whole database and its manipulation during the query processing is a costly and time consuming action. At the same time, a keyword query can cover only a fairly small part of the database schema. Therefore we need to develop efficient algorithms for query-dependend hierarchy construction during search.
The focus of our work lies in integration of YAGO ontology into the IQP query construction system. In this context, we face two main problems, namely:

1. schema matching between Freebase and YAGO
2. efficient hierarchy construction

Chapter 4 gives the deeper insight into the design of our solution. First we provide the notions and definitions needed for a proper understanding of our implementation. Then we describe in detail the techniques used for schema matching and hierarchy construction.

4.1 Definitions

Before we proceed with the description of our conceptual design, we introduce the terms and concepts needed for understanding of our solution.

In order to query a database, a user constructs a keyword query, that consists of one or more keywords.

Definition 4.1.1. A keyword query is a set of keywords, denoted by $K = \{k_1, k_2, \ldots, k_n\}$. 
Some examples of the keyword queries are: *stephen king, munich, brad inglorious bastards*.

To a keyword query the IQP framework generates a set of structured queries and a number of query construction options.

**Definition 4.1.2.** A structured query $Q$ is an expression of relational algebra which is composed of keyword interpretations $\{A_i : k_i, \ldots, A_i : k_j\}$ and the operators selection ($\sigma$) and natural join ($\bowtie$) [DZN11].

**Definition 4.1.3.** A keyword Interpretation is denoted by $A_i : k_i$, which maps a keyword $k_i$ to an element $A_i$ of a structured query. $A_i$ can refer to a table, an operator, an attribute, or a value in a predicate. $A_i$ is also called an interpretation of $k_i$ [DZN11].

Some examples of the structured queries are:

$$\sigma_{(stephen,king)} \in \text{name}(Author),$$

$$\sigma_{(stephen,king)} \in \text{name}(Author) \bowtie \sigma_{It} \in \text{title}(Book)$$

**Definition 4.1.4.** A query construction option $QCO$ is a statement. Given a query interpretation $I : K \rightarrow Q$, we say that $QCO$ subsumes $I$, if and only if we can infer $QCO$ based on $I$ [DZN11].

**Definition 4.1.5.** Query Interpretation: given a keyword query $K$, we say that a structured query $Q$ is a query interpretation of $K$, if and only if there is a set of keyword interpretations $A_i : k_i$, where $A_i \in Q$ and $k_i \in K$, such that (1) each keyword in $K$ is interpreted as at most one element of $Q$; (2) given a sub-structure $Q'$ of $Q$, if after removing $Q'$ the remaining structure of $Q$ is also a structured query, then there is at least one keyword in $K$ that is interpreted as an element of $Q'$.

If $A_i : k_i$ contains the keyword interpretations for all the keywords in $K$, we call $Q$ a complete interpretation of $K$. Otherwise, we call $Q$ a partial interpretation of $K$ [DZN11].

Before describing the schema matching of YAGO and Freebase, we need to introduce the concepts of matching, correspondence, similarity, alignment and mapping.

**Definition 4.1.6.** Matching is the process of finding correspondences between the elements of different ontologies.
4.2 Overview of Preprocessing and Search Steps

**Definition 4.1.7.** Correspondence is the relation holding according to a particular matching algorithm between elements of different ontologies.

**Definition 4.1.8.** A similarity \( \alpha : \sigma \times \sigma \to R \) is a function from a pair of elements to a real number expressing the similarity between two objects such that:

1. \( \forall x, y \in \sigma, \alpha(x, y) \geq 0 \) (positiveness)
2. \( \forall x \in \alpha, \forall y, z \in \sigma, \alpha(x, x) \geq \alpha(y, z) \) (maximality)
3. \( \forall x, y \in \sigma, \alpha(x, y) = \alpha(y, x) \) (symmetry)

**Definition 4.1.9.** Alignment is a set of correspondences between two or more ontologies. The alignment is the output of the matching process.

**Definition 4.1.10.** Mapping is the directed version of an alignment: it maps the elements of one ontology to at most one element of another ontology.

### 4.2 Overview of Preprocessing and Search Steps

For efficient query construction we need to group the database tables associated with keyword occurrences to enable more general view in the construction process. For this purpose we map the Freebase tables to the semantic categories of YAGO. The hierarchical relationships between the categories allow us to organize the database schema and thus reduce the query interpretation space.

