From Queriability to Informativity, Assessing "Quality in Use" of DBpedia and YAGO

Tong Ruan(✉), Yang Li, Haofen Wang, and Liang Zhao

Department of Computer Science and Engineering,
East China University of Science and Technology, Shanghai 200237, China
{ruantong,whfcarter}@ecust.edu.cn, marine1ly@163.com, 252007913@qq.com

Abstract. In recent years, an increasing number of semantic data sources have been published on the web. These sources are further interlinked to form the Linking Open Data (LOD) cloud. To make full use of these data sets, it is necessary to learn their data qualities. Researchers have proposed several metrics and have developed numerous tools to measure the qualities of the data sets in LOD from different dimensions. However, there exist few studies on evaluating data set quality from the users' usability perspective and usability has great impacts on the spread and reuse of LOD data sets. On the other hand, usability is well studied in the area of software quality. In the newly published standard ISO/IEC 25010, usability is further broadened to include the notion of "quality in use" besides the other two factors, namely, internal and external. In this paper, we first adapt the notions and the methods used in software quality to assess the data set quality. Second, we formally define two quality dimensions, namely, Queriability and Informativity from the perspective of quality in use. The two proposed dimensions correspond to querying and answering, respectively, which are the most frequent usage scenarios for accessing LOD data sets. Then we provide a series of metrics to measure the two dimensions. Last, we apply the metrics to two representative data sets in LOD (i.e., YAGO and DBpedia). In the evaluating process, we select dozens of questions from both QALD and WebQuestions and ask a group of users to construct queries as well as to check the answers with the help of our usability testing tool. The findings during the assessment not only illustrate the capability of our method and metrics but also give new insights on data quality of the two knowledge bases.

1 Introduction

In recent years, an increasing number of semantic data sources are published on the web. These sources are further interlinked to form Linking Open Data (LOD).

This work was partially supported by the 863 plan of China Ministry of Science and Technology (project No: 2015AA020107), and Software and Integrated Circuit Industry Development Special Funds of Shanghai Economic and Information Commission (project No: 140304).

H. Sack et al. (Eds.): ESWC 2016, LNCS 9678, pp. 52–68, 2016.
DOI: 10.1007/978-3-319-34129-3_4

They include not only encyclopedic knowledge bases (KBs) such as DBpedia and YAGO, which serve as data hubs, but also domain-specific LOD data sets such as DrugBank[1] and DailyMed.[2]

For a user who wants to utilize existing KBs, it is a demanding task to know their qualities. Their quality can be measured in various ways. A systematic review of the different approaches for assessing the data quality of LOD can be referred to [1]. The authors summarized 68 metrics and categorized them into four dimensions, namely, *Availability*, *Intrinsic*, *Contextual* and *Representational*. Glenn and Dave[3] listed 15 metrics to assess the quality of a data set. The metrics include *Accuracy, Completeness, Typing*, and *Currency*, etc.

While the metrics proposed in literature could measure different characteristics of a data set, these metrics neither take enough users' point of view into consideration, nor do they measure the "usability" of the data set. Despite that most studies [1–3] agree with the opinion that data quality is "fitness for use in special application context," no research works have proposed quality models or metrics related to this definition. In contrast to the state of the art of LOD usability research, software usability is well studied and has mature models and metrics. Since the definitions of software usability in traditional research do not distinguish different usage contexts, the ISO/IEC 25010 (2011) broadened the concept of software usability with *quality in use*. In the new standard, software quality contains three factors, which are internal quality, external quality, and quality in use. Quality in use is measured from the users' point of view and is obtained from using the software in the working environment.

In this paper, we propose metrics and methods to evaluate quality in use of data sets. We use the concept *quality in use* instead of usability. The reason is that usability is usually used to measure the user interface design, and a data set may not provide any user interface. However, a data set without a user interface can still be utilized in different contexts so that we call how easy a user utilizes a data set *quality in use*. The most common usage scenario of LOD data set is to access the information returned by executing a query. Therefore, we propose two quality dimensions, namely, Queriability and Informativity. Queriability measures how easily an end user can construct a correct query on a data set. Informativity shows how informative a data set is under a particular usage context. Furthermore, we define three metrics *Query Construction Time*, *Number of Attempts*, and *Difficulty Rating* to measure Queriability, and we also use *Precision*, *Recall*, *Comprehensive Informativity*, and *Informativeness Rating* to measure Informativity.

To investigate the effectiveness of our method, we carried out a few evaluations on DBpedia and YAGO. The two encyclopedic knowledge bases contain a large amount of instances or entities distributed in multiple domains. They are not designed for specific purposes of a particular group of users and are widely applicable theoretically. Therefore, the quality in use in different usage contexts

[1] http://www.drugbank.ca/.
[2] http://dailymed.nlm.nih.gov.
[3] http://lists.w3.org/Archives/Public/public-lod/2011Apr/0145.html/.

is very important for the spreading of these knowledge bases. We choose questions from two standard Q&A (questions and answers) test sets, namely, QALD and WebQuestions as query contexts and ask a group of users to construct queries complying with these questions and check the results with the answers in the test sets. We also develop a GUI tool to help users to construct queries in case they are not familiar with the SPARQL syntax. Our evaluation leads to a few interesting findings. For example, DBpedia has too many similar properties with different names, which greatly degrade the Queriability and Informtivity of DBpedia. The number of classes in YAGO is so large, which makes it difficult for evaluators to find the suitable domain in the query.

