International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 3 (2011) pp. 780 -787

Integrating Knowledge of City Entities Extracted from DBpedia and GeoLite into the EKOSS Failure Cases Repository to Enhance Semantic Search Capabilities*

Weisen Guo[1] and Steven B. Kraines[1]

[1] Science Integration Program (Human), Department of Frontier Science and Science Integration, Division of Project Coordination, the University of Tokyo, Kashiwa, Japan
{gws, sk}@scint.dpc.u-tokyo.ac.jp

***Abstract*: Domain-specific repositories of manually created semantic descriptors usually contain detailed knowledge with "heavy-weight" semantics about particular aspects of the domain, but they often lack common knowledge about the concepts and entities that are described. Integrating some of this common knowledge into the repository can enhance the capabilities of semantic search in the repository. We use common knowledge about city entities from DBpedia knowledge base and GeoLite city database to enhance the EKOSS failure cases repository, which contains knowledge about failures in engineering. Custom parsers are used to extract common knowledge from the two open data sources, and special semantic descriptors, which we call standard entity statements, are generated from the extracted knowledge. We link the city instances in failure case semantic statements with the standard city entity semantic statements, and we demonstrate the new types of semantic search capabilities that are made possible by the integration of the three semantic resources.** *

***Keywords*:** EKOSS, DBpedia, GeoLite, Semantic Search, Logic Inference, Semantic Integration

I. Introduction

The Web is becoming one of main ways that human share knowledge. On the conventional Web, data have only human-readable semantics. However, more and more data sources are adding structure to their data, including semantic tags that are computer-readable. Structured data with semantics can be utilized by computers more intelligently than data having only human-readable semantics [2].

For example, in order to obtain more accurate results for some types of complex search queries [3], the EKOSS system uses Description Logics (DLs) [2] to create semantic statements in the form of DL ABoxes, each of which describes a knowledge resource, e.g. an abstract of a scientific article. The semantic statements are then loaded into a knowledge base for reasoning. Due to current limitations of natural language processing technologies, the semantic statements need to be created manually [2] or semi-automatically [4] in order to guarantee that they describe the domain-specific knowledge with rich and accurate semantics. However, even though DLs enable inference of implied relationships between entities to get more interesting results [5], the EKOSS knowledge base still can only answer a limited range of search queries.

One reason is that while a semantic statement always contains detailed domain-specific knowledge, it often omits common knowledge about related concepts and entities. This occurs because 1) the authors of the semantic statements leave out common knowledge that is known to most human readers and 2) the authors of semantic statements often cannot input all of the related information due to limitations of time and personal knowledge. For example, third-party curators entering data based on some text authored by another person, such as an abstract of scientific article, are generally limited to the specific content of the source text.

Human readers can associate expressions of domain-specific knowledge with related common knowledge that is already in their heads. But computers do not have this ability. Therefore, it is necessary to directly provide the computers with the related common knowledge that is needed to understand the whole idea conveyed by the knowledge resource.

Many data sources contain common sense knowledge. Some of them have broad domain coverage, such as DBpedia [6], [7], Freebase [8], and YAGO [9]. Some of them are more domain-specific, such as the DOAP dataset [10] that is used to describe software projects, the GeoNames [11] geographic database that is available and accessible through various Web services, the FOAF dataset [12] that is used to describe basic attributes of people and relationships among them, and the DBLP database [13] that provides bibliographic information on major computer science journals and conference proceedings. Several of these data sources have been linked with each other, and many more are planned to be linked in the future. These linked data sources contribute to a web of Linked Open Data (LOD) [14].

We can obtain common knowledge related to a particular

* A preliminary version of this paper was presented at NWeSP 2010 [1].

repository of domain-specific semantic statements from the available LOD or other open structured data sources. First, we must decide which kind of common knowledge should be added to the repository. Second, we need to find available data sources containing the required common knowledge. Third, we extract the common knowledge and convert it into the representation of the repository. Fourth, we enrich semantic statements with the related common knowledge.

The basic procedure is simple. If each step can be finished successfully, then the enriched repository should be able to answer more complex and interesting search queries. However, there are several concrete problems that must be addressed; we will discuss these in the following sections.

First, we explain the basic approach we have used to enrich the EKOSS failure cases repository with common knowledge of city entities extracted from DBpedia knowledge base, one of the main LOD data sources containing structured information extracted from Wikipedia [15], and GeoLite City database [16], which is another open structured data source containing geographic information.

EKOSS (Expert Knowledge Ontology-based Semantic Search) [2] is a web-based knowledge sharing system that enables users to create one or more computer-understandable descriptors, called semantic statements, that describe their knowledge resources, such as scientific articles, using OWL-DL [17] ontologies. EKOSS then uses a reasoner based on the RacerPro DL inference engine [18] to match a semantic query against the semantic statements. The EKOSS system has been used to create semantic statements for 291 failure cases selected from the failure knowledge database created by the Japan Science and Technology Agency (JST) [19]. These 291 semantic statements, hereafter "failure case statements", were created using the SCINTENG ontology [20], which provides a formal knowledge representation language and background knowledge in engineering.