We split the generation process of YAGO-based query construction options into two parts: preprocessing and search (see Figure 4.1).

The schema matching takes place during the preprocessing step is query-independent. The input of the matching process is two repositories (YAGO and Freebase) and the resulting output is the mapping of the database tables to semantic categories of YAGO.

After a user submits a keyword query, the IQP system generates a number of query construction options. The options are generated based
on the keyword occurrences in the database, a keyword query and the mapping between YAGO and Freebase during the preprocessing step. Further in order to reduce the number of options, we construct the query-based hierarchy of concepts using the subordinary and superordinary relations between the semantic categories of YAGO.

### 4.3 Schema Matching

The following actions take place during the preprocessing step:

1. an identification of concepts and instances in YAGO and Freebase. We treat the semantic categories of YAGO as concepts and the real-world objects the categories are associated with as instances.
4.3 Schema Matching

As to Freebase, the entity tables build a set of concepts and their data entries are regarded as instances.

2. matching, i.e. establishment of correspondences between Freebase and YAGO concepts

3. based on the resulting alignment, each Freebase entity table is mapped to the most likely YAGO concept.

4.3.1 Concept Identification

The common elements of YAGO ontology are concepts, e.g. *Person, Entity* (see Figure 4.2).

![Concepts and instances in YAGO](image)

Figure 4.2: Concepts and instances in YAGO

The concepts represent the semantic categories and are hierarchically organized with the help of relation *subClassOf*. The concepts are associated with a set of instances, e.g. *Book* with \{Stephen King, James Joyce\} and *Entity* with \{Stephen King, James Joyce, It, Ulysses\}. The relation *type* links together a concept with its associated instances.

We think of the Freebase dataset as a collection of entity tables, that describe the objects of the real world, like book genre, location,
airline, award, national anthem, etc. For the matching of YAGO and Freebase we treat the entity tables as concepts, e.g. the tables author, book (see Figure 4.3). Each entity table is associated with its data entries, which are seen as instances, e.g. Stephen King, James Joyce are instances that belong to the concept author.

### 4.3.2 Matching

In order to map a database table to a semantic category of the ontology we have to match the schemas of Freebase and YAGO, i.e. reveal the correspondences between their elements. An example of the matching process is shown in Figure 4.4.

An input is: YAGO and Freebase. During the matching process a set of correspondences between the elements of knowledge bases is retrieved. For example, the table entry King, Stephen from Freebase database is matched to the instance Stephen King from YAGO, a table author corresponds to the concept Writer. The output of the matching process is a mapping of Freebase entity tables to the semantic categories of YAGO. Each Freebase concept (table) is mapped only to one most likely YAGO concept (semantic category), whereby each YAGO
concept can be associated with several Freebase concepts. For example, the YAGO concept *writer* may unify the Freebase concepts *author*, *comic book author*.

### 4.3.3 Similarity Score Computation

For estimation of the correspondences between two or more schemas the following matching strategies can be used: element-, structure- and instance-based (see Section 2.4). In our work we use a combination of element- and instance-based techniques.

The assumption behind our approach is that the similarity between two concepts $X$ and $Y$ - $\alpha(X,Y)$ - depends on:

- element-based similarity $\alpha_{el}$ - the similarities of the terms used to
designate concepts (labels or names)

- instance-based similarity $\alpha_i$ - the similarity of the instances of the concepts

The total similarity $\alpha(X, Y)$ is calculated as a weighted sum of similarities $\alpha_{el}$ and $\alpha_i$:

$$\alpha(X, Y) = a \times \alpha_{el}(X, Y) + b \times \alpha_i(X, Y) \quad (4.1)$$

We consider that both the element-based and instance-based similarity factors contribute equally to the revealing of the correspondences between several ontologies and thus $\alpha_{el}$ and $\alpha_i$ are weighted equally $a = b = 0.5$.

