International Journal of Information System Modeling and Design
Volume 13 • Issue 7

Altering OWL Ontologies for Efficient Knowledge Organization on the Semantic Web

Abhisek Sharma, National Institute of Technology Kurukshetra, India*
Sarika Jain, National Institute of Technology Kurukshetra, India
https://orcid.org/0000-0002-7432-8506

ABSTRACT

The increase in the number of users on the internet and the advancement in information technology have spiked the generation of information to an unprecedented level making information retrieval and web mining a difficult task. Semantic technologies can help improve the results of web mining by providing constructs that can help represent the web documents in a machine-understandable manner. To keep providing semantically rich services while keeping this surge in the amount of information in mind, we have to work towards ways to make the process of information management efficient while retaining its effectiveness. One of the ways to accomplish the above task is to improvise the knowledge organization in a manner that every piece of information is in its designated place. This paper discusses and addresses the problems with current knowledge organization methodologies and presents an algorithm to alter the available OWL ontologies. The authors were able to get a noticeable improvement in the amount of storage used by the ontology with fewer axioms without losing any information.

KEYWORDS

Knowledge Base, *Knowledge Organization*, *Ontology*, *OWL*, *Semantic Web*

MOTIVATION AND INTRODUCTION

We have a large variety of concepts in this world (both on the web and in enterprise applications) with huge volume of it, which poses problems when it comes to processing of that data because we need to keep the processing time as low as possible to provide almost any service efficiently. Moreover, most of this data is unstructured in nature. With this exponential growth of data that is being generated on a daily basis, the existing data processing tools faces issues in accommodating and processing this huge data hampering the capabilities of the tools in providing services effectively and efficiently. Though being very rich in information, the current web mining technologies are unable to fetch meaningful information that makes sense in right context. This is because the web documents are largely unstructured and unorganized. The challenge remains to efficiently and effectively

DOI: 10.4018/IJISMD.313431 *Corresponding Author

This article published as an Open Access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0/) which permits unrestricted use, distribution, and production in any medium, provided the author of the original work and original publication source are properly credited.

determining and digging up the machine understandable information. In order to smooth web mining (automatically discovering knowledge from the web documents), we require automated schema and knowledge representation formalisms that can formalize the description of domain knowledge at a conceptual level (Patel et. al., 2018). We need such a representation that focuses on efficient knowledge management while retaining the benefits offered by current semantic technologies.

Traditionally knowledge in represented using Web Ontology Language (OWL) which is a family of languages for knowledge representation, for the construction of ontologies. Ontologies are a formal representation of taxonomies and their properties. Ontologies allow machines to understand and process the information in a semantically rich manner which allows the machines to answer the queries while focusing on the semantics of the information and query because simply keyword search is not enough considering the ambiguity in the meaning of the words (Jain et. al., 2008) (Shandilya, 2009). The purpose of insisting on the semantics of the information is to make the systems able to provide appropriate information related to the specific context. The knowledge provided by ontology is extremely useful in defining the structure and scope for mining Web Content.

Previously there have been many proposals on how representation can be done, some of them are semantic network, production rule, logic and frame. All of these representation methods provide many functionalities from being semantically rich to capabilities to infer new information from the existing one, but still there is scope of improvement as all of these representation don't focus on redundant nature of the data that is present all around us, this causes many problems like inefficiencies in the ontology mapping process as we will be iterating over many redundant information, it is also making the ontology heavy which will be problematic for knowledge management systems when providing the services, etc.

Here in this paper we propose a knowledge representation technique that keeps check on different types of properties, helping in retaining all the necessary information without redundancy. When a representation format keeps redundancy into check then we can attain many other benefits, such as the size of the ontologies will be reduced which will further result in better management of the knowledge. We also discuss about ways through which any knowledge base/ontology can be brought to this representation format.

THE PROBLEM

Knowledge Organization Systems (KOS) (Zeng, 2008) are being seen as semantic tools used in plurality of contexts by diverse communities. The typology and spectra of KOS have been studied and reviewed in literature (Zeng et. al., 2019) with the most common briefed here:

- Glossaries are the domain specific list of concepts and their definitions, for example Dublin Core.
- Folksonomies are the collaborative tagging systems with informal semantic relations, for example Del.icio.us and Flickr.
- Controlled Vocabularies are glossaries ensuring consistency and ambiguity.
- Taxonomies have strict hierarchical categorization, for example the computer file system and a catalogue of product categories.
- Thesauri are restricted to lexical relation, for example WordNet (Miller et. al., 1990).
- Conceptual Data Models also exist for designing information systems and DBMSs, for example EER and UML.
- Ontologies consist of classes, relationships, data, data properties, axioms, and object properties.