We can do semantic searches of the EKOSS failure cases repository using both logical and rule-based inference [3], [20]. However, the EKOSS failure cases repository lacks common knowledge about the different entities. In particular, almost all of the failure cases occur in specific cities, but the common knowledge about the city where the failure occurred, such as population and location, is generally not specified. If such common knowledge about cities were made available, more useful searches would be possible.

We first attempted to extract the information about city entities from the DBpedia knowledge base only. However, upon inspection of the DBpedia data, we discovered that the accuracy of geographic information is somewhat low. Therefore, we use information from the GeoLite City database to replace the geographic information for the corresponding entity from DBpedia (Section II). Next, we use a set of templates to automatically generate a special semantic statement in the EKOSS system, called a standard entity statement, for each city entity extracted from DBpedia and GeoLite (Section III). The third step is to match each instance of the class **city** (we show class names in bold type) in the failure case statements with the standard entity statement that describes the equivalent city (Section IV). In Section V, we enhance the original EKOSS reasoner to support searches of the enriched repository. Our experiments show that we can get informative search results from the enriched repository for some complex search queries that did not obtain any results from the original repository. Finally, we conclude this paper with a discussion of related work and a summary.

II. Extracting Common Knowledge about City Entities from DBpeida and GeoLite

The aim of the work presented here is to add some common knowledge to the EKOSS failure cases repository. The SCINTENG ontology class **city** is used often in the repository, appearing 181 times. Therefore, we decided to extract information about city entities from the DBpedia knowledge base. We have used the DBpedia 3.4 datasets [7], which contain 41,062 city entities. The DBpedia 3.4 ontology has 166 properties whose domain can be an instance of the class **city**. However, considering the hierarchy of the property subsumption tree and the semantic meanings of the properties, we determined that just half (83) of the properties are semantically unique. Furthermore, we found that 14 properties were not used with any entities in the DBpedia knowledge base. The remaining 69 properties that are used to describe attributes of specific city entities in the DBpedia knowledge base are listed in Table 1 together with the number of times that they are used. The property counts in Table 1 include usages of the properties with entities other than city. For example, the property count for "areaTotal" includes usages for country entities as well as city entities.

Our goal is to add common knowledge about cities that could be useful for conducting semantic searches. Considering the meanings and content of the 69 properties, we selected 19 properties that are most likely to be useful for semantic searches of the EKOSS failure cases repository. The selected properties are marked with "Y" in the "Use" column in Table 1.

Population density is a property that is particularly relevant to the failure cases. Upon inspection of the data from DBpedia, we found that some cities do not have population density values even though they have an area value and a population value. Furthermore, even when a city has a population density value, that value does not always agree with the corresponding area value and population value. In order to keep the data consistent, we discarded the population density values extracted from the DBpedia knowledge base and recalculated them using the corresponding area value and population value whenever both values were available.

Name	Count	Use
Properties whose domain is http://dbpedia.org/ontology/City		
Day	1709	N
distanceToEdinburgh	5	N
scottishName	118	N
irishName	175	N
cornishName	108	N
federalState	9	N
associationOfLocalGovernment	2	N
administrativeCollectivity	65	N
Province	6611	N
Saint	1271	N
Frazioni	541	N
crownDependency	32	N
distanceToCardiff	21	N
distanceToLondon	16	N

gaelicName	582	N
welshName	272	N
manxName	23	N
Meaning	84	N
Department	17	N
administrativeDistrict	96	N
jointCommunity	7	N
Region	6481	N
Properties whose domain is http://dbpedia.org/ontology/PopulatedPlace		
areaMetro	910	Y
populationMetro	1,408	Y
populationTotal	146,216	Y
populationMetroDensity	157	Y
populationAsOf	104,147	Y
foundingDate	16,935	Y
areaUrban	245	Y
populationUrban	1,803	Y
populationDensity	91,741	Y
populationUrbanDensity	66	Y
foundingYear	82	Y
leaderTitle	26568	N
postalCode	75363	N
motto	2120	N
areaMagnitude	16	N
areaCode	62151	N
establishedTitle	20003	N
demonym	5035	N
censusYear	190	N
leaderParty	389	N
largestSettlement	10	N
language	1076	N
languageType	64	N
capital	4891	N
leaderName	13920	N
largestCity	424	N
regionalLanguage	225	N
ethnicGroup	1022	N
foundingPerson	47	N
Properties whose domain is http://dbpedia.org/ontology/Place		
areaTotal	136,042	Y
areaWater	71,851	Y
maximumElevation	921	Y
coordinates	202,766	Y
areaLand	72,886	Y
elevation	139,621	Y
minimumElevation	839	Y
location	66784	Y
nativeName	8112	N
nickname	2522	N
length	6892	N
width	2128	N
nearestCity	7107	N
otherName	6270	N
percentageOfAreaWater	4631	N
height	1071	N
depth	1822	N
type	61247	N

Table 1. The 69 properties that can be used with the class city in the DBpedia 3.4 knowledge base together with the number of usages and whether the property was used in our work.