**Element-Based Similarity**

Element-based similarity $\alpha_{el}(X, Y)$ reflects the similarity of the identifiers of the concepts and is based on N-Gram similarity presented by Kondrak [Kon05]. An n-gram of a string $s$ is any substring of $s$ of some fixed length $n$. The simplest way is to choose $n$, and count the number of n-grams two strings $s$ and $t$ have in common $\|ngram_n(s) \cap ngram_n(t)\|$. For example, for $n = 2$, a string $s = \text{Stephen}$ has n-grams: \{st, te, ep, ph, he, en\} and a string $t = \text{Steven}$: \{st, te, ev, ve, en\}. So two strings $s$ and $t$ have the n-grams in common: \{st, te, en\} and $\delta_{ngram}(s, t) = 3$.

The normalised version of N-Gram similarity is defined as the ratio of the number of n-grams that are shared by two strings and the total number of n-grams in both strings (Dices’s Coefficient):

$$\alpha_{ngram}(s, t) = \frac{2 \times \|ngram_n(s) \cap ngram_n(t)\|}{\|ngram_n(s)\| + \|ngram_n(t)\|}$$

For example, for $n = 2$, $s = \text{Stephen}$ and $t = \text{Steven}$, the N-Gram similarity is calculated as $\alpha_{ngram}(s, t) = \frac{2 \times 3}{6 + 5} = 0.54$.

In our model we use the n-grams of the length $n = 2$ and compute the similarity of the labels used to designate concepts (database table names of Freebase and semantic concepts of YAGO):

$$\alpha_{el}(X, Y) = \alpha_{ngram}(X, Y) \quad (4.2)$$
Thus, the $\alpha_{el}(X,Y)$ is defined as N-Gram similarity of the labels used for identification of the concepts $X$ and $Y$.

**Instance-based Similarity**

Besides a label that distinguishes a concept from other concepts within a collection, each concept is also associated with a set of its instances. The intuition behind our approach is that concepts $X$ and $Y$ are same if $X$ contains the same instances as $Y$. The more instances are shared by $X$ and $Y$, the more similar the concepts are. For example, *writer* and *author* share the instances \{Stephen King, James Joyce\} and are considered to describe the same domain, whereas *writer* and *book* have no common instances and thus are not similar. So in order to compute $\alpha_i(X,Y)$ we have to find what instances the ontologies $X$ and $Y$ have in common.

As Wikipedia constitutes a backbone of both Freebase and YAGO ontology, we have a unique opportunity to exploit the Wikipedia ids as the individual representations of the instances.

![Number of Freebase Concepts with a Given Percentage of Wikipedia Instances](image)

Figure 4.5: Number of Freebase concepts with a given percentage of Wikipedia instances

The pie chart in Figure 4.5 shows the number of Freebase concepts with the percentage of the Wikipedia instances, i.e. the instances that possess a Wikipedia id. The pie reflects the whole set of Freebase concepts. Each fraction represents a subset of concepts (tables) and a colour of the fraction denotes the percentage of instances (data entries), contained by the subset. For example the middle-sized fraction
represents a subset of 212 concepts and the colour green shows us that 34 – 66% of instances associated with the concepts possess Wikipedia id.

It can be clearly seen, that the vast majority of Freebase concepts (about 82% of the total database) have more than 66% Wikipedia instances. Meanwhile, only a small fraction (4%) contains instances that don’t possess the needed identifier.

Further we analysed the proportion of Wikipedia instances shared by both ontologies, YAGO and Freebase. The results, based on the fraction of Freebase concepts that contains the highest percentage of Wikipedia instances, are shown in Figure 4.6.

![Figure 4.6: Number of Freebase concepts with a given percentage of Wikipedia and YAGO instances](image)

The estimated results show that about 40% of the Freebase tables contain more than 66% Wikipedia instances that are also shared by YAGO ontology. Moreover, the instances of about two thirds of the database tables have an overlap of more than 33% with YAGO instances. To this end, we assume that Wikipedia id can serve as a good identifier to find the corresponding instances of two or more schemas.