All the KOS listed above differ in their semantic strength, degree of axiomatization, and structural richness. The term ontology has been borrowed from philosophy and is a specific form of KOS. In Artificial Intelligence, ontology is used as a generic term to designate any type of KOS. In addition

to functioning as conceptual vocabularies, formal ontologies also provide properties and instances. Ontologies make the data machine-understandable and interpretable, hence enabling the interaction between users and systems. They facilitate sharing of information among users and systems by handling issues like heterogeneity and data integration in a number of domains.

A large palette of languages and data models exist for representing, sharing and linking these KOS, viz. the Simple Knowledge Organization System (SKOS) (Miles et. al., 2005), the Rule Interchange Format (RIF) (Kifer et. al., 2008), Resource Description Framework (Schema), and all variants of the Web Ontology Language (OWL) (Bechhofer et. al., 2004). All differ in the expressiveness they offer and also the complexity. OWL is seen as an emerging ontological approach and semantic data model. OWL is a W3C standard and a vocabulary extension of RDF, designed to represent complex things and relationships. OWL can be read and interpreted by computers.

Even after the possibility of wide variety of applications through the use of ontologies, there are certain problems with Web Ontology Language (OWL) itself which are required to be addressed. Though addressing the weaknesses of current efforts in metadata, today's ontologies still depend on intervention by humans for them to be useful.

Dealing With Incomplete Information

When the information present in the ontology is incomplete due to any reason, the facilities provided by sematic technologies through the use of ontology (such as inferring new information out of existing one) will be inconsistent, can be incorrect, inconsistent, and ineffective as the inferred information can be misleading.

Dealing With Imprecise Information

If ontology contains imprecise information, then the ontology itself loses its credibility as the information present in the ontology can't be trusted. As more fake news or misinformation is floating on the internet we need to make sure the information that we propagate into the ontology is from credible source and is not fake. There are many fellow researchers who have been working on the problem of fake information using machine learning techniques as humans can't handle the huge size of information available on the internet. Even if there are companies like Facebook (ant its subsidiary WhatsApp) having centres for content filtering but they also face problems catching up with the amount of data generated daily.

Dealing With Large Scale Information

Large scale ontologies can pose difficulties at various places when managing ontologies, such as accessing information from ontologies, storage and loading of ontologies, and making the whole ontology accessible and queryable. All these tasks become inefficient as the size of the ontology increases and if not handled properly.

Representation of Multilingual Information

The world we live in is a world with many languages across countries; and even within countries like India and South Africa. To make the services provided by the semantic technologies available in multiple languages we have to work with different methods of representation that can accommodate multilingual information. The problem of information unavailability poses challenges especially for languages like Xhosa whose descriptions are hard to find and even harder to map to other languages like English.

Problem of Redundancy

The problem of redundancy spans over multiple problems in ontologies or semantic technologies as a whole, for example redundancy can accumulate towards making the ontology large scale (as

discussed in 2.3.) which would then require special measures to deal with. The problem of large-scale ontologies can be catered to if we can somehow eliminate redundancy.

For dealing with such problems, different solutions have been proposed in literature including the combination of folksonomy and ontology approaches. The formality and semantic richness of ontologies when combined with the ease and adaptability of folksonomies will provide long-lasting and far better results.

PROPOSED APPROACH

We base our theory on the possession of two types of features by every concept. The set of distinguishable features (DFs) of a concept consist of properties that distinguish all individuals of the given concept from objects of the other concepts. All distinguishable features of a concept must be true for every individual of that concept. The set of cancellable features (CFs) of a concept consist of properties that may or may not hold for an individual of that concept. The distinguishable features can also be termed as concept-specific and the cancellable features as individual-specific. The theory of distinguishable and cancellable features roots back in literature (see (Michalski, 1983) and (Patel et. al., 2018)). Patel et. al., 2018, has conceptualized this as "Unit of knowledge", where ontological concepts are stored as an atomic unit in the ontology. Storing concepts as atomic unit will make redundancy more manageable.