The paper is organized as follows: Sect. 2 introduces related work. Section 3 provides a quality model on LOD as well as the definitions of the metrics. Section 4 proposes our quality assessment process and related tools. Section 5 analyzes the results of the evaluation. Section 6 gives a conclusion and points out the future direction of our work.

2 Related Work

Our work focuses on devising new metrics and assessing KBs such as DBpedia and YAGO with these new metrics. Therefore, we survey literature regarding *metrics on LOD* and *Quality Evaluation on DBpedia and YAGO*. Since we learned usability assessment methods and metrics from the area of software usability, we would also give a brief introduction on software usability and quality in use.

2.1 Metrics on LOD

Besides the systematic review of approaches for accessing the data quality of LOD, there exist a lot of researches focusing on evaluating particular aspects of LOD quality. Labels are considered as an important quality factor of LOD in [4], and the authors introduced a number of related metrics to measure the completeness, accessibility, and other quality aspects of labels. Zhang et al. [5] designed a few complexity metrics on web ontologies. Gueret et al. [6] focused on assessing the quality of links in LOD. They assumed that unsuitable network structures are related to the low quality of links. Farber et al. [7] gave a survey on major cross-domain data sets of LOD cloud. They compared DBpedia, Freebase, OpenCyc, Wikidata, and YAGO from 35 aspects including schema constraints, data types, LOD linkages, and so on. However, they used natural languages and checklists instead of quantitative metrics to describe the special characteristics of data sets. While there are a lot of metrics on LOD, they do not measure the quality from the user's usability point of view.

2.2 Quality Evaluation on DBpedia and YAGO

Quality evaluations and quality improvements on encyclopedic KBs became a hot research topic recently. Zaveri et al. [8] classified the errors in DBpedia into

four dimensions, including Accuracy, Relevancy, Representational-Consistency and Interlinking. YAGO2 (a version of YAGO) [9] used statistic sampling with *Wilson score interval* to reduce human efforts when evaluating the correctness of the YAGO2 manually. Wienand et al. [10] detected incorrect numerical data in DBpedia using unsupervised numerical outlier detection methods. Paulheim and Bizer [11] added missing type statements and removed faulty statements in both DBpedia and NELL using statistical distributions. Kontokostas et al. [12] presented a methodology for test-driven quality assessment which automatically generated test cases based on predefined test patterns. While there is quality evaluation work on encyclopedic KBs, the assessment work on YAGO and DBpedia is mostly focused on internal qualities such as correctness.

2.3 Software Usability and Quality in Use

There are various standards and models defining software usability [13,14]. The GE model is one of the earliest work working on software quality by Mccall et al. This hierarchical quality model consists of 11 quality factors, 25 quality criteria, and 41 quality metrics. Usability is a quality factor in the GE model and relates to 3 quality criteria. Usability is the second factor in the FURPS+ quality model adopted by Rational Software.

While usability is defined as a characteristic of products in traditional approaches, more recent researches find that the required quality attributes vary according to usage scenarios [15]. As the example in [15], the required quality attributes of a text editor for a programmer should be different from those for a casual user. Therefore, the quality model in the revised ISO/IEC 9126 [16] and later in ISO/IEC 25010 [17] distinguishes three quality approaches: internal quality, external quality, and quality in use. Internal quality concerns the static properties of the code. External quality can be obtained by executing the software system. Quality in use is measured from the users' view and shows whether the user is satisfied with the software in the working environment. The three components are interrelated. The internal software attributes will determine the quality of a software product in use in a particular context.

In this paper, we adapt the notions of quality in use to the research area of data quality. Our metrics such as *Query Construction Time* and *Number of Incorrect Tempts* are highly inspired by the task-oriented metrics in [18].

3 Metrics Design

3.1 Quality Model

Inspired by the software quality model in ISO/IEC 9261, we also divide the quality of LOD into three factors, namely, internal quality, external quality, and quality in use. We further map the quality dimensions of linked data mentioned in [1] into internal and external factors in our model. The Queriability and Informativity dimensions are put into quality in use, as shown in Fig. 1.