Wikipedia ids help us to identify the instances shared by the concepts of YAGO and Freebase. The set of common instances is a key point in the computation of the instance-based similarity $\alpha_i(X,Y)$,
which is based on the joint probability distribution between two concepts \(X\) and \(Y\) (compare with the similarity measure of the GLUE system [DMDH02]). The distribution of the instances comprises the following probabilities:

- \(P(X, Y)\) - probability that a randomly chosen instance from the universe belongs to \(X\) and \(Y\), i.e. a part of universe that belongs to \(X\) and \(Y\)
- \(P(\bar{X}, Y)\) - fraction that belongs to \(Y\) but not to \(X\)
- \(P(X, \bar{Y})\) - fraction that belongs to \(X\) but not to \(Y\)
- \(P(\bar{X}, \bar{Y})\) - fraction that belongs neither to \(X\) nor to \(Y\)

For the calculation of the similarity between two sets of instances we use the similarity measure known as the Jaccard coefficient, which is based on the joint probability:

\[
\alpha_{Jaccard}(X, Y) = \frac{P(X, Y)}{P(X, Y) + P(\bar{X}, Y) + P(X, \bar{Y})}
\]

It takes the lowest value 0 when instance sets of \(X\) and \(Y\) are disjoint, e.g. *book* and *author*, and the highest value 1 when \(X\) and \(Y\) are the same concept, like *writer* and *author*.

We define the instance-based similarity of two concepts as:

\[
\alpha_i(X, Y) = \alpha_{Jaccard}(X, Y)
\]  \hspace{1cm} (4.3)

And the overall similarity \(\alpha(X, Y)\) of the concepts \(X\) and \(Y\) is an equally weighted sum of the N-Gram similarity of the labels of \(X\) and \(Y\) and the Jaccard similarity of their instance sets. Equations (4.2) and (4.3) inserted into equation (4.1) and \(a = b = 0.5\) result in:

\[
\alpha(X, Y) = a \times \alpha_{ngram}(X, Y) + b \times \alpha_{Jaccard}(X, Y)
\]  \hspace{1cm} (4.4)

For example, the similarity of \(X = \text{writer}\) from YAGO and \(Y = \text{author}\) from Freebase is calculated as follows: \(\alpha(X, Y) = 0.5 \times 0 + 0.5 \times 1 = 0.5\).
After the similarity between the elements of Freebase and YAGO ontology has been computed, we map the concepts of Freebase to the most similar concept in YAGO, i.e. to the concept with the highest value of $\alpha(X,Y)$.

### 4.4 IQP Hierarchy Construction

In order to search for information in a database, users formulate their demand in form of a keyword query and submit it to the IQP query construction system. The system analyses the occurrences of each keyword in a database and generates a number of possible attribute-keyword interpretations for each keyword. The attribute-keyword interpretations are associated with the database tables. However, the query construction options created based on the database tables only are not informative enough to enable efficient query construction. The more general ontology-based query construction options help us to cope with this problem. These options summarize the database schema and speed up the query construction process. In order to give users the possibility, starting with the simple and general categories to refine their information need and navigate to more specific categories, we have to organize the ontology-based query construction options i.e. to build a category hierarchy for the interpretations of a given keyword query.

In the following we describe our design of the hierarchy construction on the example of YAGO ontology and Freebase dataset. During the preprocessing step (see Section 4.3.2 we mapped Freebase tables to the concepts of YAGO. Now we want to exploit the hierarchical relationships between the concepts in YAGO in order to achieve efficient query construction and generate the options with the high information gain.

Figure 4.7 presents an example of such hierarchy for a keyword *king*. The Freebase tables *TVWriter* and *Author* are mapped to a YAGO concept *Writer* and can be put together. A table *Lyricist* is assigned to the concept *Lyricist*, which in its turn belongs to the concept *Writer*. Thus, the concept *Writer* groups three Freebase tables *TVWriter*, *Author* and *Lyricist*. Further, a concept *Person* unites the tables
4.4 IQP Hierarchy Construction

Figure 4.7: An example of IQP hierarchy for a keyword *king*.

covered by *Writer* and other tables, e.g. *Monarch*. In such a way all tables associated with keyword *king* are arranged to the hierarchy with the root concept *Entity*.

The hierarchy construction brings further challenges. First, the current version of IQP is based on the Freebase taxonomy and supports construction of the hierarchy with a small and well-defined balanced structure. As our example shows, the Freebase concepts can be mapped to different levels of YAGO hierarchy and the resulting tree can be unbalanced.

Second, as we deal with a large-scale database, the number of concepts to manage is big. For example, Freebase contains 1534 tables and each table has in average 6.6 superordinated YAGO concepts. Materialisation of the hierarchy for the whole database during the preprocessing step and and its manipulations while searching are time-consuming and costly operations. Besides, we made an observation that the number of concepts for a keyword covers only a part of the hierarchy. For example, keyword *stephen* is associated only with 374 tables, keyword
king with 496 and a phrase stephen king only with 65 tables. That’s why we decided to establish the hierarchical relationships between the concepts only on demand for a given query.