The locations of cities are another useful form of information. The property "location" is used 66,784 times in DBpedia; however, it is used with city entities as domain entities only 20 times (most usages are with other place entities). Therefore, we looked for other open structured data sources with city location information. The GeoLite database [16] contains accurate information for cities, including the latitude, longitude, country code, city name, state/region, and population. Therefore, by mapping the city name of a GeoLite record to the name of a city entity in DBpedia, we can get its latitude and longitude coordinates as well as the country and state or region where it is located. Mapping cities that have unique matching names in each data set is trivial. For cities having non-unique names, we utilized the state/region information in GeoLite to disambiguate them. Most city entities in the DBpedia knowledge base have labels that include the state, prefecture, or province name as a suffix. For example, DBpedia contains a city entity labeled "Paris,_Texas". There are six cities with the name "Paris" in GeoLite. One of them has state/region "TX", which we can identify as "Texas" by using US and Canada ISO 3166-2 Subcountry codes [21] (for other countries we used the FIPS 10-4 code [22]). Therefore, we mapped the city named "Paris" with state/region "TX" in GeoLite to the city entity in DBpedia labeled "Paris,_Texas". We then used the country code to identify the country where the city is located. Also, the coordinates of cities in GeoLite are more comprehensive and accurate than in DBpedia, so even when DBpedia contained coordinates data for a particular city, we replaced that with the GeoLite coordinates data.

III. Generating Standard Entity Statements through Templates

The rationale for generating standard entity statements for each of the city entities is as follows.

A semantic search on the EKOSS system begins when the EKOSS reasoner, which has loaded the DL ontology TBox into its knowledge base, receives a search query. First, it loads one of the available semantic statements into its knowledge base as the ABox. The reasoner checks whether or not the search query can find a set of matching instances in the ABox. If a set of matching instances is found, those results are recorded. The reasoner then removes the ABox from the knowledge base and loads another semantic statement. This process is repeated until all available semantic statements are checked.

For each specific entity that has some common knowledge we would like to use, such as a city entity having location and population properties, we need to load that information into the knowledge base. If the common knowledge was added to each semantic statement, the knowledge would be duplicated many times, leading to problems in scalability and data maintenance. By just creating one standard entity statement containing all of the common knowledge for that specific entity, we avoid duplicating common knowledge imported into the system.

There is no direct mapping between the properties of the SCINTENG ontology and the properties of city entities in DBpedia. Therefore, we created several templates for converting the properties of city entities extracted from DBpedia to the properties of SCINTENG ontology, as shown in Table 2.

	Property extracted	Property of SCINTENG
1	***city*** location ***country***	*city* ***(city)*** has location *nation* ***(country)***
2	***city*** coordinates	*city* ***(city)*** has location

	[***double1***, ***double2***]	*point in space (latlong)* *point in space (latlong)* has latitude *coordinate (lat)* *coordinate (lat)* has value ***double1*** *point in space (latlong)* has longitude *coordinate (long)* *coordinate (long)* has value ***double2***
3	***city*** foundingDate ***string***	*city* ***(city)*** has start time *point in time* ***(string)***
4	***city*** foundingYear ***string***	*city* ***(city)*** has start time *point in time* ***(string)***
5	***city*** population* ***double*** ***city*** populationAsOf ***string***	*city* ***(city)*** has single property dimension with unit *number of persons value (population*)* *number of persons value (population*)* has value ***double*** *number of persons value (population*)* has possible individual *point in time* ***(string)***
6	***city*** area** ***double***	*city* ***(city)*** has area *area value (area**)* *area value (area**)* has value ***double*** *area value (area**)* has specific type of unit *square kilometer (square kilometer)*
7	***city*** population*Density ***double***	*city* ***(city)*** has single property dimension with unit *density value (population*Density)* *density value (population*Density)* has value ***double*** *density value (population*Density)* has specific type of unit *density unit (persons per square kilometer)*
8	***city*** elevation ***double***	*city* ***(city)*** has length *altitude (elevation)* *altitude (elevation)* has value ***double***
9	***city*** minimumElevation ***double***	*city* ***(city)*** has length *altitude (minimum elevation)* *altitude (minimum elevation)* has min value ***double***
10	***city*** maximumElevation ***double***	*city* ***(city)*** has length *altitude (maximum elevation)* *altitude (maximum elevation)* has max value ***double***

* indicates Urban, Metro, or Total.
** indicates Urban, Metro, Total, Land, or Water.

Table 2. The templates for converting the properties extracted from DBpedia and GeoLite into expressions in the SCINTENG ontology.

