Journal of Universal Computer Science, vol. 17, no. 7 (2011), 1005-1020
submitted: 30/10/10, accepted: 15/3/10, appeared: 1/4/11 © J.UCS

Algorithms for the Evaluation of Ontologies for Extended Error Taxonomy and their Application on Large Ontologies

Najmul Ikram Qazi
(Center for Distributed & Semantic Computing, Mohammad Ali Jinnah University
Islamabad, Pakistan
najmalikram@yahoo.com)

Muhammad Abdul Qadir
(Center for Distributed & Semantic Computing, Mohammad Ali Jinnah University
Islamabad, Pakistan
aqadir@jinnah.edu.pk)

Abstract: Ontology evaluation is an integral and important part of the ontology development process. Errors in ontologies could be catastrophic for the information system based on those ontologies. As per our experiments, the existing ontology evaluation systems were unable to detect many errors (like, circulatory error in class and property hierarchy, common class and property in disjoint decomposition, redundancy of sub class and sub property, redundancy of disjoint relation and disjoint knowledge omission) as defined in the error taxonomy. We have formulated efficient algorithms for the evaluation of these and other errors as per the extended error taxonomy. These algorithms are implemented (named as OntEval) and the implementations are used to evaluate well-known ontologies including Gene Ontology (GO), WordNet Ontology and OntoSem. The ontologies are indexed using a variant of already proposed scheme *Ontrel*. A number of errors and warnings in these ontologies have been discovered using the OntEval. We have also reported the performance of our implementation, OntEval.

Keywords: Ontology Evaluation, Ontological Engineering, Knowledge Engineering, Knowledge Modelling, Semantic Computing, Information Systems Applications
Categories: M.1, M.2, M.3, H.3, H.4

1 Introduction

As we are moving from traditional Web to the Semantic Web, new technologies are being developed rapidly to facilitate this transition. XML [Bray, 00] came as a powerful language to represent information classified in a hierarchical fashion. Soon, it was realized that we need some thing more expressive than XML. So RDF [Berners-Lee, 01] was introduced, and it was seen as more powerful tool to represent information in the form of triples. As researchers moved ahead, they invented the Web Ontology Language (OWL), which is richer than RDF [Antoniou, 04]. An Ontology [Antoniou, 04] models a domain of interest. It defines domain concepts, their hierarchies, and relationships among them, object and data type properties and many other constructs that enable automated agents to understand and process the

information according to the semantics of the domain. Once the ontology is built, diverse teams can develop information processing systems in their own ways based upon the original or extended ontology. The systems built upon the agreed-upon and correct ontology can interoperate in a meaningful way. Therefore, correct and agreed-upon ontologies play a key role in describing the semantics of data, which enables globally distributed machines to understand the meaning of data.

The ontology evaluation [Antoniou, 04] is an integral part of the ontology development lifecycle. Each ontology should be evaluated against inconsistency, incompleteness and redundancy errors before it is put into operation [Gomez-Perez, 04]. Errors in the ontology can be catastrophic for the information system built on it [Qadir, 07]. Researchers have recently identified new types of errors in ontologies written in OWL-DL that can harm the information systems built upon the ontologies [Qadir, 07], [Fahad, 08b].

Previously identified errors include circulatory errors in class hierarchy, common class in disjoint decompositions, common instances in disjoint decomposition, external instances in exhaustive decomposition, incomplete concept classification, disjoint knowledge omission between classes, exhaustive knowledge omission, redundancy of sub class relation, redundancy of instance-of relations and identical formal definition of classes/instances.

The extended error taxonomy defines new errors as; circulatory errors in property hierarchy, common property in disjoint decompositions, more generalised concept by subclass, domain violation by subclass, disjoint domain by subclass, disjoint knowledge omission between properties, functional and inverse-functional property omission, sufficient knowledge omission, redundancy of sub property, identical formal definition of properties and redundancy of disjoint relation.

A large ontology may contain thousands of concepts and relations, each of them being crucial for the information system. While developing the ontology, domain experts may not remember what they have declared at some other place. Moreover, there may be many experts developing the same ontology. So they may declare some thing wrong, miss some thing, or duplicate some thing. Experts need to check the ontology thoroughly to verify that each concept and its relations have been defined correctly and nothing is missing. This is a very difficult task, and without automated tools, one would always be in doubt.

2 Related Work

Gomez-Perez et al. have proposed a framework for evaluation of ontolingua ontologies as per the design principles of ontologies [Gomez-Perez, 04]. She has contributed a number of situations that can exist in the design of ontologies where an error occurs. On this basis, she formed error taxonomy for assistance of the ontologists.

Qadir et al. [Qadir, 07] and Fahad et al. [Fahad, 08b] revised that error taxonomy and included some new errors that can be harmful to the semantic-based information systems. Fahad [Fahad, 08a] presented a survey of this domain, and present extended error taxonomy for the ontologies [Fahad, 07], incompleteness [Qadir, 07], and redundancy [Noshairwan, 07]. Racer, Pallet and Fact++ are commonly used evaluation tools that come as plug-ins with Protégé. Since the extended errors have

been discovered only recently, these systems cannot detect such errors [Fahad, 07]. There is an urgent need to develop and implement algorithms to detect the extended errors.