Before we proceed with the description of the algorithm for hierarchy construction, let us first have a closer look at the structure of the hierarchical tree of IQP(see Figure 4.8).

![Hierarchical Tree of IQP](image)

**Figure 4.8:** IQP hierarchy structure using YAGO ontology

- A hierarchical tree contains the following elements: nodes that represent concepts of YAGO and Freebase(e.g. *Entity, Author*) and edges that express superordinary and subordinary relations between parent and child node(e.g. a connection between the nodes *Writer* and *Communicator*).

- *Entity* is the root node of the YAGO ontology. It is the most general concept. Any node that is not a root node can have only one direct parent.
4.4 IQP Hierarchy Construction

- The depth of a node is the length of the path to its root, e.g. the depth of a node Writer is 3. The depth of a root node is 0.

- Parent-child relations in the ontology are transitive, thus: if (x,z) and (z,y) are in a parent(child) relation, then also (x,y) are in a parent(child) relation. If Person is a parent of Communicator and Entity is a parent of Person than Entity is also a parent of Communicator. Respectively if Person is a child of Entity and Communicator is a child of Person then Communicator is a child of Entity. Due to this property the child and parent relations between the nodes are established. For example, a concept Writer has child relation to Author and parent relations to Communicator, Person and Entity.

- Sibling nodes share the same direct parent node, like Person and Location. Only the nodes with the same depth can be siblings.

During hierarchy construction we have to ensure that all child-parent relations are established and all concepts have the proper depth. As the traversal of hierarchical tree in the later user-interaction process is expensive, we need to establish all indirect parent-child relations in advance. A naïve approach is to this process for each node of the hierarchical tree separately. The adding of a new node would trigger the updating of the whole branch of the tree. For example, if we add a node Leader to the concept Person we have to inform the nodes Entity and Person that they have new children and the children of Leader that they have got new parents. In worst case, the construction of the hierarchical tree and establishing of the indirect parent-child relation takes $O(n^2)$ time, where $n$ is the number of nodes. Therefore we optimize this process and split it into two parts: propagation of the parent relations ($O(k \times n)$ time, where $k$ is the average number of parents) and propagation of the child relations ($O(m \times n)$ time, where $m$ is the average number of direct children).

Our algorithm 1 takes as an input a set of Freebase tables $T$ associated with a keyword and an ontology $o$. 
Algorithm 1: HierarchyConstruction

\begin{algorithm}
\caption{HierarchyConstruction}
\begin{algorithmic}
\State \textbf{input}: tables $T$, ontology $o$
\State \textbf{output}: hierarchy of concepts $H$
\Function{begin}
\State establish child relations, set depth
\State $C \leftarrow \text{BottomUp}(T, o)$
\State establish parent relations
\State $H \leftarrow \text{TopDown}(C)$
\EndFunction
\end{algorithmic}
\end{algorithm}

It takes two steps to construct a hierarchy of concepts (see Figure 4.9):

1. Procedure \textbf{BottomUp} provides each concept with the corresponding depth and establishes the child relations between the concepts, i.e. each node knows its subordinated concepts. For example, the depth of the concept \textit{Entity} is 0 and concept \textit{Person} has child nodes \textit{Communicator}, \textit{Writer} and \textit{Author}.

2. Procedure \textbf{TopDown} establishes the parent relations between the concepts, e.g. concept \textit{Writer} has parent concept \textit{Entity},

The output of the procedure is a set of concepts $H$ with all hierarchical relations (parent and child relations) and properties (node depth) are defined.

The input of the Procedure \textbf{BottomUp} is a set of tables $T$ associated with a keyword and an ontology $o$. During the processing step we mapped the Freebase tables to the ontology concepts and our first step now is to retrieve for each table in $t$ the corresponding concept and its superordinate concepts.