Each entry in the second column is in the form "***domain instance label*** propertyName ***range instance label or range value***". Each entry in the third column is in the form "*domain class name* (***instance label***) property name *range class name* (***instance label***)" for object properties, and "*domain class name* (***instance label***) property name ***range value***" for datatype properties. The instances with bold labels in the second and third columns having the same names are equivalent. The other instances in the third column are local: they have unique URI's that distinguish them from other instances having the same label but that are created by applications of the template to other cities.

If a city entity has both a population value and an area value for urban, metro, and/or total, we calculated the corresponding population density (population density = population / area) and then applied template 7 in Table 2.

The DBpedia 3.4 knowledge base has 7,107 entries of property "nearestCity" for place entities. However, we only found 6 entries whose domain and range are both city entities. To get more information about proximity of cities, for each city entity city1, we calculated the distances to all other cities by using the Haversine formula [23]. Then, for each city city2 having a distance of less than 100 kilometers to city1, we created the following two properties:

city (*city1*) has proximity **proximity** ("*within 100 km*").
city (*city2*) has proximity **proximity** ("*within 100 km*").

Using the above methods, we generated standard entity statements for 9,993 cities. Figure 1 shows a graph representation of the standard entity statement for Shanghai city. Boxes show instances of classes from the domain ontology, colored according to the major upper class: spatial locations are blue, events are pink, physical objects (not shown) are gray, activities (not shown) are yellow, and values are white. The text in each box above the line is the instance label and below the line is the class name of that instance. Arrows show properties expressing the asserted relationships between instances.

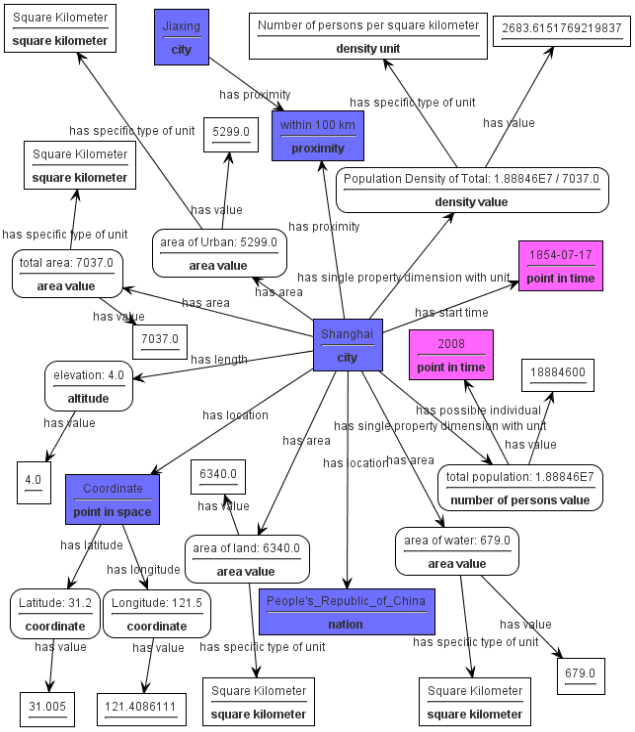

Figure 1. The standard entity statement for Shanghai city shown as a graph.

IV. Linking Failure Case Statements and Standard Entity Statements through Instances of City

Each failure case statement is populated by instances whose classes are known. Therefore, it is straightforward to find instances of the class city. However, the actual city that an instance of the class city represents is indicated by the label of the instance, and the instance labels are in free text. So we need to map each instance of city to a standard entity statement via the instance label. This is a problem of term normalization, which is usually solved by creating a lexicon.

We created a lexicon of city names based on the YAGO ontology [9]. There are two kinds of facts in YAGO that give

name information: "isCalled" and "means". For each city entity in DBpedia, we extracted these two kinds of facts from YAGO and created a set of synonyms for the city. These two kinds of facts are multilingual, which is useful for supporting cross-language knowledge sharing [24]. The YAGO facts contain other useful information, for example the fact that "CN-31" means the city of Shanghai, and "Chinese capital" means the city of Beijing. String matching [25] would be inadequate to link instances of city having such labels.

Using the lexicon of city names, we could link most of the instances of the class city in the failure case statements to the standard entity statements correctly. However, for some city instances in the failure case statements, no matching standard entity statements could be found. Some of those instances have complex labels that did not match the lexicon. Other instances are actually small towns or villages that are not in the lexicon. And of course even DBpedia does not include all cities in the world. Finally, there were some city instances in the failure case statements that matched with more than one standard entity statement and could not be resolved with state or region information using the technique described in the "Paris" example in Section II.