Researchers have reported that the previously identified errors are only partially handled by existing systems. Qadir inspects the existing evaluation system's capabilities by artificially inserting these new errors in some ontologies [Qadir, 07]. They reported that even previous errors are not fully handled by these systems. Fahad also introduced errors in some ontologies and inspected the existing systems [Fahad, 08c]. Results show that the dangerous circulatory errors make these systems behave abnormally and even crash.

2.1 OWL DL Reasoning Systems

Protege is a well-known system for ontology development. Ontology engineers use this tool to create an ontology, edit it and do various experiments to validate its design. It provides numerous facilities to view and maintain the taxonomy of terms that are defined in the ontology. An important function of the ontology development tools is reasoning. A reasoner is an agent that goes through the taxonomy of terms (formally known as concepts), properties and the relationships among terms and discovers knowledge from them.

During this process, a reasoner may encounter some errors that were overlokked by the designer. A good reasoning system should detect and report these errors. Since these errors are semantic rather than syntactic, it may be difficult for a reasoner to detect all of them. Protégé provides interoperability with the third party reasoning systems that can be installed as add-ons. We mention three such systems that are popular among the ontology development community.

RACER (Renamed ABox and Concept Expression Reasoner) was developed by Haarslev and Moller of the University of Hamburg [Haarslev, 01] in 2001. It is based on the well-known tableaux calculus. An ontology consists of two parts: Tbox, which defines the terms of the domain and Abox, which maintains the asserted facts about the terms. RACER implements reasoning process for TBox as well as for Abox. RACER is claimed to be the first full-fledged ABox description logic system and its algorithms are sound and complete. In order to provide optimized search, RACER incorporates standard techniques known as dependency-directed backtracking and DPLL-style semantic branching. By implementing these techniques in ABox reasoning, RACER has an edge over the reasoners that provide only TBox reasoning.

Pellet was developed by Parsia et al. In the University of Maryland [Parsia, 05] in 2003. It has a number of distinct features which address the issues of Semantic Web. It is claimed to have the ability to perform some kind of ontology repair. The reasoner uses standard tableau rules and implements various standard optimizations known as dependency directed backjumping, semantic branching and early blocking. The capabilities of Pellet are exposed from Java API, command line and Web forms. Pellet is the default reasoner in Swoop, a lightweight ontology browser and editor. It has some features to detect inconsistent concept description, and future versions are planned to generate explanations for inconsistencies. Although the other reasoning systems are more efficient as compared to Pellet, but they are not as rich in features as Pellet is.

FaCT++ is an improved version of FaCT system, developed in the University of Manchester [Horrocks, 05]. It has limited user interface and services as compared to other reasoners. It implements a number of novel optimization techniques in addition to the standard ones. A new "ToDo list" architecture is used, that is suitable for complex tableaux algorithms and reasoning with OWL ontologies. During a preprocessing phase, an ontology is loaded, normalized and transformed into an internal form. Initial optimizations are performed at this stage. It uses a satisfiability checker, which helps decide subsumption for given concept pairs. Further improvements in the algorithms and technology are planned for future versions of FaCT++. Parallel reasoning process and more elaborate heuristics are also planned.

2.2 Limitations of Existing Systems

In order to investigate how RACER, Pellett and FaCT++ handle various errors, we loaded OntoSem ontology in Protégé, introduced some errors in it, and then performed classification using the three reasoners. The concept "pepper" lies down the hierarchy of concept "material". We defined "material" as the sub class of "pepper". This introduces a circulatory error in the ontology. Next, we define the concept "pepper" as sub class of the concept "physical-object". It was sub class of "spice" in the original ontology and was already down the hierarchy of "physical-object". Making this new relationship introduces redundancy of sub class.
Now we perform classification using the above-mentioned reasoners. Figure 1 shows the strange behavior of RACER. It has treated our definition of "material" as sub class of "spice" to be a normal one, and gave no error or warning. Moreover, it did not detect redundancy of sub class we introduced. It has made all the concepts from "material" down to "pepper" as equivalent concepts and has disturbed all the hierarchies in the range.