Depth of a concept depends on the number of node on its path to a root node and the the depth of the root is 0. Then the depth of table $t$ is defined as the number of superordinate concepts plus one for the assigned concept. Further we follow the path of the mapped concept from the bottom up to the root. In each iteration according to the transive nature of parent-child reltions the child relations between the concepts are propagated along the tree. For example, in the first
4.4 IQP Hierarchy Construction

In the next iteration to the concept Writer we add its direct child node Author. In the next iteration to the concept Communicator a direct child node Writer and an indirect child Author. At the same time the depth of the node is set. The distinct concepts are saved in the resulting set $C$. The propagation of the child relations takes at most $O(k \times n)$ time, where $k$ is the average number of parents. In case of YAGO ontology $k = 6.6$.

The concepts with the depth and child relations set are an input of the Procedure TopDown. The idea behind the operation is to traverse the hierarchical tree starting with the root node down to the nodes with the maximal depth minus one. Here we use transitive reactions to establish the parent relations.

For this purpose we first sort the concepts according to their depth. Then we start at the root of the hierarchy and establish first parent relations.
Procedure BottomUp(T, o)

input: tables T, ontology o
output: C - concepts with child relations established

begin
k ← null
child ← null
parent ← null
P ← ∅
C ← ∅

for t ∈ T do
child ← t
get YAGO concept mapped to table t
parent ← getAssignedConcept(t, o)
get superordinate concepts of parent
P ← child.getParents()
k ← P.size()
child.setDepth(k + 1)
C.addConcept(child)
if C contains parent then
use the existing instance of parent

for (i ← 0 to k + 1) do
parent.setDepth(k − i)
establish the child relations
parent.addDirectChild(child)
parent.addChildren(child.getAllChildren())
C.addConcept(parent)
child ← parent
parent ← P.get(i)
if C contains parent then
use the existing instance of parent

relations between the root’s children nodes at depth 1 and the root, e.g. between Person and Entity. In the next step, we set the parent relations between the concepts two and one steps lower in the hierarchy. The advantage here is that the parent relations on the higher levels have been already determined. Thus the parents of each node comprise a direct parent node and the parent nodes of the direct parent. We repeat
**Procedure** TopDown(C)

**input**: concepts C with child relations set

**output**: hierarchy of concepts H

begin

\[
\begin{align*}
\text{maxDepth} & \leftarrow 0 \\
S < \text{depth}, \text{Concepts} & \leftarrow \emptyset \\
\text{Children} & \leftarrow \emptyset \\
H & \leftarrow \emptyset
\end{align*}
\]

sort concepts C sorted according to their depth

\[
S < \text{depth}, \text{Concepts} \leftarrow \text{SortConcepts}(C)
\]

\[
\text{maxDepth} \leftarrow S.\text{keySet}()
\]

for (d ← 0 to maxDepth) do

establish the parent relations

\[
\text{Concepts} \leftarrow S.\text{get}(d)
\]

for concept ∈ Concepts do

\[
\begin{align*}
\text{Children} & \leftarrow \text{concept}.\text{getAllDirectChildren}() \\
\text{for} \ c\text{Child} \in \text{Children} \ do \\
& \text{cChild}.\text{addDirectParent}(\text{concept}) \\
& \text{cChild}.\text{addParents}(\text{concept}.\text{getAllParents}()) \\
& H.\text{add}(\text{concept})
\end{align*}
\]

the procedure is repeated till the maximal depth is reached (e.g. depth 4 in our Example) and the processed concepts becomes a part of the resulting set H. The overhead is therefore reduced, so that this step can be performed efficiently in \(O(n)\) time for sorting of the concepts according to their depth plus \(O(m \cdot n)\) time for establishing of the child relations, where \(m\) is the average number of direct child nodes.
The current version of the IQP system provided support for the querying of the data from Freebase and building of query construction options based on Freebase ontology. We designed the new version with the intention to be able to support different datasets and enable integration of generic ontologies. The IQP system can be then easily extended and suitable for the future use.

The main components of our solution are presented in Figure 5.1. Users express their information need in form of a keyword query. Given a keyword query, for each keyword the IQP system generates a number of keyword interpretations based on the occurrences of the keyword in different attributes. The attributes are associated with database tables. Only the tables chosen on hand of the generated keyword interpretations can become concepts. Further each chosen table is mapped to a semantic category from ontology. While a database table can have only one corresponding match in ontology, one category can be assigned to several tables.

