MSc Artificial Intelligence
Master Thesis

Designing custom inconsistent knowledge graphs

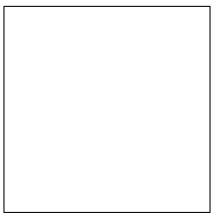
by
Thomas de Groot
11320303

February 12, 2019

36 EC
September 2018 - March 2019

Supervisor:
Dr A Person

Assessor:
Dr A Person



institute name

Definitions

Acknowledgments

Abstract

The development of larger and large knowledge base systems is growing with an ever larger speed. Even though full scale developed reasoners are still lacking behind to reason over these large knowledge bases, new algorithms are designed that can still retrieve useful, although sometimes limited entailments over the large knowledge bases. The only problem however it that these knowledge bases need to be consistent. Reasoning over inconsistent knowledge bases is a larger challenge.

While several methods exists that can remove or repair the inconsistencies, most of these algorithms can not be easily tested on the larger knowledge bases. To solve this problem, this paper proposes a generator that can reduce the size of the knowledge bases while keeping the characteristics of the knowledge base in tact. With this technique the test time of algorithms can be reduced while the generated knowledge bases are still accurate representations of the knowledge base.

To showcase the generator and 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 [3] 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

Enormous linked knowledge bases surround us in our daily life. With many teams working to get the most out of the large knowledge graphs that are available to us. Challenges that can consist out of the removal and cleansing of errors. Improving linking between entities or relation linking. Building larger and larger linked knowledge graphs with a minimal to no inconsistencies. While all these tasks can be done by hand, it would be tedious work and probably full of mistakes. To overcome this, we employ the specially designed algorithms to solve these problems for us. The problem however is that knowledge bases can have completely different characteristics, even in such a way that an algorithm for one particular knowledge base could in theory fail on a different knowledge base. [STRONG CLAIM. FIND PAPER!]
Algorithms know that try to tackle these tasks are now tested on one or multiple different knowledge graphs that are used in other papers. The problem however is that the knowledge graphs that are being used are hard to compare and some of the knowledge graphs are specifically build for testing. This could affect the algorithms, as they are tailored towards scoring as high as possible on the testcases. It would therefore be useful to have a larger variety of graphs ready for testing, or even a generator that could generate the knowledge bases.
So in this paper we do exactly that, in this paper we propose a generator that can generate a knowledge base on basis characteristics that are given by the user or can be retrieved from the inserted knowledge graph. The generator would on basis of the characteristics design a knowledge base that would fit the graph as closely as possible.
This subject is relatively new, there are no real knowledge graph generators that exist. Most knowledge graphs use existing data and with the help of experts and text processing new graphs are generated. There is no technique yet that returns a graph that based on user input on the amount of vertices and edges, the types of connections and the types of inconsistencies. There is however work related to locating and repairing inconsistencies, we use part of the localization to find minimal inconsistency patterns on which the generation process is designed. To find the patterns with the correct information about the inconsistencies we use techniques to find these smaller inconsistent subgraphs. But the generation of the work is based on the new research.
While it is hard to compare this new method to other techniques in the field, as most research is build to find and repair inconsistencies not to build a generator, we showcase the advantages are over using this technique, with knowledge graphs that have been used. This makes it hard to compare to other techniques. It is however a good to test this method to already existing and used knowledge graphs in the field. The method as shown in this paper is .... than the existing method because the knowledge bases this method produces is smaller in comparison to the new dataset thus it will return results much faster, while the knowledge graph within the smaller knowledge graph is still similar to the larger graph.
How is this method tested?
The first chapter explains the work that this paper is based on, as well as shows the work that others have done before us that is related or could benefit from the work we have done. The second chapter goes into the preliminaries, what is needed to even start the process what for preliminary work has been done to get the method working. The Approach and method explain more in detail how the inconsistent knowledge graphs are generated and how they are build according to some sets of standards. The experiments showcase the possible applications of this knowledge base generator and the possibility to generate smaller subsets of existing knowledge bases that do still match the existing knowledge bases but are only smaller in comparison. In the results the generated models are compared on bases of a set of criteria to be added in and the speed of the algorithm for retrieving these smaller knowledge bases. We conclude with a conclusion and discussion about the use cases, with an extension of future work.