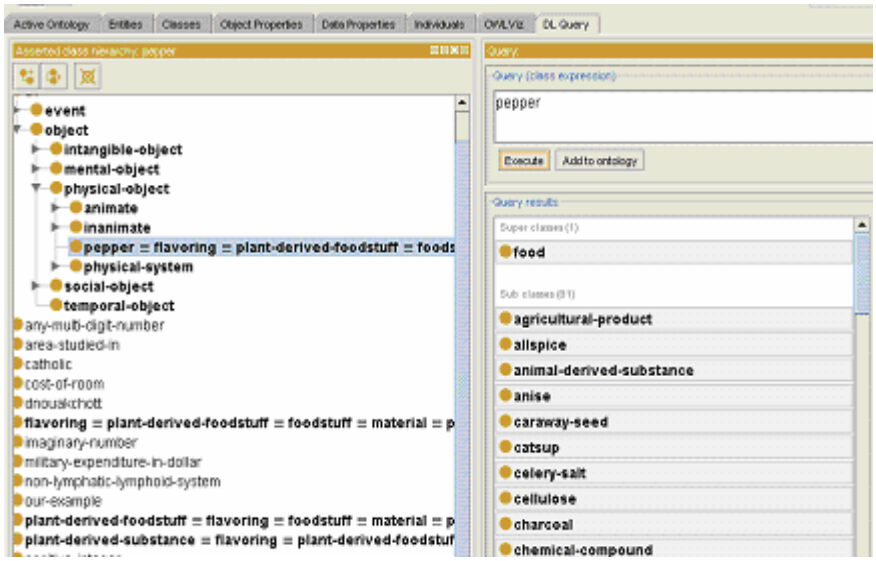


Figure 1: RACER showing strange behavior to circulatory error

3 Ontology Evaluation Algorithms

In the following, we describe how the algorithms are formulated and discuss the approach used to design and implement them. Algorithms' complete listing can be found in [Qazi, 10].

3.1 Efficient Storage and Organization of Ontology

Before we start on the design of algorithms, we have to work out an appropriate scheme for the efficient storage and retrieval of ontologies. We have to take into consideration the possible large size of ontologies. Today's ontologies may consist of tens of thousands of concepts. For example, Gene Ontology consists of more than 29000 concpts, while Wordnet ontology consists of over 60,000 concepts. Future ontologies may contain hundreds of thousands of concepts.

If an indexing scheme is totally based on RAM, it may not be appropriate for large ontologies. On the other hand, a scheme based on both RAM and persistent storage will not suffer from such limitation. Ontology evaluation algorithms have to access the concepts (and properties) of the ontology very frequently. So the scheme has to provide fast access in order to process the ontologies in a reasonable time. We suggest that ontology be represented in the form of indexed lists or tuples. A database system like SQL Server or MySql can be used for this purpose. Figure 2 shows our representation of concept hierarchy in Wordnet ontology.

ConceptHierarchy : Table

ConceptName	ParentConcept
GRENADE	BOMB
GAS_BOMB__GAS_SHELL	BOMB
CLUSTER_BOMB	BOMB
NUCLEAR_EXPLOSION__ATOMIC_EXPLOSION	BOMB_BLAST
B-52	BOMBER
DIVE_BOMBER	BOMBER
BOMBING_RUN	BOMBING__BOMBARDMENT
LOFT_BOMBING__TOSS_BOMBING	BOMBING_RUN
DIVE-BOMBING	BOMBING_RUN
DOMESTIC_SILKWORM_MOTH__BOMBYX_MORI	BOMBYCID__BOMBYCID_MOTH__SILKWORM_MOTH
NONCALLABLE_BOND	BOND__BOND_CERTIFICATE
CORPORATE_BOND	BOND__BOND_CERTIFICATE
COUPON_BOND__BEARER_BOND	BOND__BOND_CERTIFICATE
SECURED_BOND	BOND__BOND_CERTIFICATE
MUNICIPAL_BOND	BOND__BOND_CERTIFICATE
UNSECURED_BOND__DEBENTURE__DEBENTURE_	BOND__BOND_CERTIFICATE
PERFORMANCE_BOND__SURETY_BOND	BOND__BOND_CERTIFICATE

Record: 1 of 61981

Figure 2: Representation of concept hierarchy in Wordnet ontology

3.2 Circulatory Error in Class Hierarchy

Circulatory error means some hierarchy chain makes a cycle. In a correct ontology, traversing down the hierarchy chain starting from a node **d** will take us to a leaf node, and never lead to **d** again. In an ontology with circulatory errors, such traversal will lead to d again and again. This is very dangerous for reasoners because they will stuck in an infinite loop.

This suggests that, to detect circulatory error, we should traverse the nodes of the ontology along the chain of hierarchy. Our algorithm starts from the root(s) node of the ontology and performs depth-first traversal. During this traversal, it maintains a (indexed) list of concepts that form the chain of hierarchy. Concepts should not be repeated in the chain. If they do repeat, it will mean circulatory error. When a new concept is inserted in the chain, it is checked whether it already exists in the chain or not. If it already existed, the circulatory error is reported.

Performance Issues: The simple depth-first search does not pose performance bottleneck. However, checking the chain for repetition will be costly. This is optimized by using indexed list to maintain the chain. To further optimize it, we observe that the circulatory error will occur only at the nodes having more than one parent. So before checking the repetition, the algorithm should check if the node has multiple parents. If not, then the checking should be skipped.

Complexity: Each time when we access a node of the ontology graph, we actually locate it in the list of concept hierarchy tuples. The complexity of this operation depends upon the indexing scheme used. If a database is used for indexing, the complexity will depend upon the database engine. For a reasonable scheme, this complexity should be log n. Our algorithm visits each node of the hierarchy and applies checks on it. Therefore the complexity of the algorithm is n log n.

3.3 Common Class in Disjoint Decompositions

This inconsistency error occurs when the designer declares two concepts as disjoint with each other, but later, declares a concept which is subclass of both the disjoint concepts. Our algorithm starts with the pairs of concepts which have been declared disjoint. It inspects each disjoint pair. For each concept of the pair, the algorithm performs depth-first traversal and stores the descendants of the concept in a list. It then takes intersection of the two (indexed) lists, which should be empty in normal case. If the intersection is not empty, the algorithm gives error.

