MSc Artificial Intelligence
Master Thesis

Designing custom inconsistent knowledge graphs

by

Thomas de Groot

11320303

May 1, 2019

36 EC

September 2018 - March 2019

Supervisor:
Dr A Person

Assessor:
Dr A Person

institute name

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Motivation

Background. Blaise Pascal once said, "Contradiction is not a sign of falsity, nor the lack of contradiction a sign of truth." Large amounts of data has been made available for everyone to use and Multi-billion facts, also called statements or triples, for a linked-data dataset being the standard in stead of the exception. We are sitting on a large amount of open data, but the problem however is that we do not know if all the facts available to use can be considered as the truth. We know that data is inherently dirty and it is known that finding new facts from the data is impossible if the data holds facts that are contradicting each other. While there is active development in solving the problem of contradictions in datasets by ignoring the contradictory facts and still use the rest of the "true" facts to still find new information that is hidden in the dataset. This method however, would still miss facts that could have been found if we could have removed the contradictory facts and used a datasets that is devoid of contradictions. So while a contradiction is no sign of a false dataset, a lack of contradictions would certainly be a step in the right direction to the truth. The only challenge is then to find if it is the correct truth!

Motivation. Finding and removing contradictions from datasets in a simple and systematic way will have a number of benefits. Firstly we could study the types of mistakes that have been made in the datasets, and how these mistakes are made, by automatic algorithms, or by people in the process. This could help us clean problems by the source, the generation of the data, in stead of later in the process, during the cleaning. Secondly we can generalize the contradictions we have found and design blueprints that can be used to describe groups of contradictions, that can be applied to different datasets. Thirdly, we could use the found contradictions to analyze existing datasets. We can for example use the blueprints to find if there are contradictions in other datasets as well, and we could even count the amount to contradictions to analyze the datasets further. Finally, we can use the contradictions now that we understand them, to build benchmarks designed to test tools, that require large datasets with known characteristics.

Method In this work, we developed a method of extracting contradictions from linked datasets. We call these generalised contradictions 'anti-patterns', as these contradictions can be seen as common mistakes made in dataset. Most commonly used in software design, http://wiki.c2.com/?AntiPattern, describes an anti-pattern as a bad solution for a problem. We describe a formal notion of 'anti-patterns' in chapter **??**. For the extractions of the 'anti-patterns' we have designed an extraction pipeline that can extract 'anti-patterns' for an arbitrary, inconsistent linked-data dataset can retrieve the 'almost' complete set 'anti-patterns'. We show this by retrieving such 'anti-patterns' from the LOD-a-lot[2] a dataset containing 28 billion facts. We also use the set of 'anti-patterns' we have found in the LOD-a-lot as input for the two implementations we designed. The two implementations we designed are, analysis for inconsistent knowledge graphs and a sampling method, and these implementations make extensive use of the found 'anti-patterns'.

1.2 Research questions

Research questions. The main research goal of this work is the task if we can discover a effective method of finding contradictions and generalize the contradictions we found into 'anti-patterns'. In the above section we alreadyy described our method informally. To formalise our we wrote down three research questions that form this paper.

- **RQ1**: Can we describe and define a formal definition for general contradiction patterns, which is persistent throughout all knowledge graphs?
- **RQ2**: How can we retrieve the contradictions that have occurred in the natural knowledge graphs and convert the contradictions into generalised patterns?
- **RQ3**: What main characteristics can describe the general contradiction patterns best, and how can we implement these commonalities to give better qualitative and quantitative information about the general contradiction patterns?
- **RQ4**: Are there methods, or can we design methods that could benefit from using general contradiction patterns in their algorithm?
  - Are we able to improve analytics about contradictions for large linked datasets, by giving qualitative and quantitative information about a linked datasets?
  - Can we use the anti-patterns, to build benchmarks. Benchmarks that have been sampled from large, but inconsistent datasets with general contradiction patterns. Would it be possible to keep the sampled characteristics invariant from sampling?