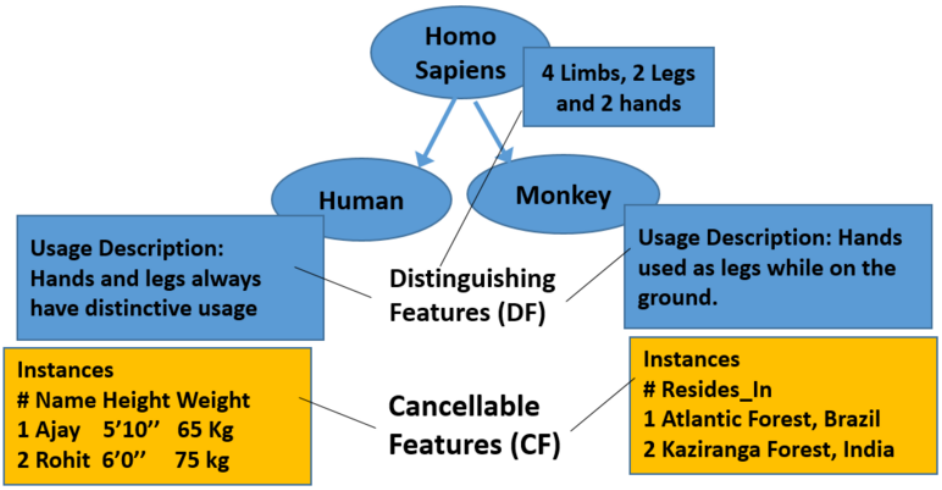
There are two major points of consideration. Firstly, as all instances of a concept share same value for all the DFs of that concept, it is highly redundant to store these values with each instance. By keeping DFs at class/concept level, we can make the storage efficient. Secondly, as we move down the hierarchy, new DFs may be added, or old values overridden, but never any DF is dropped. By keeping DFs at the highest possible level, we can make the storage efficient. The same protocol follows for the cancellable features also. The cancellable features are also mentioned and stored at the highest possible level in the hierarchy. While moving down the hierarchy vertically toward the specific concepts or horizontally toward the individuals, new features may be added, the values of old features may be overridden, or some old feature completely dropped. The value of cancellable features is put with a specific class or with an individual only if it overrides the inherited value from its general concept or its instantiating concept respectively.

Consider for example the case of concepts Human and Monkey that are different in many ways but also same in some ways. They both are Homo sapiens, have same number of limbs, even some of the portions in their brains are with similar functionalities; but the usage of limbs differs (as shown in fig 1). All these features can be categorized as DF's of the HomoSapiens concept/class as they are getting inherited by all the subclasses (Human and Monkey). For both Human and Monkey, the limbs are divided into 2 categories, legs and hands; but usage and description of both is different. Humans use their legs to walk and hands to let's say eat or to pick up things. But for Monkeys usage differs on the task at hand, such as for eating they use their fore limbs as hands but at the time of walking they use all of their limbs as legs. So, the usage description is a property that distinguishes Human and Monkey concept/class. So, usage description with different values become DF for Human and Monkey, and rest all common features go up to the HomoSapiens concept. The CF's of these concepts can be the instance specific details such as name, height, weight, etc.

Although, there will be exceptions, but they will not pose much problem. As the ontologies are being queried upwards (instance → class → superclass → so on...). This implies that if the value is present at instance, it will not be inherited from the class, so the exception instances (like Birds who has a broken wing) will have information associated with the instance itself.

The said theory could be incorporated in the ontological structure in two ways, either by altering the ontology development methodology and creating all the ontologies from scratch taking care of DFs and CFs; or by augmenting the existing ontologies in a manner to incorporate the theory. Both the ways have been briefed here in sufficient detail.

Figure 1. Depiction of Distinguishable and Cancellable Features



Altering the Ontology Development Methodology

Among the methodologies to design ontologies, some design them from scratch while others reuse the existing ones. Some noteworthy methodologies include the Cyc method, Uschold and King's skeletal method, Grüninger and Fox's methodology (TOVE), Methontology, MYCIN, KACTUS, DILIGENT, UPON, SENSUS and On-To-Knowledge (OTK) (Kurilovas et. al., 2015) (Malhotra et. al., 2015). To achieve the said benefits, we need to alter the ontology development methodology. The refined methodology is as described below:

1. **Determining Scope:** We determine scope of the ontology in this phase by providing answer of the question 'why are we going to develop it?' There is no change in this phase.
2. **Identifying Concepts:** This phase involves identifying the fundamental concepts and relationships, their attributes, instances, any constraints, and restrictions. In addition to this, the attributes of every concept are divided into the distinguishable and the cancellable features.
3. **Concept Analysis and Organization:** This step involves analysing the terms and building the conceptual model by binding the terms into a hierarchy. There is no change in this phase also.
4. **Encoding:** This step involves transcribing the structure in OWL. Care is to be taken to store the DFs and the CFs as stated.