We used the following semi-automatic method to create the links. First, we created the links for all of the one-to-one matches that were identified unambiguously by the lexicon. This resulted in 111 links. We then manually checked the other 70 instances of the class city in the failure case statements to investigate why they did not match using the lexicon. We found that 45 instances are not actually real cities, 12 instances are cities but are not in DBpedia, 4 instances are cities that are in DBpedia but not in GeoLite, 2 instances had misspelled labels, and 7 instances had labels that matched with more than one city in DBpedia. Of the 9 instances that were misspelled or that did not have one-to-one matches, we were able to match 5 instances manually. We have integrated this linking method in the EKOSS semantic statement authoring tool so that the tool will suggest links for the user to create from instances of city to standard entity statements during the process of authoring a new failure case statement in the enriched repository.

V. Enhanced Semantic Search in Enriched EKOSS Failure Cases Repository

By linking the instances of the class city in the failure case statements to the standard entity statements, we obtain an enriched repository. We hypothesize that we can find more useful information from the enriched repository than from the original repository. However, supporting semantic search in the enriched repository required that we modify the EKOSS reasoner. In this section, we describe the modifications to the EKOSS reasoner and present a scenario to demonstrate the kind of complex semantic search results that are made possible by the enriched repository.

We described the semantic search procedure of the regular EKOSS reasoner in Section III. In order to support semantic searches in the enriched repository, we added a function so that when the reasoner loads one failure case statement into the knowledge base as the ABox, it also loads all of the standard entity statements that are linked to the failure case statement. However, because we include proximity information between cities, it is possible that one standard entity statement will link to another standard entity statement ad infinitum, causing the ABox to become large. This could cause the reasoner to stop working because description logics systems often cannot handle large amounts of instances [26]. Therefore, we added a parameter for the maximum length of a chain of linked standard entity statements, md, to limit the size of the ABox. If this parameter is set too low, it could limit the ability of the EKOSS reasoner to find results for some special search queries. Therefore, the md parameter provides us with a means to tune the size of ABox in order to speed up the semantic search as well as to avoid causing the reasoner to stop working due to an excessively large ABox.

The standard entity statements for cities contain many datatype properties, many of which we would like to use for query matching. Therefore, we added a function for numerical comparison to the EKOSS reasoner. The SCINTENG ontology contains a class **single property dimension with unit** that has some sub-classes for specific types of value, e.g. **area value**, **speed value**, and **length value**. The SCINTENG ontology also provides a datatype property "has value" with eight sub properties, such as "has max value", "has average value", "has exact value". Each instance of the class **single property dimension with unit** can have a "has value" property (or sub-property) pointing to a number as well as a "has specific type of unit" property that points to an instance describing the units of the value.

The EKOSS reasoner uses RacerPro [18] as the inference engine, which uses the new Racer Query Language (nRQL) that supports constraint checking. For example, RacerPro can answer queries like "find all cities with the area value greater than 1000 square kilometers". However, while the separation of the value and unit of SCINTENG ontology increases the expressiveness, it makes direct numerical comparison impossible. We created a function to normalize each value with a non-standard unit to a standard unit before sending the value to the RacerPro inference engine. For example, there are several sub-classes of **area unit** class, such as **square meter**, **square foot**, and **square kilometer**. We set **square meter** as the standard **area unit**. Each time the EKOSS reasoner finds an instance of **square foot** or **square kilometer** class in a semantic statement or a search query, it will convert that instance into the standard **square meter** class and update the corresponding value. This makes it possible for the RacerPro inference engine to compare the numerical values correctly.

Using these functions, the EKOSS reasoner can support a wide range of complex semantic searches in the enriched repository. In the following, we use a scenario about a researcher looking for information related to her research in order to demonstrate the kind of semantic searches that can be made.

Lucy is a researcher studying indirect impacts of disasters on large cities. She wants to search the failure cases knowledge base using the query "find all pairs of cities where one city, with a population density less than 3000 people per square kilometer, is the location of some disaster, and the other city, with a population density more than 8000 people per square kilometer, is in proximity to the first city". Figure 2 shows her search query as a graph. Boxes show query variables from the domain ontology, colored according to the

major upper class as in Figure 1. The text in each box above the line is the matching expression and below the line is the class name of that variable. Arrows show properties expressing the asserted relationships between variables.

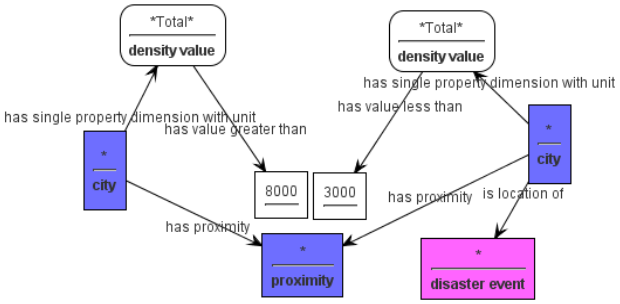

Figure 2. Lucy's search query shown as a graph.

The original EKOSS reasoner can evaluate matches of a search query with a semantic statement at five levels of complexity [3]. As described above, we have added functions to support searching in the linked standard entity statements and to support numerical comparison. We also added an auxiliary function to support text matching using simple matching expressions of the form "*XYZ*", where the symbol "*" indicates any text. For example, in Figure 2, the labels of two variables of the class density value are "*Total*", which indicates they can match with instances of density value that have labels containing the string "Total".