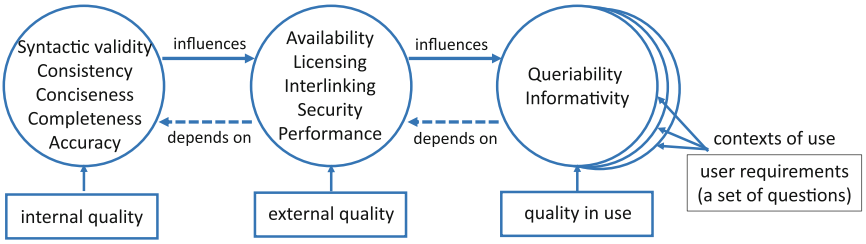


Fig. 1. The quality model of linked data

– Quality in use relates to the notion "context of use", which sometimes refers as usage context or usage scenario. Context of use implies user requirements. Users provide their data requirements in the form of a set of questions in our paper, just like user requirements are modeled as a set of use cases in software engineering. In the evaluation process, user-oriented questions are converted into data set oriented queries, the queries are executed on the data set, and the results are returned. Both the converting process and the results are measured by quality in use metrics. The whole quality evaluation process is requirement-oriented, conforming to the data quality definition "fitness for use in special application context". As a result, the metrics results depend on the requirements.
– The internal quality of a KB influences the quality in use of a KB. For example, the *accuracy* in Fig. 1 is categorized as internal quality. If there are errors in the triples, users may find the query results less informative.
– The external quality of a KB influences the quality in use of a KB. For example, the availability in Fig. 1 is categorized as external quality. Both DBpedia and YAGO have SPARQL endpoints, but the exploring service of YAGO[4] on the web can be accessed easily, which makes it easier for users who are not familiar with SPARQL to find the data.

3.2 Analyzing the Process of Constructing a Query

We designed an experiment to investigate the processes of constructing SPARQL queries for different evaluators on different data sets with different questions. In the experiment, evaluators assessed Queriability and Informativeness manually with the source files of data sets. We selected ten questions from WebQuestions and QALD and asked five graduate students to construct queries of the ten questions on both DBpedia and YAGO. Each evaluator wrote down his steps in constructing the query. After we checked all the step flows in their reports, we give a summarization, and the detail of the experiments can be accessed via our web sites[5].

[4] https://gate.d5.mpi-inf.mpg.de/webyago3spotlx/Browser.
[5] http://kbeval.nlp-bigdatalab.com/qiu.html.

1. Analyze each question and find the patterns of the question. For example, the question "what was Abe lincoln's wife name?" contains a subject and its property, and the object is the answer. But the question "Give me all female Russian astronauts" is much more complex. The pattern may contain a target domain (astronauts or Russian astronauts) and one or two constraints (female and Russian).
2. Find suitable vocabularies in the KBs. The vocabularies include domain names, property names, property values as well as instance names. We call the step to find domain names the "domain selection" step. As property names and properties values are related, the sub-step to find them are adjacent in our experiment records. They are combined together as the "property constraint selection" step. We find "domain selection" step is usually before "property constraint selection" step. Among 100 SPARQL constructing records (10 questions*5 evaluators*2 data sets), there is only one record in which a property name instead of a domain name is first found. The two steps are time-consuming (15 min on average) and evaluators may try many times before success. For example, the "wife" may not be the property name in a data set. It could be "married to" or something else. What's worse, the domain could be "Russian astronauts" instead of "astronauts," and its property names in the question are vacant.
3. Construct the query using SPARQL syntax and execute the query.
4. Repeat step 1–3 if the results are not desirable.

While there exist questions which are so simple that some steps may not be required, the above steps are unavoidable in general. The above steps contain all the possible steps in constructing a query, which give us guidelines on developing the evaluation tool. Since major difficulties arise from the "domain selection" step and the "property constraint selection" step, we design special metrics for the two steps in Sect. 3.3. Since we target at evaluating KBs and we do not want to bother evaluators on question understanding and syntax of SPARQL, we should eliminate the difficulties in steps 1 and 3. We develop a GUI tool which help users construct a SPARQL query interactively. The functions in the tool have direct mappings with the steps above in the manual construction process. Thus, evaluators can construct queries in the same process with the same results manually as in the tool and vice versa, as described in Sect. 4.3.

3.3 Metrics

The Queriability and Informativity focus on the process of query and the results of the query, and the corresponding metrics are shown in Table 1. Queriability measures how easily an end user can construct a correct query on a data set. Informativity shows how informative a data set is under a particular usage context.

3.3.1 Queriability

We design two kinds of Queriability metrics: one is the subjective metrics which are collected from direct feedbacks of evaluators, and the other is the objective

Table 1. Metrics definitions

Dimension	Metric	Description
Queriability	Query construction time on domain	Time (T_a) spent on setting the domain of the query $T_a = \frac{1}{NOA}\sum_{i=1}^{NOA} T_{ai}$ (T_{ai} is the time spent on setting the domain of the attempt i)
	Query construction time on property constraint	Time (T_b) spent on setting the properties and property values of the query $T_b = \frac{1}{NOA}\sum_{i=1}^{NOA} T_{bi}$ (T_{bi} is the time spent on property constraint setting of the attempt i)
	Query construction time	Time (T) spent on constructing the query $T = NOA(T_a + T_b)$
	Number of attempts	Times (NOA) tried for constructing the query
	Difficulty rating	Users' rating on the difficulties on constructing the query
Informativity	Precision	The precision (P) of the results for the query $P = NCA/A$ (NCA is the number of correct results, and A is the number of query results)
	Recall	The recall (R) of the results for the query $C = NCA/NA$ (NA is the number of standard answer for the query)
	Comprehensive informativity	Comprehensive Informativity (CI) is a comprehensive metric to measure the informativity of the answer
	Informativeness rating	Users' rating on the information that the results contain

metrics which are collected in the process of constructing a query. *Difficulty Rating* in Table 1 is a subjective metric. After evaluators finish (may be success or fail) constructing a query, they give ratings on how difficulty the process was. The rating has five levels: (1) very easy, (2) easy, (3) average, (4) difficult, and (5) very difficult. Objective metrics are designed according to the process of constructing a query. There are two important aspects in query constructing: time spent in constructing a query and how many times a user has tried. We use *Query Construction Time On Domain* T_a and *Query Construction Time On Property Constraint* T_b to measure the former, and we use *Number of Attempts* NOA to measure the latter. Then, *Query Construction Time* $\mathrm{T} = \mathrm{NOA}(\mathrm{T}_a + \mathrm{T}_b)$.