Augmentation Approach

As semantic technologies are matured by now and many core datasets are already available to use; we provide a procedure to augment the existing ontological datasets. The said procedure is semi-automatic in the sense that final authority lies with the human counterpart.

In the augmentation process the instances of existing ontologies will be traversed based on their classes and if a property has same value for each and every instance of a particular class then that property will be labelled as DF of that particular class, this above process will be done on every class and all the DF's corresponding to a class will be kept inside a list. After all the DF's have been identified then the process of shifting DF's up in the ontology's hierarchy will began, in which all the DF's identified per class will be shifted and stored with the corresponding class. This process of identification of DF's and shifting will be carried till there is no new DF identified.

AUGMENTING THE EXISTING DATASETS

The procedure of augmentation works with the properties only, so it is applicable to all hierarchical knowledge organization systems. We concentrate our idea on the OWL ontologies. The process of ontology augmentation takes ontology as an input and return augmented ontology in accordance with the theory of distinguishable features. The augment algorithm passes through three phases as described below with the detailed description of each component in subsequent subsections:

1. Fetching the Leaves
2. Marking Distinguishable Features
3. Moving Distinguishable Features

The input to the algorithm is an OWL ontology and the output is augmented ontology with the distinguishable features stored in a non-redundant manner. Here is the consolidated algorithm.

```
augment(ontology){
leavesList ¬ fetchLeaves()
classDFPairList ¬ fetchClassDFPair(leavesList)
updateOntology(classDFPairList)
}
```

Fetching Leaves

The process starts with fetching the classes that are immediate children of Thing, this gives us the major generalized concepts in the ontology, after we will be traversing through the children of classes in hand till we find the classes with no child, which are the leaf concepts. The leaves as identified are stored in a list. Fig 2 provides the algorithmic overview of this component.

Marking Distinguishable Features

Having fetched all the leaf concepts of the ontology, it is time to mark the distinguishable features of concepts. For this, we fetch the individuals of these leaf concepts, one leaf concept is selected at a time and the properties of its individuals are checked, if some property is present

Figure 2. Fetching Leaves

```
fetchLeaves(){
leavesList ← NULL
internalNodes ← Concepts having predicate as rdfs.subClassOf and object as Thing.
   // Fetch the immediate children of Thing (the root node)
while(internalNodes is not empty)){
   nextInternalNode ← next(internalNodes)
        //get the next internal node and delete it from the list
   children ← fetch immediate children of nextInternalNode
   if no child fetched{
        add nextInternalNode to leavesList
   }
   else{
        add all children to the list of internal Nodes.
   }
return leavesList
}
```

in all the individuals of a concept, that property is marked as DF of that concept. If the values of this property are same for every individual of that concept, then that value becomes the default value for this property of the concept, otherwise the value with maximum occurrence is chosen as the default.

Only the individuals that override the default value of the property will save the value for the property. Figure 3 provides the algorithmic overview of this component.

Moving Distinguishable Features Toward the Root

Having marked the distinguishable features of every leaf concept, it is time to move the features up in the hierarchy. For this, we group the leaf classes into groups of immediate siblings. For every group, we look for the common DFs in all siblings. These common DFs are moved to their parent. If the values of some common DF are same for all the siblings, then that value becomes the default value for this DF of the parent, otherwise the value with maximum occurrence is chosen as the default. Only the concepts that override the default value of the DF will save the value for the DF. The process continues till we reach the root concept. Fig 4 provides the algorithmic overview of this component.

In this step we use the output of the previous i.e. the class and DF property pair list and add the DF properties and their corresponding values alongside the class while iterating over the list. Rest of the properties which are left are CF's and we are not changing anything in them as they will automatically be left after we extract DF's out. In the above code snippet '?' resembles a variable which can be anything (in our case it is individual of the ontology which contains the particular property-value pair).