2 Related Work

While there is a large community on trying to build "consistent" knowledge bases on data that already exists. Or merging knowledge bases with possible conflicting data or triples, no found work is done on actively building and designing real inconsistent knowledge bases. It is therefore not possible to match this problem to any of the state of the art solutions as there are none.

While this subject is relatively new, the research in this subject connects to a multitude of other papers on which parts of this paper are loosely based, as well as that several test cases come forth out of these papers. [5] shows the necessity of a knowledge base generator. In his paper a set of the largest Linked Datasets are compared on a large set of different criteria and show a framework for given a set of criteria which dataset would most closely resemble your expectations. While this is a good way to select your knowledge base for your experiments, this paper takes a step further and is would use the input of the user to generate a database that would closely resemble the larger database while giving the user the freedom to choose the amount of triples and the types of inconsistency.
[8] showcases the need for a standardized evaluation method. In the survey the paper the research is based on shows that the papers choose sometimes different knowledge graph(s). This makes it harder to compare. Giving the user a hard time figuring out which research paper produces a good result in comparison with a different paper. Removing the this discrepancy would benefit the community as now it will be easier to compare algorithms.

There is work done on inconsistency locating: `https://github.com/raki123/inconsistent-subgraph-query`
A generator for databases: UIS Database Generator

3 Preliminaries

Before we go deeper into the problem Approach and the method used it is necessary to first understand a set of key concepts that will be used in the coming chapters, although a general knowledge of mathematics is needed. We will also show where the initial data comes from.

3.1 Knowledge

Knowledge graphs are a "graphical" representation of a set of statements with information, the knowledge base. This information can be anything about everything. The data is stored in triple format. These are statements that consist out of a <Subject>, <Predicate>, <Object>. Where the Subject and Object are vertices in the graph and the Predicate is the edge between the subject and the object. Both the vertices and the edges have different properties, related to each other. Even though the size of the graph can change according to the amount of triples it has, the graph is most of the time connected. But this is not necessarily so.

Definition 1 (Knowledge graph) :

To give more structure to the graphs we need a framework that can help to solidify the relations between vertices. This is done with RDS, RDF(s) and OWL.
RDF [6] stands for **r**esource **d**escription **f**ramework and is a method of storing data in triple format. These statements, consisting out of the <Subject>, <Predicate>, <Object>. can hold different types. The subject can be defined as a URI or a blank node. The predicate represents the relation between the Subject and the Object, and is always defined as a URI. finally the object is the second resource, this can be a URI, blank node or a literal. RDF is designed to exchange information between processes and applications without the intervention of humans. The goal was to build a framework that is designed link resources without having to add in expensive parsers to convert the information to the correct format every time a new process is added which wants to use the information.

Extending to this is RDFS, this the RDF schema, with this schema we can allow to define ontologies within RDF. It can be used to give structure to the RDF RDF(s) and OWL are ontologies from the semantic web community. But they are special in the sense that these languages can be used to reason with. thus making it possible to infer new facts from the knowledge of other triples.

Finally OWL [7], the Web Ontology Language, designed to represent the language tha can describe things and set of things well with regards to relations. But not only that, in OWL it is also possible to reason with that knowledge and make implicit knowledge explicit, for example reason that a subClassOf(Car, Audi) and subClassOf(Audi, A1 Sportsback). This makes it possible to reason that subClassOf(Car, A1 Sportsback) is also a type of relation that exists. The second type of reasoning that is possible now is inconsistency reasoning. A main part of this paper.

Inconsistencies are mistakes in the knowledge graph. It is possible to have data that contradicts other data. An example is a ¿¿¿¡¡¡ . In this example the error is in that two different parts of the data contradict each other. While this can be true in real life data and happen her the real world. Knowledge bases have a hard time understanding inconsistencies and the moment a inconsistency in the data is found it is no longer possible to reason over the data as this means that we can no longer assume that any part of the data is "correct", while it may perfectly be that case.

Definition 2 (Inconsistency) :

Finally we need a language that helps us extract information or patterns from the knowledge graph. This we do with SPARQL. SPARQL is a query language, The technique behind SPARQL is to query and find patterns in the graph and tries to match these patterns to the pattern in query. If a hit is found it can be seen as a hit to the match the complete graph pattern of the query. The advantage of SPARQL to others for finding these patterns is that SPARQL is optimized fro the linked data community. [9]

3.2 Reasoner

What is a reasoner?