Based on the analysis of the query construction process in Sect. 3.2, constructing a query typically consists of two steps. One step is to set the domain of the question, and the other step is to set the property constraints. T_a is closely linked to the taxonomy system of the KB. Both the complexity of the taxonomy

system and the specificity of the vocabularies of the classes may lead to larger T_a. T_b is closely linked to properties used in the KB. The redundancy of the properties and the ambiguity of the properties may influence the time T_b. In general, the longer the time spent, the harder the process is.

A user may have tried many incorrect queries before he/she successfully constructed a query. If a user finds nothing returned or the returned results are wrong after he executes a query, he may reconstruct a query. For example, for the question "Which presidents were born in 1945?" if the property is set as "wasBornIn" on YAGO, nothing will be returned because the range of the property "wasBornIn" is the city, and the correct property should be "wasBornOnDate." There may be extreme conditions that whatever the user tries, he/she does not get the intended answer since the KB does not contain the answer. The whole process fails. In whatever situation, the *Number of Attempts* reflects the quality of the KBs. The larger the number is, the more difficult to construct a query. Since we limit the maximum number to 10 so that the value of *Number of Attempts* is an integer between 1 to 10.

3.3.2 Informativity

We also design two kinds of Informativity metrics as the Queriability metrics above. *Informativeness Rating* in Table 1 is the subjective metric, and it is the users' rating on the informativeness that the results contain. The *Informativeness Rating* has five levels: (1) very little information, (2) little information, (3) some information, (4) a fair amount of information, and (5) lots of information. The objective metrics are computed according to the standard answers of the questions, and we use *Precision, Recall, Comprehensive Informativity* to measure the query results.

Precision and *Recall* are used to measure the query results in the Informativity dimension. Precision is the fraction of returned results that are relevant, and it measures the correctness of the query results. Recall is the fraction of relevant instances that are returned with querying, and it measures the completeness of the query results. *Comprehensive Informativity* (CI) is a metric which integrates different factors that influence the users' comprehension of information. From our understanding of information comprehension, the factors should include not only precision and recall of a query results but also the accuracy of data returned as well as the understandability of the data. Therefore, the formula is

$$CI = \frac{NCA}{NA} * (\frac{NCA}{A})^2 * \alpha * \beta \tag{1}$$

NCA is the number of correct answers of the query results. NA is the number of standard answers for the question. A is the number of query results. The range of CI is [0,1]. The square function is used to punish the irrelevant answers. The data in the KBs may be inaccurate. In our previous work [19], we use the *correctness ratio of facts* metric to measure the accuracy. Here, we use α to denote the metric result. In the process of calculating the CI, we just directly reuse results in [19] to α. β is the understandability of the data in the KB, and

it measures whether the data of the KBs is human-readable. We do not measure the dimension in this paper, so we just set it to a constant 0.8.

4 Experiment Design

4.1 Evaluated Knowledge Bases

We applied our metrics to DBpedia and YAGO in our experiment. DBpedia and YAGO are two major general-purpose knowledge bases serving as the hubs of Linked Open Data. In particular, we chose two representative versions (DBpedia2014 and YAGO2S) for DBpedia and YAGO, respectively.

4.2 Questions and Patterns

Data requirements of users are modeled as question sets, as mentioned in Sect. 3.1. Here we use the following criteria to collect questions which meet the requirements for a good "wikipedia like data set".

1. Biases on either data set should be avoided. We utilized two question sets in the KB-based QA area. One source is the Question Answering over Linked Data[6] (QALD), and the other is WebQuestions[7] from the NLP laboratory of Standford. We choose QALD-2 whose questions are cross-domain. Because the number of questions in WebQuestions is huge, we chose 50 questions from the beginning, 50 questions in the middle, and 50 questions at the end.
2. Questions having no answers in either DBpedia or YAGO are avoided.
3. The diversity of question patterns is considered. For example, "what are the official languages in spain?" and "what is the official language spoken in mexico?" are similar, therefore, we only choose one of them.

At last, we chose 13 questions from QALD and 13 questions from WebQuestions. A full list of 26 questions, 150 questions from WebQuestions, and 100 questions from QALD could be found on our website.[8]

As explained before, our intention is to assess data set quality instead of SPARQL syntax. We should try our best to eliminate the time that the evaluators spend on SPARQL syntax. To meet that goal, we analyze the questions in QALD and WebQuestions and find most of them (the detail statistics are also on our website mentioned above) can be categorized to special patterns shown in Table 2. The patterns in Table 2 cover all the 26 questions in the list. We build a tool to support these patterns so that users will just fill in the appropriate vocabularies and operators to instantiate the corresponding pattern, and the executable SPARQL query is generated automatically.