Figure 3. Marking Distinguishable Features

```
fetchClassDFPair(leavesList){
classDFPairList ← NULL
while(leavesList in not empty){
  nextLeafNode ← next(leavesList)
       //get the next leaf node and delete it from the list
  DFPropertiesForLeafNode ← NULL
  individualsList ← instances having rdf:type as nextLeafNode
  if (individualsList is not empty){
     individualsCount ← count(individualsList)
     //counts number of element in individualsList
     while(individualsList is not empty){
     individual  ← next(individualsList)
        propertiesList ← properties of individual
        while(propertiesList is not empty){
           property ← next(propertiesList)
             //get the next property and delete it from the list
           countOfPropertyOccurrences ← # of instances with same property-value
           if (countOfPropertyOccurrences is equals individualsCount or greater than
           individualsCount/2){
           add tuple(property and value) to DFPropertiesForLeafNode
           }
        }
     }
     add nextLeafNode along with DFPropertiesForLeafNode to classDFPairList
  }
}
return classDFPairList
}
```

Figure 4. Moving Distinguishable Features Toward the Root

```
updateOntology(classDFPairList){
while (classDFPairList is not empty){
        nextClassDFPair ← next(classDFPairList)
        class ← extractClass(nextClassDFPair)
          //extracts Class from nextClassDFPair
        DFList ← extractDFList(nextClassDFPair)
          //extracts DFList from nextClassDFPair
        while(DFList not empty){
                nextDF next(DFList) //get the next DF and delete it from the list
                add nextDF to class //add nextDF (property and value) along with class
                remove nextDF from all instances
          //removes nextDF from instances that uses it
        }
   }
}
```

RESULTS AND DISCUSSION

For the purpose of this evaluation, we are using a set of ontologies[1] that we have created. Before we go further let's talk about the reason behind creating ontologies and not using previously available ones. The existing ontologies doesn't contain proper instantiation, at least publicly available ones that we came across. Why we need instantiation? We need instantiation because we are focusing on the properties and values associated with them. And values are only assigned to a property when we have an instance that uses that property. So, in our case we need ontologies which are properly instantiated to check whether a property and value associated with it is either DF or CF.

Now let's discuss about the ontologies and their details. First ontology is named Eats which contains classes like omnivorous, carnivorous and herbivorous. Second ontology is named Animal which have classes named Warm Blooded, Cold blooded, Bird, Reptile, etc. It contains many details about animal (Specifically Birds) like number of legs, number of wings, type of blood (warm or cold), etc. We have also created many instances of these classes. In the case of conventional approach, we have kept the values of all the properties with instances itself. But in the case of augmented approach, the DF properties has been moved to the upper layer (Class) through the use of above defined algorithm. There are also cases where the properties are pushed even further up in the classes.

The "Eats" Ontology

Consider the "Eats" ontology. Figures 5 and 6 depicts the classes and instances respectively when they are created in conventional way. Figures 7 and 8 depicts the classes and instances respectively when they are created in augmented way.

Figure 5. Classes of "Eats" Ontology in conventional way

Figure 6. Instances of "Eats" Ontology in Conventional way

Figure 7. Classes of "Eats" Ontology after Augmentation

Figure 8. Instances of "Eats" Ontology after Augmentation

The "Animal" Ontology

Consider the "Animal" ontology. Figures 9 and 10 depicts the classes and instances respectively when they are created in conventional way. Figures 11 and 12 depicts the classes and instances respectively when they are created in augmented way.

Figure 9. Classes of "Animal" Ontology in conventional way

Figure 10. Instances of "Animal" Ontology in Conventional way

Figure 11. Classes of "Animal" Ontology after Augmentation

Figure 12. Instances of "Animal" Ontology after Augmentation

We present here three aspects of how the augmentation approach supersedes the conventional approach to ontology building. The augmentation approach to enhance ontologies reduces the existing redundancy, reduces the total number of axioms in the ontology, and also benefits in ontology mapping. Above all, these benefits are attainable without any side effects. In section 5.2, we see that even after augmentation we are able to get the similar results for the posed queries.

Evaluation

Table 1 shows different aspects of the developed ontologies using both the approaches. The difference in storage space and number of axioms is apparent, while preserving the available information.