The EKOSS reasoner checks Lucy's query against the 291 failure case statements in the enriched repository and finds one matching failure case entitled "Leakage of toxic substances at chemical plant". The matching results are shown in Figure 3. Boxes show instances of classes from the domain ontology, colored according the major upper class as described in Figure 1. The text in a box shows the instance label or matching expression, followed by a colon, followed by the class name. Arrows with blue labels show properties between instances in the enriched failure case statement. Arrows with red labels show properties between class variables in the search query. Red boxes show the instances from the failure case statement that match with the class variables of the search query.

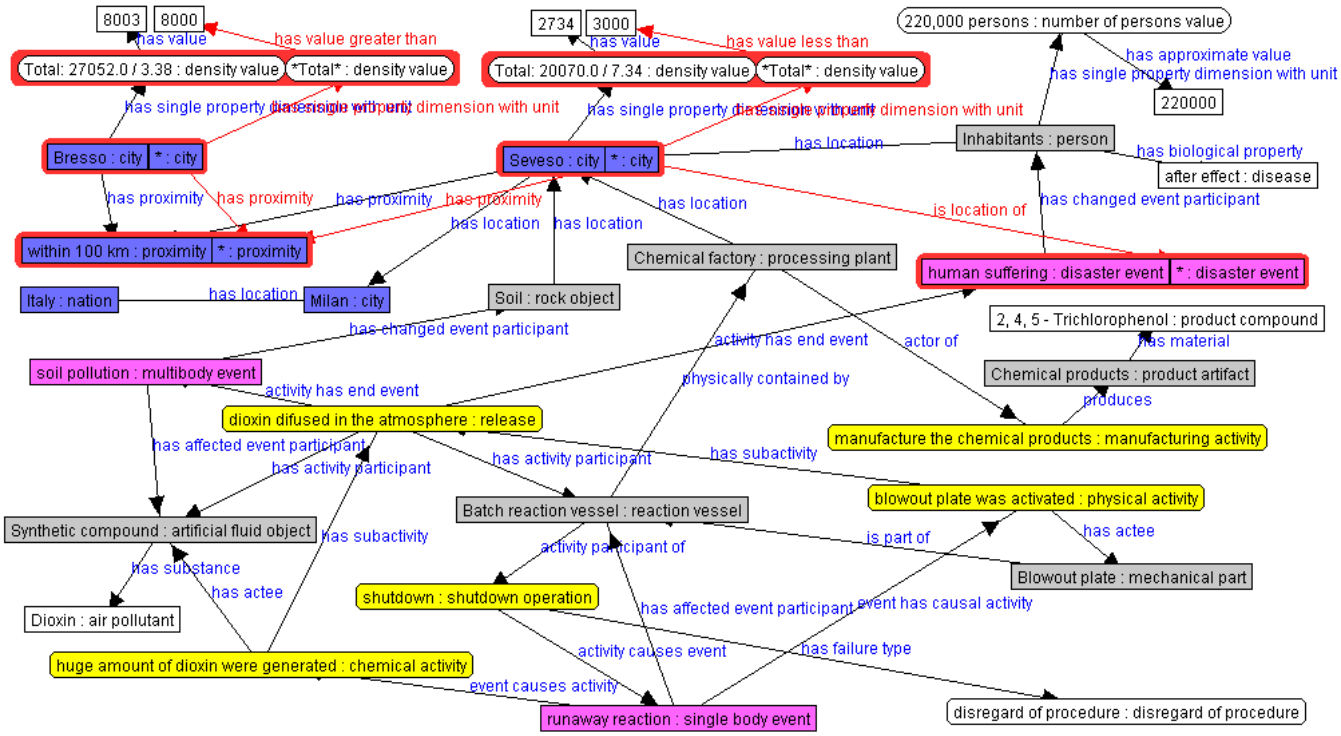

Figure 3. The matching results graph of the enriched failure case statement entitled "Leakage of toxic substances at chemical plant" and the search query shown in Figure 1.

The instances from the failure case statement that match with the query variables satisfy both the class membership and the property connections specified by the query. For example, the relationship between "human suffering : **disaster event**" and "Seveso : **city**" is inferred from the rule that "if *A* is location of *B*, and *C* has changed event participant *B*, then *A* is location of *C*", which matches with the specification of "**city** is location of some **disaster event**" in the query. The city of Seveso matches with the query specification of a city with population density less than 3000 people per square kilometer, and the city of Bresso matches with the query specification of a city with population density greater than 8000 people per square kilometer.

The original EKOSS failure cases repository and reasoner cannot find any matches with Lucy's query because the information about population density and city proximity is absent. Lucy could search in the repository with a simple query "find a city that is location of some disaster event" first, and then check DBpedia and GeoLite by hand to find which matching cities meet the population density and city proximity conditions. But it will take more time and effort. By using the enriched EKOSS failure case repository and the

enhanced EKOSS reasoner, Lucy can find the matching failure case easily.