[6] http://greententacle.techfak.uni-bielefeld.de/~cunger/qald/index.php?x=home\&q=home.
[7] http://nlp.stanford.edu/software/sempre/.
[8] http://kbeval.nlp-bigdatalab.com/qiu.html.

Table 2. Question patterns and SPARQL query patterns

Pattern	Transformation	Example
DomainPattern	Find all sub classes	Give me all school types
PropertyPattern	Select * where { ?s propertyname ?o }	Population of cities
InstancePattern	Select * where { instancename ?p ?o}	Information of Google
ValuePattern	Select * where { instancename propertyname ?o}	What was abe lincoln's wife name?
AttributeEqual	Select * where { ?s attributename attributevalue }	Which presidents were born in 1945?
AttributeRange	Select * where{ ?s attributename ?o filter(?o operator value) }	The cities whose population bigger than 3 million
AggreExpression	Select ?s where {?s attributename ?o} group by ?s having(aggreateFuncname(?o) operator value)	Which countries have more than two official languages?
OrderedTop	Select ?s where {?s attributename ?o} order by ?o	What is the highest mountain?
OrderedTop + AggreExpression...	Select ?s where {?s attributename ?o} order by aggreateFuncname(?o)	Who produced the most films?

4.3 Evaluation Process and Evaluation Tool

We design a tool to support the manual process in Sect. 3.2, as Fig. 2 shown. Each evaluator first chooses a question from the question list and selects a target KB (YAGO or DBpedia). He then sets the domain of the question and inputs property constraints to construct a corresponding query. The tool is shown in Fig. 3. After clicking the "execute" button, the tool constructs and executes the SPARQL query automatically, and the results are returned to the evaluator.

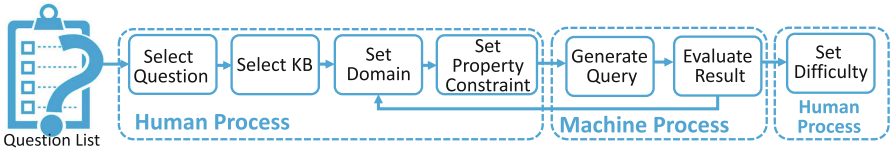


Fig. 2. Evaluation process

The vocabulary finding step in manual constructing process requires too much human repeated laboring work, for example, searching for a property name in an editor with the search function while the occurrences of the name is large, or find all sub domains. Since these repeated laboring work may obscure our mission of finding usability problems in the data sets, we provide some user-friendly functions to relieve the evaluators from these work. All the functions

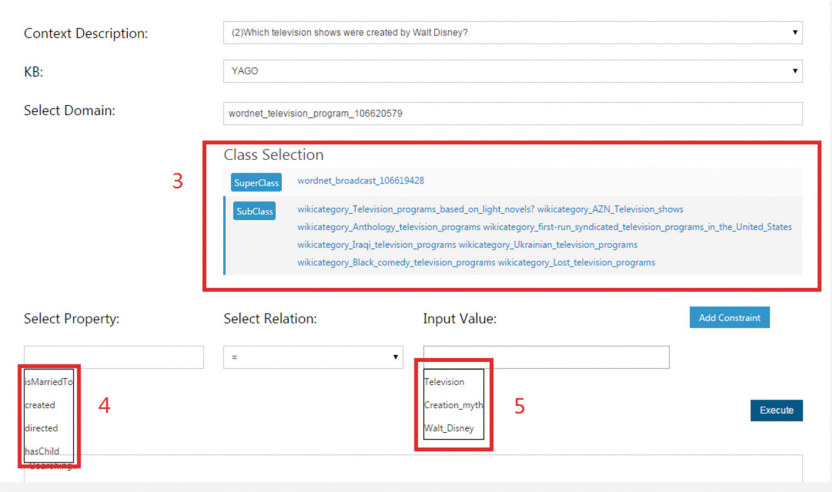


Fig. 3. Our evaluation tool

are just suggestions to help user find vocabularies, and it is the evaluator who ultimately chooses the vocabularies and determines the SPARQL query to be executed.

1. Autofill functions. When users or evaluators input the name of a domain or a property name, all the class names or property names containing the input letter sequence would be popped up as suggestions. The function is a replacement of the general search function in the editor in the manual construction process.
2. Show the subclasses and superclasses of a domain. In the manual construction process, the evaluator may find appropriate domain names by searching the subclasses or superclasses of existing class. For example, a user needs to select the domain of the question "Give me all presidents of the United States." The user may first come up with "President," and then he may find a subclass "Presidents_of_the_United_States" which may be closer to the question. Therefore, we provide the function which shows both subclasses and superclasses of a domain.
3. PATTY [20] is integrated in our tool to obtain the candidates of a property. We can use the relation patterns provided by PATTY to get the properties contained in the sentence. For example, the question is "Which television shows were created by Walt Disney?" and the candidates are "isMarriedTo," "created," "directed," "hasChild." While candidates provided by PATTY often have nothing to do with the question, the function broadens users' conjectures on possible properties names.
4. DBpedia Spotlight [21] is integrated in our tool to annotate the instances in the question. For the above question, DBpedia Spotlight can annotate the

instance "http://dbpedia.org/resource/Walt_Disney." This function can be considered as an advanced search function for instances.