3.4 Redundancy of Subclass

Redundancy of subclass occurs when a concept **c** is declared as direct child of a concept **p**, whereas, it was already indirect child of **p** through a chain of hierarchy. To detect such redundancy, the concepts having more than one parent are significant. Redundancy of sub class will occur when a concept is child of a concept directly as well as indirectly through some intermediate concepts. This means that such concept will have multiple parents.

3.4.1 Algorithm before optimization

The algorithm uses greedy approach to detect redundancy of subclass. It starts with determining the list of nodes ***M*** having more than one parent. It takes first node **a** having multiple parents. It selects first parent **p** of **a** to traverse upwards. Before starting the traversal, all the parent nodes of **a** other than **p** are stored in a list **L**. Then a traversal is performed upwards, starting from the selected parent node **p**. During this traversal, each visited node is compared with the list **L**. Since the nodes in **L** are immediate parents of **a**, and the traversal is done through the indirect ancestors of **a**, therefore no match should be found.

When some concept traversed matches with one in the list, it means redundancy. The concept is immediate parent of **a** since it is in the list, and is indirect parent of **a** because it is encountered during upward traversal.

Performance Issue: Although this algorithm produces correct results, it has an inherent performance problem. Figure 3 depicts a pattern of nodes which represents complexity of some ontologies like Gene Ontology. Node **d** has several children and each one of them will have further descendants. It has two parents, and each one has in turn two parents. When traversing upwards from a node like **d**, algorithm writers may not expect that the algorithm will diverge exponentially, whereas in practice, it may do so. To inspect one node below **d**, the algorithm visits several nodes along its chain of hierarchy. A large portion of this chain will be visited again, when it inspects some other nodes. This repetition, although difficult to avoid completely, causes serious performance problem. We overcome this performance problem by employing another approach to detect redundancy.

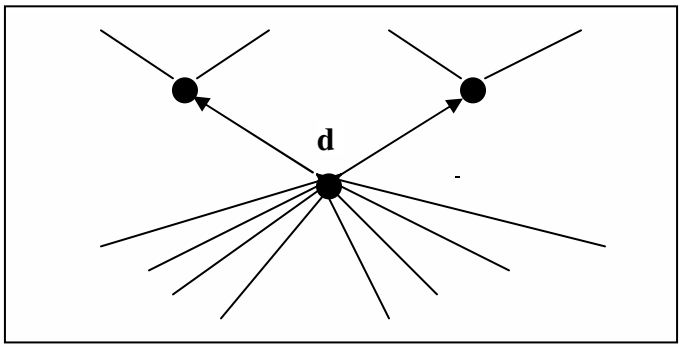


Figure 3: A pattern representing complexity of Gene Ontology

3.4.2 Optimized Algorithm

The optimized algorithm avoids the repeated processing of same nodes. To achieve this, it traverses the ontology in a top-down depth-first manner. When it visits a node, it saves it in a (indexed) list **L**. When it leaves a node, it deletes it from **L**. So at any point, the list contains the chain of hierarchy from root to the current node. When the algorithm visits a node **b**, it also checks its immediate parents **P**. If the node has only one parent, there is no question of redundancy of sub class. If the node has more than one parent, it checks whether any of the parents, other than that in the chain of hierarchy, also exists in the chain. In an ontology free of redundancy, it should not be present in the chain.

If any of the immediate parents also exists in the chain of hierarchy, it means that this node has been declared the parent of **b** twice: once as direct parent and once as indirect parent. So the redundancy error is reported.

```
SCRed(Ontology O)
{
  R = root(O)
  Ω = {R}
   CheckSCRed (R, Ω)
}
CheckSCRed (α,Ω)
{
   For all β ε children (α)
   {
      Ω = Ω U {β}
       Π = Parents(β) \ {α}
       For all γ ε Π
       {
            If γ ε Ω Then
                 Redundancy of subclass found
       }
      CheckSCRed (β,Ω)
      Ω = Ω \ {β}
   }
```

Figure 4: Revised algorithm to diagnose redundancy of sub class

3.5 Redundancy of Disjoint Relation

The algorithm to detect redundancy of disjoint relation loops for every disjoint declaration in the ontology. For each pair of concepts ***(d1, d2)*** declared disjoint, it inspects whether their ancestors are also declared mutually disjoint. It takes the first concept of the disjoint pair and visits its ancestors up the hierarchy. For each ancestor visited, it checks whether there is some disjoint declaration for it. If there is a disjoint declaration, it notes the disjoint concept. Then it checks whether this disjoint concept is among the ancestors of ***d2***. If so, the redundancy is reported.

3.6 Disjoint Knowledge Omission

Ontology designers may omit disjoint declaration between concepts which are actually disjoint in the domain. It is strongly recommended that if the two concepts are disjoint in the domain, they should be declared so. This helps information systems in better reasoning.

