MSc Artificial Intelligence

Track: track

Master Thesis

Designing custom knowledge bases For inconsistency work

by

Thomas de Groot

student number

January 14, 2019

Number of Credits

Period in which the research was carried out

Supervisor:
Dr A Person

Assessor:
Dr A Person

institute name

Definitions

Acknowledgments

Abstract

The development of larger knowledge based systems is growing rapidly. and the reasoners over these large datasets are following quickly behind. While reasoning over these large knowledge graphs is improving ++ADD IN CITATION++, it is mandatory that these knowledge systems are consistent. With one inconsistency the knowledge graph can break and it is no longer possible to reason of these graphs. Several methods exists that clean the knowledge bases from these inconsistencies. Other methods try to reason around the inconsistencies or use other methods to incorporate the knowledge in their reasoners. While most methods work well, the test cases that are used for these models are not a great representation of the complete world wide web of linked data. Most of the test cases are selected for their characteristics, or the datasets are specifically designed for the test purposes of the method.
To improve the general availability of inconsistent knowledge bases we designed an general knowledge base generator that uses generalized forms of inconsistencies found in the LOD-a-LOT ++ADD IN CITATION++ and use these inconsistencies to build an inconsistent knowledge base that is designed according to a set of parameters that can be given by the user.

Contents

1 Introduction **1**

2 Related Work **2**

3 Preliminaries **3**

3.1 Data 3

4 Approach & Method **4**

4.1 Retrieval of the inconsistent patterns 4

4.2 Generalizing the patterns on basis of isomorphic graphs 4

4.3 Generalizing the patterns on basis of 5

4.4 Locating the most used inconsistency cases 5

5 Experiments **6**

6 Results **7**

7 Conclusion **8**

8 Bibliography **9**

9 Appendices **10**

9.1 Appendix A 10

1 Introduction

2 Related Work

3 Preliminaries

Explain LOD-a-lot Explain hdt

3.1 Data

4 Approach & Method

This chapter will explain in detail how the implementation of the model has been done. Explaining how the inconsistencies were retrieved from the LOD-a-lot, and generalized. Then the chapter goes more in detail about how the generalized inconsistencies are used in the experiments.

4.1 Retrieval of the inconsistent patterns

¿Why are the subgraphs needed?

A reasoner is limited in the amount of data it can reason over within limited time. As more data means many different facts which will take more time to reason over. Even with optimizations it is not possible to reason over the LOD-a-lot without using massive amounts of CPU or taking a very long time. To improve the time needed to reason over we can either add more resources or make the amount of data to reason over smaller. The first option is not viable, but the second option is. It does however mean that we need to generate correct subgraphs without losing any of the inconsistencies that could be found only in the complete graph.

¿So how are the subgraphs generated?

Generating the subgraphs without losing information is tricky as links are broken to retrieve a small part of the graph from a large graph. The goal is to minimize the loss by making sure that the severed links will be added in later stages of the pass through the complete graph.

To retrieve subgraphs from the graph we use rootnodes. Generating rootnodes is done by taking a triple from the complete graph as our starting point. Then for this triple the subject is chosen as the root node. This is the node from which the subgraph is generated. To build the subgraph the node expanded upon such that a graph will be generated around the rootnode. The rootnode is expanded by recursively taking all the triples for which the rootnode is the subject, then if the amount of triples that has been added stays under a limit all the objects are added to the list and are now used as root nodes. These will then be questioned and expanded upon with the rootnode, expanding the graph quickly if there are many links and more slowly with less links. After doing this until a stopping criteria is reached. A limit on the amount of triples or no more links can be found. Due only going into one direction namely the switching to the object position the amount of links is quite often limited. It is also expected that most to all inconsistencies will be found as the limit can be set to an acceptable amount and with the expectation that all links will be visited the amount of inconsistencies will be almost complete. We can not however say that this will find all inconsistencies as it would be possible to design an inconsistency that is larger than the limit set. ¿¿But I Expect that this will not happen.