After executing the query, the results may be empty or are not the ones as the users expect. Users need to construct another new query until the results are satisfactory or the number of attempts achieves the maximum limit. Each question in the question list has a standard answer set as ground truths, so the metrics in the Informativity dimension can be computed. Finally, users should set the *Difficulty Rating* according to their evaluation process and set the *Informativeness Rating* according to the query results after the evaluation for each question. Since the entire evaluation process includes several manual steps, in order to reduce errors caused by the subjectivity of the users, we employ eight people, and each of them tested all the 26 questions.

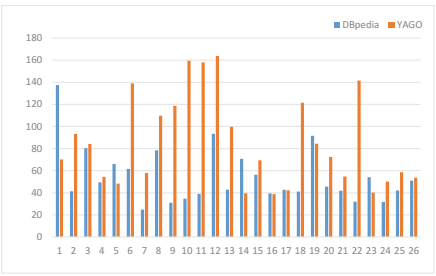
5 Evaluation Results

5.1 Queriability of DBpedia and YAGO

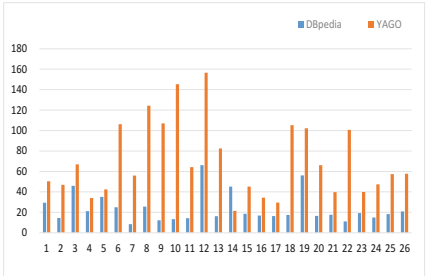
The comparison results of two KBs with respect to the average *Query Construction Time*, *Query Construction Time on Domain*, as well as the *Query Construction Time on Property Constraints* for every question are shown in Fig. 4(a–c), respectively. The means and variations for these metrics are shown in Fig. 4(d).

We find in Fig. 4(a) that it costs evaluators more time on YAGO than on DBpedia to find a satisfactory query. We look into detail Fig. 4(b) and Fig. 4(c). From Fig. 4(b), we find that when users select domains, they spend much more time on YAGO than on DBpedia. This is because YAGO contains a huge number of classes, and some class names are excessively long. As mentioned in Sect. 4.1, YAGO has 451k classes. The length of some class names may exceed 50 characters. For example, the category name wikicategory_Failed_assassins_of_United_States_presidents is really difficult for users to read or input. On the contrary, classes of DBpedia are fewer, and the names of the classes are easier to understand. From Fig. 4(c), we find that when considering properties selection, it takes longer on DBpedia than on YAGO. Compared with the number of properties defined in YAGO (i.e., 75), DBpedia has more than 55,000 properties. Moreover, there exist a lot of nearly duplicated properties, which lead to more effort on selecting properties. For example, the property names "dateOfBirth," "birthDate," "birth," "birthdate," and "birthday" in DBpedia are all related to the notion "birthday."

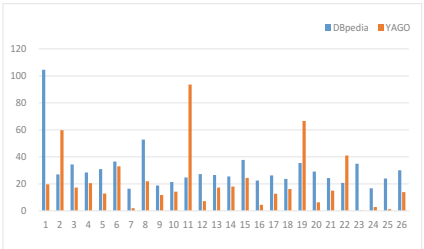
Figure 4(d) shows that when users query on YAGO, *Query Construction Time On Domain* and *Query Construction Time On Property Constraint* have a larger fluctuation. The fluctuation of domain selection results from the differences of the classes in YAGO. Classes with relatively simple names which are close to the root of the taxonomy hierarchy are easier to be found, such as "wordnet_actor_109765278." Classes with long names which are close to the leaf level of taxonomy hierarchy are difficult to find. The reason for the fluctuation of



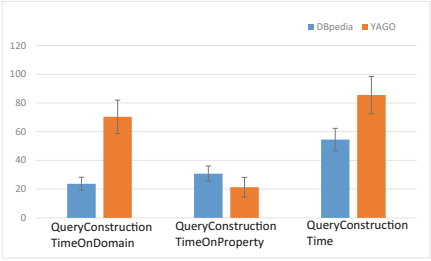
(a) The average Query Construction Time



(b) The average Query Construction Time On Domain



(c) The average Query Construction Time On Property Constraint



(d) The mean of Query Construction Time

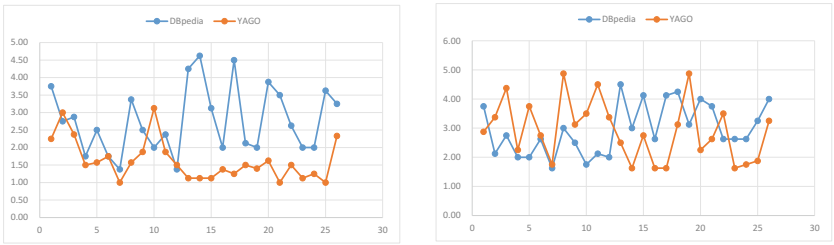
Fig. 4. Results of query construction time

property selection is that, for simple queries, users only need to set the domain, and then the *Query Construction Time On Property Constraint* is 0. However, if the selected domain is not suitable, users have to spend a lot of time to switch between similar properties.

