MSc Artificial Intelligence

Track: track

Master Thesis

Designing custom knowledge bases For inconsistency work

by

Thomas de Groot

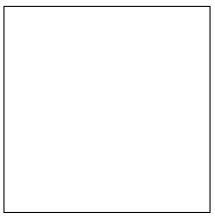
student number

January 11, 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

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 slowely 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.

4.2 Generalizing the patterns

Why is generalization needed.

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