When knowledge is organized in ontologies according to the theory of DFs and CFs, we perceive obvious benefits in all the use cases that can be thought over an ontology:

1. **No redundancy:** The usage of DF's and CF's reduces redundancy as all the redundant properties that are being used by most of the instances of a class will be moved to the class level and will be stored only once, these properties are still accessible through each of the instances as they are the properties which will be inherited by the instances from the classes in the upper level in the ontological hierarchy.
2. **Scalability:** As a result, the ontology will have same number of classes and properties (and all other information) but becomes considerably lighter as the number of axioms are reduced drastically. This reduces the overall size of the ontology. As can be seen in table 3, we notice a 20% reduction in the size of the ontology on disk and 30% reduction in the number of axioms required to represent the same knowledge. When it comes to big ontologies like DBPedia and OpenCYC with millions of entities, the proposed solution proves to be more scalable than any other.
3. **Efficient Ontology Mapping:** In the typical ontology mapping process, we first find syntactic similarity between terms then we try to disambiguate the mappings by finding the similarities based on meaning or semantics of the terms, for this we look for similarities in hierarchical structure and their properties/features. Syntactic similarity is straightforward and differs just on the selection of an efficient algorithm. Similarities based on properties/features where computational complexity is directly correlated with the number of properties per individual corresponding to a class is more complex. And as the number of properties increase, computational complexity increases as well. The concept of DFs state that not all of the properties are significant when calculating similarity, the properties that are being used by most or all of the instances (DFs) are the one which contributes to the similarity score, the rest of the properties are not contributing to the similarity score (CFs) and are an unnecessary computational overhead while calculating similarity. We should therefore utilize only the DF's (the contributing properties for comparison) for comparison, hence increasing the efficiency of the complete mapping process.

No Side Effects

One obvious consideration and evaluation of the proposed approached is looking into the side effects that may happen because of the reduction in the number of axioms. As look obvious, the reduction

Table 1. Comparative statistics of ontologies developed using conventional and augmentation techniques.

Name of the Ontology	Ontology Development method	File Size in KB	No. of Axioms	No of Classes
Eats	Conventional	22.2	140	5
	Augmented	18.5	98	5
Animal	Conventional	38.0	258	14
	Augmented	30.2	173	14

in the number of axioms could result in information loss and also making the querying inefficient. To evaluate upon this aspect, queries are devised to show that we are able to fetch same information without any loss from the ontologies developed using either of the approach. Table 2 contains three queries for the Eats ontology and Table 3 contains the same three queries for the Animal ontology. In both the tables, the first column mentions the ontology of the conventional approach, the second column mentions the ontology of the augmentation approach, and the third column mentions the output received from both the ontologies. We notice that SPARQL queries for the augmented ontologies could be written with the same ease and also that the query output obtained in both the cases is the same. Although, we have to change the query a bit but we were able to get similar results from both versions of the ontologies. There is no information loss.

It has been therefore been verified that:

1. There is no major change in writing the SPARQL queries or other processing involved.
2. There is no information loss as could be seen in section 5.2.

CONCLUSION AND FUTURE PLANS

Existing Knowledge Organization Systems provides various services like ability to keep information in a semantically rich way, perform inference on the existing knowledge to depict new information, ability to fetch information from point in the ontology as each node acts as an entry point, etc. Still there is scope of improvement in order to handle the scale at which the data is increasing. The methodology and the algorithms proposed in this work will aid the researchers and academicians in augmenting their ontological datasets by giving due weightage to the distinguishing features. Once such datasets are prepared, they will serve as an effective and efficient source of information for the knowledge management applications and the semantic web. Our experimentations have found noticeable difference between conventional ways and our approach, even though our ontologies were