The comparison results of two KBs with respect to the average *Number of Attempts* are shown in Fig. 5(a). In general, the *Number of Attempts* in DBpedia is more than that in YAGO. The conclusion becomes more obvious for the second half of the questions. After investigation, we find that in the beginning, users are not familiar with YAGO classes. They find wrong classes and try to change the properties to construct a query, which leads to a number of failed attempts. In the later stages, users have a better understanding of the YAGO taxonomy and are aware that they should rely more on it.

The comparison results of two KBs with respect to the average *Difficulty Rating* are shown in Fig. 5(b). Even with the help of our tool, no question is rated by users as easy on average, whether for DBpedia or for YAGO. Figure 5(b) also shows a tendency that in the first-half questions, DBpedia is easier, while in the second half, YAGO is easier.

We also calculate the correlation between the *Difficulty Rating* and the other Queriability metrics. We find that *Difficulty Rating* closely correlates with *Number of Attempts* in DBpedia and closely correlates with the sum of *Query Construction Time on Domain* and *Query Construction Time On Property Constraint* in YAGO. For DBpedia, it does not cost much time to set the



(a) The average Number of Attempts (b) The average of Difficulty Rating

Fig. 5. Results of number of attempts and difficulty rating

domain and property. However, duplication exists in the property names. When users enter wrong property names with no query result, they have to try other property names. In this case, users have to try many times before being successful, and they feel confused and have a difficult time. That's the reason the *Difficulty Rating* correlates with *Number of Attempts* in DBpedia. For YAGO, it costs time to select the appropriate domain and property, especially when the target class is complex. That's the reason why *Difficulty Rating* correlates with the total time.

5.2 Informativity of DBpedia and YAGO

In terms of Informativitiy, we focus on the amount of information a user can gain under a certain context, and we do not care about how many incorrect queries are constructed by users. Therefore, we choose the best result achieved by all evaluators for each question to assess Informativity. In general, there are 20 questions that return at least one answer in YAGO and 14 questions in DBpedia. The comparison results of two KBs with respect to the average *Precision* and *Recall* as well as the *CI* for every question are shown in Fig. 6. All of them show that YAGO has better Informativity than DBpedia.

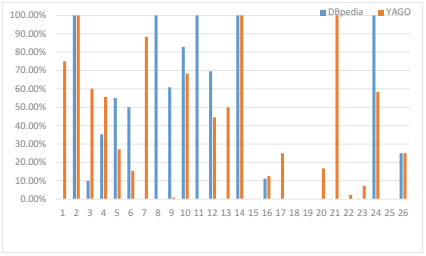
From Fig. 6(a), we find DBpedia has a rather low precision. One reason is the polysemy of type words. For example, presidents can be presidents of countries or presidents of organizations. So for the question "Give me all presidents of the United States," we found that the results include *the Chairman of the Federal Reserve* and other types of presidents. The low precision of YAGO arises from our misinterpretation of the vocabulary in YAGO. For example, the class "wikicategory_German_actors" means actors whose nationality, not birthplace, is German.

From Fig. 6(b), we find DBpedia and YAGO have low recalls. The reason for DBpedia is that it has duplicated properties as mentioned before. The reason for YAGO is that it really does not contain abundant properties with enough facts. For the question "Which actors were born in Germany?" if the users set the domain as "wordnet_actor_109765278," set the property as "wasBornIn," and set its value as "Germany," no results are returned. For the question "who is

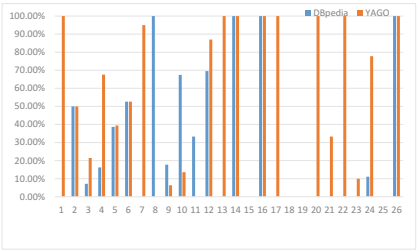
number 5 on the Boston Celtics?" evaluators cannot find the property in YAGO to describe "number" in the context.

Informativeness Rating before and after the user checked the correct answers are different. The former relates to how many times users attempt to construct a query and how many results are returned. The latter relates to the precision and the recall, and relates to CI as a whole.

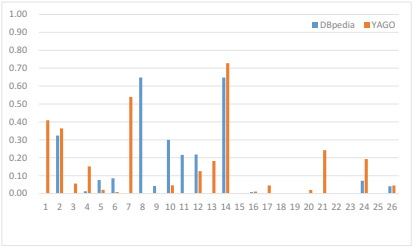
In summary, the results on Queriability and Informativity relate to internal characteristics of KBs, namely, the schema design and the richness of the data. For YAGO, the huge class hierarchy increases difficulties in finding the classes, the small number of properties reduces the "expressiveness" of the data set, and the smaller number of facts makes query results less informative. While DBpedia seems to have a better balance between the number of classes and the number of properties, the property duplication problem largely decreases the quality in use of DBpedia.