VI. Related Work

Choudhury et al. [27] integrated the YouTube tag space with the LOD cloud. After getting a cleaned and ranked tag space, they created a mapping mechanism from user tags to ontological resources by using WordNet, a similarity module, and the semantic indexing engine Sindice [28]. Therefore, they created links between user tags and LOD. Our work is different in that the EKOSS failure cases repository is ontological resource, not a free text tag space.

Passant and Laublet [29] developed the MOAT model to capture the semantics of tagging systems using a lightweight ontology. MOAT provides a collaborative way for Web 2.0 content producers to give meanings to their tags in a machine-readable format. Their approach used Linked Data principles, such as defining the meanings by using URIs from existing data resources. Therefore, by using MOAT, users can create semantic tags for their contents, some of which contain existing URIs from LOD. However, because their work is based on a lightweight ontology, it does not support complex reasoning.

Kobilarov et al. [30] used DBpedia as a controlled vocabulary to connect identical entities in different domains of the British Broadcasting Corporation (BBC). They developed a system to automatically interlink existing BBC concepts with DBpedia. They also developed a named entity extraction system called Muddy Boots to extract entities in BBC News articles, and then they used the DBpedia identifier for those entities to link the News articles with documents in other BBC domains. Consequently, they used LOD to interlink the BBC domains but not to import common knowledge into the BBC domains for logical reasoning.

Konyk et al. [31] created an OWL ontology to represent chemical knowledge. They also created a new drug repository extracted from PubChem, DrugBank and DBpedia. Their system supports simple DL queries. They mapped the Wikipedia link found in some of the DrugBank records to the corresponding DBpedia entities. They then imported the DBpedia entities into their knowledge base as individuals of the corresponding drug class from the DrugBank ontology, together with the RDF graphs that provide some common knowledge about those entities. However, they did not develop any means for integrating DrugBank records that did not contain Wikipedia links with DBpedia entities. And their system does not appear to support complex DL queries such as the one we have described in the previous section.

We imported the common knowledge about cities extracted from DBpedia knowledge base and GeoLite City database into the EKOSS failure cases repository by creating a set of standard entity statements. This resulted in an enriched repository that made it possible to do more complex logical reasoning.

VII. Conclusion

Detailed domain-specific "heavy-weight" semantic statements that are manually authored often lack common knowledge about the concepts and entities that are described. Enriching such semantic statements with more common knowledge can enhance their ability to answer complex search queries. We presented a simple procedure to enrich the EKOSS failure cases repository with common knowledge about city entities extracted from DBpedia knowledge base and GeoLite City database. We first extracted information about city entities from the two data sources, and then used that information to generate standard entity statements for each city entity. We created links between the instances of class **city** in failure case statements and the standard entity statements using a lexicon built from the YAGO ontology, and we added several functions to the EKOSS reasoner to support a wide range of searches in the enriched repository. Finally, we described a search scenario demonstrating that we can get informative search results from the enriched repository for some complex search queries that do not get any results from the original repository.

In addition to the enrichment procedure, the original contributions of this paper are: 1) extracting and augmenting the DBpedia information on cities, including development of a mapping function to GeoLite that uses a lexicon created from YAGO together with region data for city disambiguation; 2) a set of templates for translating DBpedia and GeoLite information into the SCINTENG ontology framework; and 3) enhancements to the EKOSS reasoner to enable semantic searches in the enriched repository.

In future work, we will investigate methods for enriching the failure cases repository with LOD "on the fly" by importing the information about city entities and other standard entities from LOD dynamically. To do this, we will implement web services that provide linking APIs for semantic data, map instances in the failure case repository to other types of entities in the DBpedia knowledge base, and represent the common knowledge of those types of DBpedia entities using the SCINTENG ontology.

Acknowledgment

Funding for this research was provided by the Knowledge Failure Database project at the Japan Science and Technology Agency and the Office of the President of the University of Tokyo. This article uses DBpedia knowledge base created in DBpedia project, available from http://dbpedia.org/. This article uses GeoLite data created by MaxMind, available from http://www.maxmind.com/.

References

[1] W. Guo, S.B. Kraines. "Enriching City Entities in the EKOSS Failure Cases Knowledge Base with Linked Open Data", In *Proceedings of the 6th International Conference on Next Generation Web Services Practices (NWeSP2010)*, November 23-35, Gwalior, India, pp: 58-63.

[2] S. Kraines, W. Guo, B. Kemper, Y. Nakamura. "EKOSS: a knowledge-user centered approach to knowledge sharing, discovery, and integration on the Semantic Web", In *ISWC 2006*, pp. 833–846.