The algorithm given in Figure 5 detects the disjoint knowledge omission in an ontology. In the start, it initializes a (indexed) list of all disjoint declarations Ω in the ontology. The algorithm has to repeat for every pair of sister concepts in the ontology. This is quite expensive due to the large number of possible combinations of sister concepts. One solution for this is to prioritize the search. Inspecting the nodes higher in the hierarchy has more chances of finding the disjoint knowledge omission cases.

For each pair of sister concepts *(β1, β2)*, the algorithm first checks if the two concepts are declared disjoint, by searching them in the list of disjoint declarations *Ω*. If they are not declared disjoint, the algorithm checks if there are common concepts among the children of the two concepts. It visits the nodes down the hierarchy of the first node, and prepares a (indexed) list of these nodes *Π*. It does the same for the second node, and makes the second list *Π'*. It then takes intersection of the two lists. If the intersection is empty, then the algorithm considers the two concepts as the candidates for disjoint knowledge omission. It stores the two concepts in a list, along with the number of descendants.

When the list is presented to human expert, the number of descendants is helpful in ranking. The candidate cases which have higher number of descendants, have more chances to be endorsed by human expert as the valid disjoint omission case.

```
DisjointOm(Ontology O, Depth)
{
   Ω = {Disjoint declarations in O}
    For each node α of depth <= Depth
    {
      For each pair (β1, β2) ε ImmChildren (α)
      {
        If not (β1, β2) ε Ω
        {
          Π = FindDesc(β1)
          Π' = FindDesc(β2)
            If Π ⋂ Π' = ϕ Then
               Warning for disjoint omission
        }
      }
   }
}
```

Figure 5: Algorithm to diagnose disjoint knowledge omission

4 Evaluation of well known ontologies

The algorithms were applied to evaluate three well known ontologies: OntoSem, WordNet and Gene Ontology (GO). Since many systems are dependant on them, evaluation of these ontologies will be an important contribution.

4.1 WordNet Ontology

WordNet® is a large lexical database of English, developed in Princeton University. Nouns, verbs, adjectives and adverbs are grouped into sets of synonyms (synsets), each representing a particular concept. Synsets are linked together by means of conceptual-semantic and lexical relations. This makes a network of meaningfully

related words and concepts which can be processed by software systems. WordNet is an open system and is freely available for download. WordNet's structure makes it a useful tool for computational linguistics and natural language processing. The goal of WordNet was to develop a system that would be consistent with the knowledge acquired over the years about how human beings process language.

WordNet Ontology organizes the concepts and properties in a hierarchical manner, specifying generelization and specialization relations. The network of nouns is far deeper than that of the other parts of speech. Verbs have a bushier structure, and adjectives are organized into many distinct clusters. Adverbs are defined in terms of the adjectives they are derived from

Figure 6 shows one of three circulatory errors in class hierarchy found in Wordnet ontology. Region is defined as being a Location. Location is a Space region, Space region is a Physical region, while Physical region is a Region. This makes a circle. These errors have serious consequences for the systems built on this ontology. For example, a system needs to find all descendants of concept 'space-region'. It will repeatedly visit the four concepts in the circle and will get stuck in the infinite loop. The right side of Figure 6 shows a redundancy of sub class.

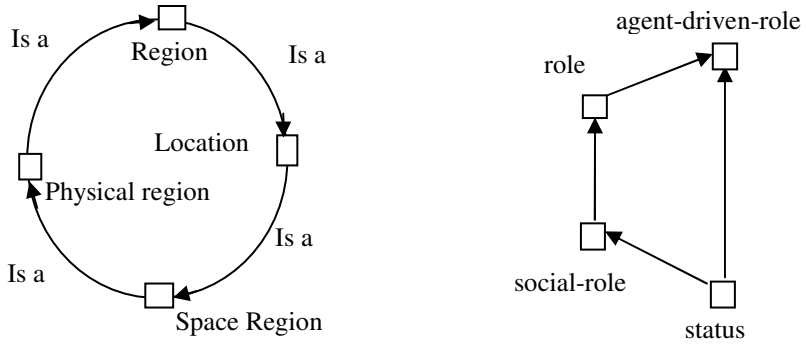


Figure 6: Left: a circulatory error, right: a redundancy of sub class in Wordnet ontology

List of all three circulatory errors in class hierarchy is given in Figure 7 while Figure 8 lists two cases of redundancy of sub class found in this ontology.

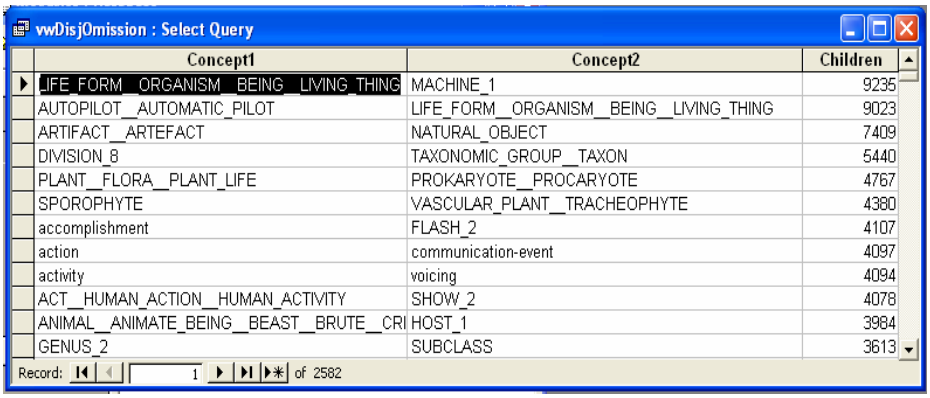
1. region *is-a* location *is-a* space-region *is-a* physical-region *is-a* region
2. method *is-a* know-how *is-a* ability_power *is-a* technique *is-a* method
3. gestalt *is-a* form_shape_pattern *is-a* gestalt