Table 2. Queries for the Eats Ontology

1. Fetching all the values corresponding to a particular instance/individual		
PREFIX eats: <http://www.semanticweb.org/eats#> SELECT distinct ?p ?object WHERE {eats:Cow ?p ?object .} *//Cow can be replaced with any Instance*	PREFIX eats: <http://www.semanticweb.org/eats#> SELECT distinct ?pred ?values WHERE{ { eats:Cow ?pred ?values} UNION {eats:Cow rdf:type ?class . ?class ?pred ?values}}	rdf:type - Herbivore rdf:type - owl:NamedIndividual eats - “Plant based material”
2. List of all the Herbivore (or instances of any class)		
PREFIX eats: <http://www.semanticweb.org/eats#> SELECT distinct ?subject WHERE { ?subject rdf:type eats:Herbivore .}	PREFIX eats: <http://www.semanticweb.org/eats#> SELECT distinct ?subject WHERE { ?subject rdf:type eats:Herbivore .}	Horse Deer Okapi Cow Elephant Capybara …
3. List all the properties associated with a class		
PREFIX eats: <http://www.semanticweb.org/eats#> SELECT distinct ?property WHERE { ?subject rdf:type eats:Omnivore . ?subject ?property ?o }	PREFIX eats: <http://www.semanticweb.org/eats#> SELECT distinct ?property WHERE {eats:Omnivore ?property ?values}	rdf:type eats

Table 3. Queries for the Animal Ontology

1. Fetching all the values corresponding to a particular instance/individual		
PREFIX animal: <http://www.semanticweb.org/animal#> SELECT distinct ?p ?object WHERE{ { animal:Albatrosses ?p ?object .} UNION {animal:Albatrosses rdf:type ?class. ?class ?p ?object} UNION {animal:Albatrosses rdf:type ?class. ?class rdfs:subClassOf ?superclass. ?superclass ?p ?object } }	prefix animal: <http://www.semanticweb.org/animal#> SELECT distinct ?p ?object WHERE{ { animal:Albatrosses ?p ?object .} UNION {animal:Albatrosses rdf:type ?class. ?class ?p ?object} UNION {animal:Albatrosses rdf:type ?class. ?class rdfs:subClassOf ?superclass. ?superclass ?p ?object } }	Color - “Golden White” rdf:type - owl:NamedIndividual rdf:type - Bird rdf:type - owl:Class rdfs:subClassOf - Warm_Blooded numberOfLegs - 2 numberOfWings - 2 rdfs:subClassOf - Vertibrate bloodType - “Warm”
2. List of all the Herbivore (or instances of any class)		
PREFIX animal: <http://www.semanticweb.org/animal#> SELECT distinct ?subject WHERE { ?subject rdf:type animal:Bird .}	prefix animal: <http://www.semanticweb.org/animal#> SELECT distinct ?subject WHERE { ?subject rdf:type animal:Bird .}	Finches Tit Wrens Kingfisher Crane Owl Thrush …
3. List all the properties associated with a class		

PREFIX animal: <http://www.semanticweb.org/animal#>
SELECT Distinct ?p
WHERE {
{animal:Bird ?p ?object.}
union
{animal:Bird rdfs:subClassOf ?o. ?o ?p ?object}
union
{?s rdf:type animal:Bird. ?s ?p ?object}
}
	prefix animal: <http://www.semanticweb.org/animal#>
	SELECT Distinct ?p
	WHERE {
	{animal:Bird ?p ?object.}
	union
	{animal:Bird rdfs:subClassOf ?o. ?o ?p ?object}
	union
	{?s rdf:type animal:Bird. ?s ?p ?object}
	}
		rdf:type
		rdfs:subClassOf
		numberOfLegs
		numberOfWings
		bloodType
		color

Note: We have kept the queries a bit descriptive in our examples for better understanding, they can be written in more concise manner following property paths (sequential or iterative).

really elementary and were small in size. We saw 20% improvement in size of the ontology on disk and 30% decrease in the number of axioms required to represent the same knowledge.

As this is first work of its kind, further work is required to provide processes to access, visualize and search such ontological datasets, to list a few. The proposed methodology and the algorithms work well on well-crafted ontologies that have been built using the standardized methodologies. In future, we plan to work for those ontologies also that may not be modelled in the standard form and bring them in the form representing concepts as unit of knowledge and make them more efficient.

For the current work we have taken leaf concepts as leaf classes primarily as the ones that have instances (there may be some exceptions). In future, we plan to expand it to work with instances whether they belong to leaf concept or not. We have multimedia information in our sight as well. Multimedia files are relatively large in size and removing redundancy there have the potential to same huge resources.

One more direction that is planned is to provide ontology design patterns such that new ontologies could be built from scratch taking care of the unit of knowledge concept. This when done will bring a revolution in terms of storage of ontologies and run-time fetching of concepts.

CONFLICT OF INTEREST

The authors of this publication declare there is no conflict of interest.

FUNDING AGENCY

Authors would like to thank National Institute of Technology Kurukshetra, India for financially supporting the research work.