[3] W. Guo, S.B. Kraines. "Mining relationship associations from knowledge about failures using ontology and inference", In *LNAI, vol 6171, ICDM 2010*, pp. 617-631.
[4] W. Guo, S. Kraines. "Generating Semantic Statements related to Energy and Sustainability Semi-automatically", *The Alliance for Global Sustainability (AGS) Annual Meeting 2010*, 16-19 March 2010, Tokyo, Japan, poster.
[5] W. Guo, S. Kraines. "Explicit Scientific Knowledge Comparison Based on Semantic Description Matching, American Society for Information Science and Technology Annual Meeting", In *ASIS&T 2008*, Columbus, Ohio, 2008, DOI: 10.1002/meet.2008.1450450210.
[6] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, S. Hellmann. "DBpedia - a crystallization point for the Web of Data", *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, Issue 7, Pages 154–165, 2009.
[7] DBpedia 3.4, http://wiki.dbpedia.org/Downloads34?v=hmc
[8] Freebase - A wealth of free data, http://www.freebase.com
[9] F.M. Suchanek, G. Kasneci, G. Weikum. "Yago: a large ontology from Wikipedia and WordNet", *Journal of Web Semantics*, vol. 6, no. 3, 2008, pp. 203–217.
[10] Doapstore, http://doapstore.org/
[11] GeoNames, geographical database, http://www.geonames.org/
[12] FOAF dataset, http://ebiquity.umbc.edu/resource/html/id/82/
[13] DBLP Bibliography, http://www4.wiwiss.fu-berlin.de/dblp/
[14] T. Berners-Lee, "Linked Data - design issues," date: 2006-07-27, http://www.w3.org/DesignIssues/LinkedData.html
[15] Wikipedia, the Free Encyclopedia, http://en.wikipedia.org/wiki/Main_Page.
[16] GeoLite City Database, http://www.maxmind.com/app/geolitecity
[17] OWL, http://www.w3.org/TR/owl-guide/
[18] RacerPro, http://www.racer.systems.com
[19] JST Failure Knowledge Database, http://shippai.jst.go.jp/en/Search
[20] S.B. Kraines, W. Guo. "Supporting Reuse of Knowledge of Failures through Ontology-based Semantic Search", In *KMIS 2010*, Valencia, Spain.
[21] ISO 3166-2, http://www.maxmind.com/app/iso3166_2
[22] FIPS 10-4 Subcountry codes, http://www.maxmind.com/app/fips10_4
[23] R.W. Sinnott. "Virtues of the haversine", *Sky and Telescope*, vol. 68, no. 2, 1984, pp. 159.
[24] W. Guo, S.B. Kraines. "SEMCL: A Cross-Language Semantic Model for Knowledge Sharing", *International Journal of Knowledge and Systems Science*, vol. 1, no. 3, 2010, pp. 1-19.
[25] W.W. Cohen, P. Ravikumar, S.E. Fienberg. "A comparison of string distance metrics for name-matching tasks", In *Proc. of the ACM Workshop on Data Cleaning, Record Linkage and Object Identification 2003*.
[26] A. Ankolekar, M. Krotzsch, T. Tran, D. Vrandecic. "The two cultures: Mashing up Web 2.0 and the Semantic Web", *Journal of Web Semantics*, vol. 6, 2008, pp. 70-75.
[27] S. Choudhury, J.G. Breslin, A. Passant. "Enrichment and ranking of the YouTube tag space and integration with the Linked Data Cloud", In *LNCS, vol 5823, ISWC 2009*, pp. 747-762.
[28] Semantic Indexing Engine - Sindice, http://sindice.com/
[29] A. Passant, P. Laublet. "Meaning of a tag: a collaborative approach to bridge the gap between tagging and Linked Data", In *LDOW 2008 at WWW 2008*, Beijing, China.
[30] G. Kobilarov et al.. "Media meets Semantic Web - how the BBC uses DBpedia and Linked Data to make connections", In *LNCS, vol. 5554, ESWC 2009*, pp. 723-737.
[31] M. Konyk, A. De Leon, M. Dumontier. "Chemical knowledge for the Semantic Web," In *LNBI, vol. 5109, DILS 2008*, pp. 169-176.

Author Biographies

Weisen Guo is a project researcher in the Science Integration Program (Human) in the Department of Frontier Sciences and Science Integration in the Division of Project Coordination at the University of Tokyo. He holds a PhD in Management Science and Technology from the Dalian University of Technology, China. He has served as a lecturer of the Institute of Systems Engineering in the Management School at the Dalian University of Technology, and as an adjunct project manager of the Projects and Results Management Office in the Planning Bureau at the National Natural Science Foundation of China. His research interests include Semantic Web technology, graph mining, knowledge science and technology, information system engineering.

Steven Kraines is an associate professor in the Division of Project Coordination at the University of Tokyo. He obtained his PhD in Chemical Engineering from the University of Tokyo. His research interests include applications of knowledge representation languages, ontologies, and inference-based semantic matching to problems in urban sustainability and health science.