Figure 7: Three circulatory errors in WordNet Ontology

***Parent*:** geographical-object
***Child*:** GEOGRAPHIC_POINT__GEOGRAPHICAL_POINT
***Redundant Hierarchy*:** geographical-object**region **physical-region **space-region**LOCATION**POINT_6
**GEOGRAPHIC_POINT__GEOGRAPHICAL_POINT
***Parent*:** agent-driven-role
***Child*:** status
***Redundant Hierarchy*:** **agent-driven-role**ROLE **social-role**status

Figure 8: Redundancy of sub class in WordNet Ontology

Our algorithm was applied on the Wordnet ontology to find the candidate cases for disjoint knowledge omission. It found over 48,000 candidate cases and took 90 minutes for complete processing. It was observed that number of children of the candidate concepts is a relevant factor. It helps the human expert in determining the real cases from the candidates and saves his time.
We applied a threshold of 20 on the number of children and processed the ontology again. This means that the leaf nodes and nodes slightly above the leaves are eliminated from the candidate cases. The algorithm then produced 2582 candidates. These were sorted on the number of children in descending order, further assisting the human expert as shown in Figure 9.


vwDisjOmission : Select Query

Concept1	Concept2	Children
LIFE_FORM__ORGANISM__BEING__LIVING_THING	MACHINE_1	9235
AUTOPILOT__AUTOMATIC_PILOT	LIFE_FORM__ORGANISM__BEING__LIVING_THING	9023
ARTIFACT__ARTEFACT	NATURAL_OBJECT	7409
DIVISION_8	TAXONOMIC_GROUP__TAXON	5440
PLANT__FLORA__PLANT_LIFE	PROKARYOTE__PROCARYOTE	4767
SPOROPHYTE	VASCULAR_PLANT__TRACHEOPHYTE	4380
accomplishment	FLASH_2	4107
action	communication-event	4097
activity	voicing	4094
ACT__HUMAN_ACTION__HUMAN_ACTIVITY	SHOW_2	4078
ANIMAL__ANIMATE_BEING__BEAST__BRUTE__CRI	HOST_1	3984
GENUS_2	SUBCLASS	3613

Record: 1 of 2582

Figure 9: Candidate cases of disjoint knowledge omission in WordNet Ontology

At the top of the list, there are two candidates having over 9,000 children. The algorithm has highlighted them from thousands of cases to facilitate the human expert. We can see that both these cases are rightly identified as those of disjoint knowledge omission. We can strongly recommend that they should be declared as disjoint. By looking at the candidates, we have identified some cases which show that the disjoint knowledge was omitted. They are shown in Figure 10.

	Concept1	Concept2	Children
▶	AUTOPILOT__AUTOMATIC_PILOT	LIFE_FORM__ORGANISM__BEING__LIVING_THING	9023
	NURSING_CARE	THERAPY	111
	BACTERIA__BACTERIUM	VIRUS	52
	POSITION__SPATIAL_RELATION	TEMPORAL_RELATION	120
	LEAF__LEAFAGE__FOLIAGE	ROOT_1	87
	DEFINITE_QUANTITY	RELATIVE_QUANTITY	732
	GAS_2	LIQUID_2	64
	PROOF__COGENT_EVIDENCE	SYMPTOM_2	303
	linguistic-object	SIGNAL__SIGNALING__SIGN	545
	FIRMWARE	SOFTWARE__SOFTWARE_SYSTEM	69
	CHANGE_OF_LOCATION__TRAVEL	HARMONIC_MOTION	80
	BODY_PART	PLANT_PART	2274
	CHEMICAL_PHENOMENON	PHYSICAL_PHENOMENON	460
	FUNGUS_GENUS	PLANT_GENUS	184
	LINE_8	ROUND_SHAPE	149
	AIRCRAFT	VESSEL__WATERCRAFT	215
	AMMUNITION__AMMO	WEAPON__ARM__WEAPON_SYSTEM	137
	BLACK_HOLE	geographical-object	2238
	IRRITATION__ANNOYANCE__VEXATION	TRAUMA	5

Figure 10: Some cases of disjoint knowledge omission after human inspection

Disjoint omission between Autopilot and Living_Thing is of special interest. Without disjoint, it will be possible to define a concept which is both autopilot and living thing at the same time. The ontology will allow that, causing strange effects. So it is recommended that these two concepts be declared disjoint.

4.2 OntoSem Ontology

OntoSem ontology is a large ontology and knowledge representation system for language understanding tasks developed by the Institute for Language and Information Technologies (ILIT). OntoSem contains natural language words organized as a hierarchy of concepts and properties. Like WordNet Ontology, this ontology is useful for natural language processing.