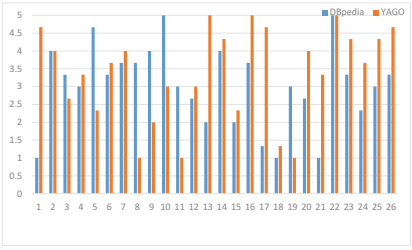
(a) Precision



(b) Recall



(c) Comprehensive Informativity



(d) Informativeness Rating

Fig. 6. Informativity of DBpedia and YAGO

The lessons learned from our assessment work include: (1) Naming convention for classes, objects and properties are required in the LOD world, similar to that in software (2) Duplicate property names should be avoided since they will mislead the users. (3) There should be a tradeoff between the number of classes and the number of properties. Users may encounter difficulties when both of the numbers become too large. In general, our experiments show that a well-designed

schema is very important for people to utilize a data set. We wish someone could run a "linked open schema (LOS)" website which contains more schema data, vocabularies and constraints than existing schema web sites such as schema.org. Every data set should be linked and registered to the LOS websites before the data set is published. There could be facilities such as Q&A engines on this LOS site so that every data set can be accessed via natural language interfaces. In this way, the data sets published will be of higher quality, and end users could utilize the data set immediately.

6 Conclusion and Future Work

In this paper, we designed two metric sets, namely, Queriablity and Informativity with respect to the "quality in use" factor on LOD. The metric results on YAGO and DBpedia not only show that users have experienced difficulties in utilizing these KBs, but also give many hints on where the difficulties arise as well as how to improve these KBs. In the future, we plan to assess other usage scenarios of quality in use, such as search and browsing or Q&A, and assess more cross-domain KBs such as Wikidata. We also plan to collect data requirements on the medical domain and evaluate the quality in use of medical data sets of the LOD cloud.

References

1. Zaveri, A., Rula, A., Maurino, A., Pietrobon, R., Lehmann, J., Auer, S.: Quality assessment methodologies for linked open data. Semant. Web J. (2012)
2. Strong, D.M., Lee, Y.W., Wang, R.Y.: Data quality in context. Commun. ACM **40**(5), 103–110 (1997)
3. Tayi, G.K., Ballou, D.P.: Examining data quality. Commun. ACM **41**(2), 54–57 (1998)
4. Ell, B., Vrandečić, D., Simperl, E.: Labels in the web of data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 162–176. Springer, Heidelberg (2011)
5. Zhang, H., Li, Y.F., Tan, H.B.K.: Measuring design complexity of semantic web ontologies. J. Syst. Softw. **83**(5), 803–814 (2010)
6. Guéret, C., Groth, P., Stadler, C., Lehmann, J.: Assessing linked data mappings using network measures. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 87–102. Springer, Heidelberg (2012)
7. Färber, M., Ell, B., Menne, C., Rettinger, A.: A comparative survey of DBpedia, freebase, opencyc, wikidata, and yago
8. Zaveri, A., Kontokostas, D., Sherif, M.A., Bühmann, L., Morsey, M., Auer, S., Lehmann, J.: User-driven quality evaluation of DBpedia. In: Proceedings of the 9th International Conference on Semantic Systems, pp. 97–104. ACM (2013)
9. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: Yago2: A spatially and temporally enhanced knowledge base from wikipedia. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence, pp. 3161–3165. AAAI Press (2013)

10. Wienand, D., Paulheim, H.: Detecting incorrect numerical data in DBpedia. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 504–518. Springer, Heidelberg (2014)
11. Paulheim, H., Bizer, C.: Improving the quality of linked data using statistical distributions. Int. J. Semant. Web Inf. Syst. (IJSWIS) **10**(2), 63–86 (2014)
12. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven evaluation of linked data quality. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 747–758. ACM (2014)
13. Al-Qutaish, R.E.: Quality models in software engineering literature: an analytical and comparative study. J. Am. Sci. **6**(3), 166–175 (2010)
14. Seffah, A., Donyaee, M., Kline, R.B., Padda, H.K.: Usability measurement and metrics: a consolidated model. Softw. Qual. J. **14**(2), 159–178 (2006)
15. Bevan, N., Azuma, M.: Quality in use: incorporating human factors into the software engineering lifecycle. In: Third IEEE International Software Engineering Standards Symposium and Forum, Emerging International Standards, ISESS 1997, pp. 169–179. IEEE (1997)
16. ISO/IEC: ISO/IEC 9126-4 Software engineering -Product quality- part4: Quality In Use metrics (2002)
17. ISO/IEC25010: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models (2011)
18. Albert, W., Tullis, T.: Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics. Newnes, Oxford (2013)
19. Ruan, T., Dong, X., Wang, H., Li, Y.: Kbmetrics - a multi-purpose tool for measuring quality of linked open data sets. In: The 14th International Semantic Web Conference, Poster and Demo Session (2015)
20. Nakashole, N., Weikum, G., Suchanek, F.: Patty: a taxonomy of relational patterns with semantic types. In: Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, Association for Computational Linguistics, pp. 1135–1145 (2012)
21. Daiber, J., Jakob, M., Hokamp, C., Mendes, P.N.: Improving efficiency and accuracy in multilingual entity extraction. In: Proceedings of the 9th International Conference on Semantic Systems (I-Semantics) (2013)