REFERENCES

Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., & Stein, L.A. (2004). OWL web ontology language reference. *W3C Recommendation, 10*(2).

Jain, S., & Jain, N. K. (2008). A generalized knowledge representation system for context sensitive reasoning: Generalized HCPRs System. *Artificial Intelligence Review*, *30*(1-4), 39–52. doi:10.1007/s10462-009-9115-8

Kifer, M. (2008, October). Rule interchange format: The framework. In *International Conference on Web Reasoning and Rule Systems* (pp. 1-11). Springer.

Kurilovas, E., & Juskeviciene, A. (2015). Creation of Web 2.0 tools ontology to improve learning. *Computers in Human Behavior*, *51*, 1380–1386. doi:10.1016/j.chb.2014.10.026

Malhotra, M., & Nair, T. G. (2015). Evolution of knowledge representation and retrieval techniques. *International Journal of Intelligent Systems and Applications*, *7*(7), 18–28. doi:10.5815/ijisa.2015.07.03

Malik, S., & Jain, S. (2021). Sup_Ont: An upper ontology. *International Journal of Web-Based Learning and Teaching Technologies*, *16*(3), 79–99. doi:10.4018/IJWLTT.20210501.oa6

Michalski, R. S. (1983). A theory and methodology of inductive learning. In *Machine learning* (pp. 83–134). Springer.

Miles, A., Matthews, B., Wilson, M., & Brickley, D. (2005, September). SKOS core: simple knowledge organisation for the web. In *International Conference on Dublin Core and Metadata Applications* (pp. 3-10). Academic Press.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., & Miller, K. J. (1990). Intro-duction to WordNet: An on-line lexical database. *International Journal of Lexicography*, *3*(4), 235–244. doi:10.1093/ijl/3.4.235

Patel, A., Jain, S., & Shandilya, S. K. (2018). Data of SemanticWeb as Unit of Knowledge. *Journal of Web Engineering*, *17*(8), 647–674. doi:10.13052/jwe1540-9589.1783

Shandilya, S. K., & Jain, S. (2009, March). Opinion Extraction & Classification of Reviews from Web Documents. In *2009 IEEE International Advance Computing Conference* (pp. 924-927). IEEE. doi:10.1109/IADCC.2009.4809138

Upadhyay, S., Manwani, R., Varshney, S., & Jain, S. (2020). Analytics and storage of big data. *CEUR Workshop Proceedings, 2786*, 202-210.

Zeng, M. L. (2008). Knowledge organization systems (KOS). *Knowledge Organization*, *35*(2-3), 160–182. doi:10.5771/0943-7444-2008-2-3-160

Zeng, M. L., & Mayr, P. (2019). Knowledge Organization Systems (KOS) in the Semantic Web: A multi-dimensional review. *International Journal on Digital Libraries*, *20*(3), 209–230. doi:10.1007/s00799-018-0241-2

ENDNOTE

[1] Available at: https://github.com/abhiseksharma/Ontologies

Abhisek Sharma has been working as a Research Scholar since 2019 in the National Institute of Technology, Kurukshetra, under the supervision of Dr. Sarika Jain, Assistant Professor, Department of Computer Applications. His Ph.D. is in the field of Knowledge Representation and Engineering. He has completed his Master's in Computer Applications in 2018. He has also worked on project funded by DRDO, India. In general, he is driven towards finding solutions for humans' problems when trying to get Multilingual and Multicultural semantically rich results from computers and the issues currently making the processing of semantics by the computers difficult.

Sarika Jain graduated from Jawaharlal Nehru University (India) in 2001. Her doctorate, awarded in 2011, is in the field of knowledge representation in Artificial Intelligence. She has served in the field of education for over 19 years and is currently in service at the National Institute of Technology Kurukshetra (Institute of National Importance), India. Dr. Jain has authored or co-authored over 150 publications including authored and edited books. Her current research interests are knowledge management and analytics, the semantic web, ontological engineering, and intelligent systems. She has been PI of sponsored research projects and works in collaboration with various researchers across the globe, including in Germany, Austria, Australia, Malaysia, Spain, the USA, and Romania. She serves as a reviewer for journals published by IEEE, Elsevier, and Springer. She has been involved as a program- and steering-committee member at many prestigious conferences in India and abroad. She is a senior member of the IEEE, member of ACM, and a Life Member of the CSI.