We found 10 redundancy of sub class errors in OntoSem, five of which are shown in Figure 11. Figure 12 shows some cases of disjoint knowledge omission in OntoSem Ontology.

***Parent**: independent-device, **Child**: buzzer*
Redundant Hierarchy**: *** independent-device communication-device*** buzzer*
***Parent**: non-work-activity, **Child**: dance*
Redundant Hierarchy**: *** non-work-activity hobby-activity*** artistic-activity*** dance*
***Parent**: artifact-part, **Child**: handle*
Redundant Hierarchy**: *** artifact-part furniture-part*** handle*
***Parent**: printed-media, **Child**: illustration*
Redundant Hierarchy**: ***** printed-media*** picture*** illustration
***Parent**: artifact-part, **Child**: wheel*
Redundant Hierarchy**: ***artifact-partvehicle-part***wheel*

Figure 11: Five out of 10 redundancy of subclass errors in OntoSem ontology

Concept1	Concept2	Children
communicable-disease	non-communicable-disease	182
animal-derived-foodstuff	plant-derived-foodstuff	305
criminal-activity	sports-activity	89
human	spider-monkey	381
metallic-liquid-element	metallic-solid-element	100
corporation	sole-proprietorship	198
animal-living-event	plant-living-event	315
printed-media	television	145
film-artifact	printed-media	158
governmental-organization	private-organization	285

Figure 12: Some disjoint knowledge omission cases in OntoSem ontology

4.3 Gene Ontology

The Gene Ontology project is a well-known bioinformatics initiative with the aim to standardize the representation of gene and gene product attributes across species. The project makes available a vocabulary of terms that describe gene product characteristics and gene product annotation data from GO Consortium members.
The Gene Ontology (GO) project is a collaborative effort to meet the need for consistent descriptions of gene products in different databases. The project began in 1998 as a collaboration among three famous organism databases: FlyBase (Drosophila), the Saccharomyces Genome Database (SGD) and the Mouse Genome Database (MGD). Since then, the GO Consortium has grown to include many databases, including several of the world's major repositories for plant, animal and microbial genomes.
We found Gene Ontology free of circulaory and redundancy of sub class errors. Figure 13 shows candidate cases of disjoint knowledge omission in Gene Ontology.

Initial informal inspection of these cases by human expert leads to interesting and useful results.

vwDisjOmission : Select Query

Concept1	Label1	Concept2	Label2	Children
GO_0044236	multicellular organismal metabolism	GO_0044237	GOC:mah	29153
GO_0044091	GOC:jl	GO_0044237	GOC:mah	29122
GO_0019222	regulation of metabolism	GO_0031341	GOC:mah	28922
GO_0044238	primary metabolism	GO_0045730	Wikipedia:Respiratory_burst	14579
GO_0043112	receptor metabolism	GO_0043283	biopolymer metabolism	7936
GO_0003824	Wikipedia:Enzyme	GO_0005198	GOC:mah	7827
GO_0030155	cell adhesion receptor regulator activity	GO_0031323	regulation of cellular metabolism	7727
GO_0048583	GOC:jic	GO_0050792	GOC:tb	6400
GO_0060255	GOC:tb	GO_0060263	GOC:tb	5594
GO_0050792	GOC:tb	GO_0050793	GOC:go_curators	4748
GO_0005198	GOC:mah	GO_0005215	Reactome:1391	4722
GO_0060263	GOC:tb	GO_0080090	PMID:19211694	4296

Record: 1 of 2008

Figure 13: Candidate cases of disjoint knowledge omission in Gene Ontology

4.4 Evaluation of Algorithms

The algorithms were first run on fictitious ontologies to verify their correctness. Then they were tested on the real ontologies. Errors were induced in the ontologies and then algorithms were applied on them. The algorithms were found correct since they successfully detected the induced errors. After verification of correctness, the algorithms were applied on original ontologies to detect the actual errors. The actual errors were again verified manually.

Table 14 shows the number of errors detected and (time taken in minutes) by some of our algorithms to process three well known ontologies. To our surprise, redundancy of sub class algorithm took 420 minutes on Gene Ontology containing 29,534 concepts. This is drastically high as compared to 2.5 minutes on WordNet ontology containing 61,299 concepts. An analysis of Gene Ontology shows complex graph structure of this ontology where many concepts have multiple parents. When the algorithm scans ancestors of a node, the scan diverges in exponential manner, thus incurring high computational cost. Therefore, concepts having multiple parents should be given due consideration while designing algorithms for ontologies. The revised algorithm outperforms normal algorithm in Gene Ontology, but not in other ones, where normal algorithm proves more efficient.