Due to our model a concept can be either a database table or a category from ontology. Concepts have common properties, like a depth, parent or children concepts, an ability to add a new child or parent. The difference lies in the establishing of parent/child relations between the concepts. A database table can’t have further tables as children concepts. Whereas a semantic category can have another more general category as a parent or less general category as a child. The similarities
and differences of properties motivate our design of a concept as an abstract class. A table and a category inherit the functionality and properties of a concept, and then add new functionality of their own. Hierarchy represents the collection of the concepts.

Keyword query, keyword, keyword interpretation, table, attribute, category, concept and hierarchy constitute a static core of our solution. As we want to use different datasets and integrate generic ontologies, we define interfaces for an ontology and for a database. While adding new ontology the implementing class should provide information about the semantic categories contained by the ontology and the parent categories of each category. Database schema knowledge and mapping between the database tables and the categories of an ontology is required while adding a new dataset. The implementing of a new database. The ad-
vantage of our design lies in the fact that no matter what ontology
or database was chosen, there is no need to change the core imple-
mentation. The generic nature of our solution ensures reusability and
extendability of the IQP system.
To evaluate the new query construction options, we performed extensive experiments.

### 6.1 Experiment Setup

Our experiments are performed using Freebase dataset from June 2011 [Goo11]. This dataset includes approximately 7500 tables containing more than 20 million entities in about 100 domains. We imported data dumps provided by Freebase into a MySQL database and indexed the data and the schema using Lucene [MHG09].

Freebase data comes along with a set of user-defined views. A view in Freebase is defined in MQL - a structured query language and possesses a natural language description supplied by the view’s creator. Some examples of Freebase views include *Artists of Krautrock*, *Courtney Love discography*, and *Malls in Dubai*. A MQL view contains the information about the tables and keywords involved to retrieve the results from Freebase. Thereby, the view can span over several tables and contain multiple keywords. In order to obtain a query set for our experiments, we automatically transformed a set of MQL expressions describing the views into the IQP query format. To form the corresponding keyword query, we extracted keywords from the view description.

For the evaluation we randomly selected 254 queries with up to three
Chapter 6 Evaluation

keywords. We partitioned the extracted test queries into three sets according to the number of keywords contained in the keyword query.

6.2 Experimental Results

The aim of our experiments is to examine how good can YAGO-based query construction options assist users to construct a structured query starting with single keywords.

To this end we compare efficiency of the query construction process with different options. Specifically, we are interested in the efficiency of query construction using YAGO in comparison to other ontologies.

We evaluate the following scenarios:

- **NoOntology** - options generated without the involvement of any ontology. In this case the options are sub-interpretations of the structured queries as described in [DZN11].

- **Freebase** - options generated with existing domain hierarchy of Freebase. Here we involve more general query construction options based on the Freebase taxonomy in addition to the sub-interpretations.

- **YAGO** - options generated with YAGO ontology. This scenario includes YAGO-based options and sub-interpretations (YAGO) in the query construction process.

Our experiments are performed automatically and emulate the query construction process. The options are regarded one by one in the same order as they are presented to users. In each step we check whether the current option subsumes the intended structured query. In this case the option is accepted. Otherwise the option is denied. Besides the query construction options, IQP presents a list of the structured queries to the users. The acceptance of the option lead to an update of this set. If the intended query is among the top-5 structured queries, the process terminates. In this way we count the number of options the user would need to evaluate to retrieve the intended result from Freebase.
6.2 Experimental Results

Figures 6.1, 6.2, and 6.3 present the results of our experiments using 1-keyword, 2-keyword and 3-keyword test query sets respectively. The Y-axis is the average number of the query construction options needed to construct the intended structured query. The X-axis presents the scenario used by the evaluation. For example, 9.49 options on average were needed by “NoOntology” scenario of 1-keyword test set and 6.84 by YAGO scenario of 3-keyword test set.

As Figures 6.1, 6.2, and 6.3 show, the worst efficiency was achieved using the options generated by “NoOntology” scenario without the involvement of any ontology. In this case, query construction required on average 9.48 steps in case of 1-keyword queries, 9.36 with 2-keyword queries and even 22.14 options while running the experiments on the 3-keyword query test set.

The options generated using an ontology decrease the cost of the query construction process. The average number options that the user need to evaluate dropped by more than 50% for 1-keyword queries and about 30% for 2- and 3-keyword query sets.