So how are the inconsistencies retrieved from the subgraphs?
Now that the large knowledge graph has been split into multiple smaller subgraphs we can start to locate the inconsistencies that coincide in these subgraphs. This is done by employing an reasoner that can locate inconsistencies, the importance is though that this reasoner does not reason over triples and adds in new triples that the reasoner could implicitly add. This way we do not find inconsistencies that could occur in the dataset without the user knowing, or that the reasoner the inconsistency ”shrinks” due to the reasoner being able to shortcut inconsistencies, as seen in the EXAMPLE.

Each subgraph can have multiple inconsistencies, so each subgraph is checked extensively for inconsistencies. ¿¿TO AVOID DUPLICATES: ¡¡ Each inconsistency in handled seperately.

4.2 Generalizing the patterns on basis of isomorphic graphs

Why is generalization needed?
All inconsistencies are different, but the inconsistencies can be in essence one and the same inconsistency. As the amount of inconsistencies rise to more than a ¿massive number¡, it would be more manageable if we shrunk the list of different inconsistencies down by finding generalization between the inconsistencies, as multiple inconsistencies can be equal when we remove the specific classes and instances hanging on the vertices. This could lower the amount of inconsistencies while it is still possible to retrieve these specific inconsistencies with simple SPARQL queries that can be run over the graph.

So how are the inconsistencies generalized?
The first step is figuring out how it is possible to generalize the patterns without losing information about the inconsistencies or at least a minimal amount. The first step can be started with the notion of isomorphic graphs. Isomorphic graphs are a great way to generalize graphs without losing information that is irreversibly lost. But as the information on the vertices is not relevant for the inconsistencies the information in on the edges is important. As the connections between the nodes describe the possibility to be inconsistent as described in the ¿EXAMPLE¡. While de edges are also important for the isomorphic graphs. As two graphs could be isomorphic

according to normal standards, with the notion that the edges also must align graphs could now be no longer isomorphic. See ¿EXAMPLE¡

The problem however is that to check if two graphs are isomorphic is difficult to test ¿CITATION¡. To improve the efficiency of the isomorphic matching the amount of graphs are pruned such that matching the graphs will not happen with graphs that can in no way be isomorphic. There are a number of characteristic before a graph is isomorphic:
- It needs to have the same number of vertices.
- It needs to have the same number of edges.
- It needs to have the same amount of degrees.
- In our case it also needs to have the same amount of edges based on the edge types we have.

If two graphs do match we then run the matching algorithm and only if the matching algorithm returns true the two graphs are matched. The algorithm that is used in matching is explained here ¿ALGORITHM¡

4.3 Generalizing the patterns on basis of ...

¿¿¿¿¿¿¿¿.................................¡¡¡¡¡¡¡¡¡

4.4 Locating the most used inconsistency cases

Finished with the generalization is it now possible to be able to retrieve meaningful statistics about the generalized inconsistencies. Interesting statistics are the top 10 most common mistakes:

1. Import the LOD-a-lot
2. Retrieve a sub graph of the LOD cloud by choosing a vertex as root node and retrieving a small part of LOD-a-lot by expanding the root node.
3. Check the sub graph for inconsistencies.
4. Retrieve the inconsistencies and check if they can be generalized.
5. Generalize on basis of isomorphic, check if vertex isomorphic and edge isomorphic
6. Add in ... steps to further generalize.
7. Count the amount of occurrences per ”generalized” inconsistent subgraph
8. Make a generator that uses input from the generalized inconsistencies and user given parameters to build a knowledge base.
9. The generator uses a .... ALGORITHM to build these graphs from the LOD-a-lot Cloud.
10. The generator returns the graph to the user.
11. Testing the generated graphs with the method to check if the consistencies are correct.
12. Implement several methods to test whether these methods perform as well as that the researchers propose.
13. Show results.

5 Experiments

Possible Experiments for testing: https://openproceedings.org/2018/conf/edbt/paper-61.pdf

6 Results

7 Conclusion

8 Bibliography

9 Appendices

9.1 Appendix A