Error	OntoSem	WordNet	Gene Ontology
Circulatory error in class hierarchy	0 (0.7)	**3** (1.8)	0 (10)
Common class in disjoint decompositions and partitions	0 (0)	0 (27)	0 (3.5)
Redundancy of Subclass-of relation	**10** (0.7)	**2** (2.5)	0 (420)
Redundancy of Subclass **(revised)**	**10** (15)	**2** (18)	0 (23)
Disjoint Knowledge Omission (warnings)	404(7)	2582(55)	2008 (65)

Table 14: Errors detected, time taken (in minutes) in 3 well known ontologies

5 Conclusions and Future Work

The existing facilities for detection of inconsistency, redundancy and incompleteness errors in the ontologies are inadequate as demonstrated in this paper. In order to develop high quality ontology-based systems, such facilities are needed urgently. Even the well-known ontologies contain errors, that were left unchecked by the existing tools. To fill this gap, the algorithms to evaluate ontologies against published error types are designed and implemented. The algorithms are tuned for optimum performance, and are able to process large ontologies in reasonable time. By carefully designing algorithms, we can reduce the processing time by more than 10 times. The algorithms do not require to load the entire ontology in memory, and therefore do not put limit on the size of ontology being processed. Various errors have been reported in the well known ontologies: Gene Ontology, Wordnet Ontology and OntoSem Ontology. The algorithms will be useful for researchers working in Semantic Computing.

The algorithms may be used to evaluate more ontologies to improve their accuracy. The errors, specially disjoint knowledge omission may be verified by human experts. Algorithms may be further tested for correctness and performance.

References

[Antoniou, 04] Antoniou, G. and Harmelen, F.V., (2004): A Semantic Web Primer. MIT Press Cambridge, ISBN 0-262-01210-3

[Berners-Lee, 01] T. Berners-Lee, J. Hendler, and O. Lasila: The Semantic Web. Scientific American 284, May 2001, 34-43.

[Bray, 00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, eds: Extensible Markup Language (XML) 1.0, 2nd ed, W3C Recommendation, October 6, 2000, http://www.w3.org/TR/REC-xml.

[Corcho, 05] Corcho et at., (2005): A survey on ontology tools. Technical report no. D 1.3

[Denny, 04] Denny, M., (2004): Ontology Editor Survey Results. Technical report.

[Fahad, 07] Fahad, M., Qadir, M.A., and Noshairwan, W., (2007): Semantic Inconsistency Errors in Ontologies. Proc. of Intl Conf. on Granular Computing, Silicon Valley, USA, IEEE.

[Fahad, 08] Fahad, M. (2008): Ontology Evaluation, Mapping and Merging. M.S Thesis, Muhammad Ali Jinnah University Islamabad.

[Fahad, 08a]Fahad, M., Qadir, M. A., Noshairwan, M. W., (June 2008): Ontological Errors: Consistency, Incompleteness and Redundancy. Proceedings of 10th International Conference on Enterprise Information Systems (ICEIS'08). Barcelona, Spain.

[Fahad, 08b]Fahad, M., and Qadir, M.A., (July 2008): A Framework for Ontology Evaluation. Proceeding of 16th International Conference on Conceptual Structures (ICCS'08), Toulouse, France. Vol-354, pages 149-158.

[Fahad, 08c]Fahad, M., Qadir, M.A., and HussainShah, S.A., (Oct 2008c): Evaluation of Ontologies and DL Reasoners. Proceeding of 5th International Conference on Intelligent Information Processing (iip'08) Oct 2008. Beijing China.

[Gomez-Perez, 94] Gomez-Perez, A., (1994): Some ideas and examples to evaluate ontologies. Knowledge Systems Laboratory, Stanford University.

[Gomez-Perez, 99] Gomez-Perez et al., (1999): Evaluation of Taxonomic Knowledge on Ontologies and Knowledge-Based Systems. International Workshop on Knowledge Acquisition, Modeling and Management.

[Gomez-Perez, 04] Gomez-Perez, A., Fernández-López, M., Corcho, O., (2004): Ontological engineering: With examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web. Springer, London.

[Haarslev, 01] Haarslev, V. and M¨oller, R., (2001): Racer system description. International Joint Conference on Automated Reasoning, IJCAR' 01, June 18-23, Siena, Italy, Springer-Verlag (2001) 701–705.

[Horrocks, 05] Horrocks, I., Sattler, U., (2005): A tableaux decision procedure for SHOIQ. Proceedings of Nineteenth International Joint Conference on Artificial Intelligence.

[Khalid, 09] Khalid A., Shah S.A.H., Qadir M.A., (2009): OntRel: An Ontology Indexer to Store OWL-DL Ontologies and Its Instances. International Conference of Soft Computing and Pattern Recognition Malacca, Malaysia.

[Noshairwan, 07] Noshairwan, W., Qadir, M.A., Fahad, M. 2007: Sufficient Knowledge Omission error and Redundant Disjoint Relation in Ontology, InProc. 5th Atlantic Web Intelligence Conference June 25-27, 2007 - Fontainebleau, France.

[Parsia, 05] Parsia, B., and Sirin, E., (2005): Pellet: An owl dl reasoner. Proc. International Semantic Web Conference.

[Qadir, 07] Qadir M.A., Nosherwan W., (2007): Warnings for Disjoint Knowledge Omission in Ontologies. 2nd International Conference on Internet and Web Applications and Services (ICIW 07).

[Qazi, 10] Qazi N.I., Qadir, M.A. (2010): Algorithms to Evaluate Ontologies Based on Extended Error Taxonomy. International Conference on Information and Emerging Technologies (ICIET 10).