In summary, we can see that “YAGO” scenario leads to a significant improvement in comparison to “NoOntology” scenario for all query sets. The options generated using YAGO also bring significant improvement
Figure 6.2: Number of query construction options in a 2-keyword query set in three different scenarios

Figure 6.3: Number of query construction options in a 3-keyword query set in three different scenarios

for 2- und 3-keyword query sets compared with “Freebase” scenario. The paired t-test confirms statistical significance of this result for the confidence level of 95%.
Conclusions

The IQP system combines expressivity and effectiveness of the database structured queries without the loss of easiness and simplicity of keyword search. The aim of this thesis is to assist the IQP application to successfully overcome the challenges encountered with large-scale relational databases, such as database schema complexity and flat structure of the data. For this purpose we integrated YAGO ontology into the IQP query construction system, as the hierarchical structure of semantic categories of YAGO is a good basis for structuring of the database schema and generating informative query construction options.

In the first step, we performed a mapping between the Freebase tables and the semantic categories of YAGO. A combination of element- and instance-based matching techniques helped us to reveal similarity not only between the concepts of two schemas but also between the sets of their instances, i.e. real-life objects. Then, we derived query construction options from the YAGO ontology and arranged these options into a hierarchy. We introduced an efficient algorithm that is able to reduce the response time of the system during the interaction with users. We designed our implementation in a generic way, so that the new version of IQP became reusable and extendable offering an easy process for integration of other ontologies.

We implemented our solution on the top of the Freebase dataset. We evaluated our approach using a set of 254 test queries extracted from the user-defined views provided by Freebase. Our evaluation results
confirm that ontology-based options improve efficiency of the query construction process. The average number of the YAGO-based options needed to construct the intended query is significantly smaller than the average number of options generated without the involvement of any ontology. Moreover, for 2- and 3-keyword query sets a YAGO-based scenario outperforms the scenario that uses options generated with the existing domain hierarchy of Freebase. These results indicate that YAGO ontology can be helpful for the other large-scale databases without categorical information.

This work opens many interesting future research directions. First of all, the experiments performed in this work can be repeated on other large-scale datasets. Furthermore, the usability of the system can be proven in a use case study. It is also interesting to see how the system performs with domain-specific ontologies in the enterprise settings.


[BCSW07] Holger Bast, Alexandru Chitea, Fabian Suchanek, and In- 
gmar Weber. Ester: efficient search on text, entities, and 
relations. In SIGIR ’07: Proceedings of the 30th annual in-
ternational ACM SIGIR conference on Research and develop-
ment in information retrieval, pages 671–678, New York, 
NY, USA, 2007. ACM.

[BDG+11] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, 
Raquel Trillo Lado, and Yannis Velegrakis. Keyword search 
over relational databases: a metadata approach. In Pro-
ceedings of the 2011 international conference on Manage-
ment of data, SIGMOD ’11, pages 565–576, New York, NY, 
USA, 2011. ACM.

[BEP+08] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim 
Sturje, and Jamie Taylor. Freebase: a collaboratively cre-
ated graph database for structuring human knowledge. In Pro-
ceedings of the SIGMOD ’08, pages 1247–1250, 2008.

[BHN+02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen 
Chakrabarti, and S. Sudarshan. Keyword searching and 
browsing in databases using banks. In ICDE ’02: Pro-
ceedings of the 18th International Conference on Data En-
Computer Society.

[BKKB05] Abraham Bernstein, Esther Kaufmann, Christoph Kiefer, 
and Christoph Bürki. Simpack: A generic java library for 
similarity measures in ontologies. Technical report, De-
partment of Informatics, University of Zurich, 2005.

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sren 
Auer, Christian Becker, Richard Cyganiak, and Sebastian 
Hellmann. Dbpedia – a crystallization point for the web of 
data, 2009.


[DZN11] Elena Demidova, Xuan Zhou, and Wolfgang Nejdl. A probabilistic scheme for keyword-based incremental query con-


[HP02] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In VLDB ’02: Pro-


conference on Management of data, pages 115–126, New York, NY, USA, 2007. ACM.


1http://wiki.freebase.com/wiki/MQL


Hiermit versichere ich, dass ich die vorliegende Masterarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 09.12.2011

............... 

(Iryna Oelze)