What types of reasoners are there?

Why do we need to use a reasoner?

3.3 Data

To provide a good overview about natural occurring data on the web the largest possible dataset full with linked data would be the best starting point. It would not only help with finding naturally occurring errors, but would also help solve any scaling issues that could have occurred with by first experimenting on smaller datasets. The data we are talking about is of course the LOD-a-lot.

3.3.1 LOD-a-lot

The LOD-a-lot [3] is one of the largest open source linked knowledge graphs that is readily available. The data is stored in a HDT. Where an HDT is a (Header, Dictionary, Triples) File. In this way the data is structured in a compact manner with a binary serialization format for RDF. Which has the advantage that querying large files is fast due to optimizations done which have been tailored on reading large data set [4].

To explain further what the data holds we need to first understand what linked data exactly is, specifically linked open data. Linked Open data is the notion of data that is published on the web and conforms to the five stars given to the linked open data. [1]
- The data is available on the web.
- URIs are given to identify things in the data.
- The URI are given HTTP, such that these can be located and found (dereferencing).
- The data that is returned when found is used with open standards such as RDF.
- the data is linked, it refers to other data that has the same linked open data qualities.

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

To generate a knowledge graph with inconsistencies, the first step is locating the inconsistencies needed for the generator. To do such a thing we start with a massive amount of data. With the data and a reasoner we start looking for inconsistencies. To make sure that the inconsistencies found are natural occurring inconsistencies we turn the reasoning capacity off, and only use the reasoner to locate inconsistencies in the data. But there is a problem with the reasoners. A reasoner is namely, 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.

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.

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 separately.

4.2 Generalizing the patterns on basis of isomorphic graphs

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.

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 is an adaptation of the VF2 algorithm[2] is implemented with: solving the problem.

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:

Extra ADDITIONS TO THE APPROACH:

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

In this section the experiments are discussed. This section describes the different experiments that have been done and why these experiments are a good way to showcase the algorithm that has been produced.

Possible Experiments for testing: https://openproceedings.org/2018/conf/edbt/paper-61.pdf Tools for Finding Inconsistencies in Real-world Logic-based Systems

6 Results

In this section the results of the produced method are discussed, as also the results of the experiments from the experiments section.

7 Conclusion

The final section discusses the conclusions of the produced method. In this chapter the reasons why this method is useful for others and why this method should be used as a successor to the previous method used. Namely the use of large knowledge graphs. As also what types of inconsistencies are common enough to solve. In the final subsection of this chapter the future work will explain what possible extensions can be added to this work and why this could benefit the community.

¿What conclusions can be found based upon?

7.1 Future Work

¿ What are the next steps for the algorithm? See papers in the future work for inspirations

References

[1] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5:1–22, 2009.

[2] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1367–1372, Oct 2004.

[3] Javier D. Fernández, Wouter Beek, Miguel A. Martínez-Prieto, and Mario Arias. Lod-a-lot - a queryable dump of the lod cloud. In *International Semantic Web Conference*, 2017.

[4] Javier D. Fernndez, Miguel A. Martnez-Prieto, Claudio Gutirrez, Axel Polleres, and Mario Arias. Binary rdf representation for publication and exchange (hdt). *Web Semantics: Science, Services and Agents on the World Wide Web*, 19(0), 2013.

[5] Michael Frber, Frederic Bartscherer, Carsten Menne, and Achim Rettinger. Linked data quality of dbpedia, freebase, opencyc, wikidata, and yago. *Semantic Web*, 9:1–53, 03 2017.

[6] Y. Raimond G. Schreiber. Rdf primer 1.1. `https://www.w3.org/TR/rdf11-primer/`.

[7] P. Hitzler, M. Krtzsch, B. Parsia, P. Patel-Schneider, and S. Rudolph.

[8] Heiko Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic Web*, 8:489–508, 12 2016.

[9] A. Seaborne S. Harris. Sparql 1.1 query language. `https://www.w3.org/TR/sparql11-query/`.

8 Appendices

8.1 Appendix A