1.3 Contribution

Experiments and implementations. We test our extraction pipeline on two points, firstly if it is possible to retrieve all 'anti-patterns' from a linked dataset. This is done by using a benchmark dataset, the "pizza-ontology". The "pizza-ontology" is a dummy dataset that holds only a small number of facts and a few contradictions, simple enough to check by hand. Secondly we test our extraction pipeline on the LOD-a-lot. One of the largest open datsets, we use the 'anti-patterns' from this dataset as input for the later parts of our experiment. Finally we have selected two implementations to evaluate our method with real-world examples. Firstly, we have designed a system that can analyse linked datasets. The system retrieves 'almost' all the 'anti-patterns' based on contradictions in the LOD-a-lot, and calculates the number of times the 'anti-patterns' occur in the linked datasets. Then, the system return a detailed report of characteristics of linked datasets. Secondly, we show the implementation that samples a linked dataset and then generates sample of the original datasets that can be tweaked with a user-specified sample-size and the number of contradictions per 'anti-patterns'.

Findings. We have tested our pipeline by extracting an 'almost' complete set of 'anti-patterns' from the LOD-a-lot[2], which we made available ¿¿LINK¡¡. We developed a set of characteristics that can be used to describe 'anti-patterns' more consisely. We show that there is a correlation between the size of the 'anti-pattern' and the amount of different 'anti-patterns'. We tested two implementations by processing a set large knowledge graphs, DBpedia[1], and YAGO[6]. We found that the 'anti-patterns' give us a more detailed explanation to about the contradictions in an inconsistent knowledge graph. We show how the characteristics change between the datasets with different contradictions occur in different knowledge graphs. We also show that the distribution of 'anti-patterns' remains equal in all samples knowledge graphs. Secondly, we show that the sampled knowledge graphs match the set characteristics given by the user, and also match the characteristics of the original knowledge graph.

Contributions. The main contributions in this works, with respect to the research questions can be described in threefold.

- Firstly, we give a formal definition of 'anti-patterns' we use to describe common occurring generalised contradictions that we found in the linked datasets. We use these definitions throughout this work to describe the generalized patterns.
- Secondly, we design an extraction pipeline that can extract 'anti-patterns' from any arbitrary knowledge graph, we show that this method works by extracting "almost" all contradictions. We made all the 'anti-patterns' available to use on the web.
- Thirdly we show that 'anti-patterns' can be used to analyse inconsistent knowledge graphs and to systematically sample inconsistent knowledge graph.

1.4 Outline

The second and third chapter explain the work that this paper is based on, as well as give some background to the reader about the definitions that we use in this work. In the method, we explain more in detail how the method of 'anti-pattern' retrieval works. Section 6 showcases the possible applications that make extensive use from the 'anti-pattern' that we retrieved in the method. In section 7 we show the experiments that test our hypotheses, we set in the introduction, and analyse the results. We conclude with a conclusion, with an extension of future work.

2 Background

This chapter we introduce the set of preliminary key concept that are needed to understand the research that is presented in this work. To help the reader we also introduce a set of examples that we will use in this work to help the reader understand the work that we present here.

2.1 Knowlegde graph elements

Knowledge graph. The first term we introduce is the knowledge graph. A knowledge graph can be seen as a part of a knowledge based system. Where the knowledge base stores all the statements, which represent the facts about the known world. The second part of the knowledge based systems are is the reasoner, which will be discussed in a later part of the this chapter. A knowledge graph consist out of a large amount of facts, also known as statements or triples. Which can be seen as instantiated basic triples patterns, statements that do not contain any changeable elements.

Basic Triple Pattern. Basic triple patterns(BTP) are the building blocks of knowledge graphs, with each basic triple pattern consisting out three elements. These three elements are <Subject>, <Predicate>, and <Object>. All three elements hold different types. The Subject can be defined as a (Uniform Resource Identifier) URI, an set of characters that unambiguously identifies an abstract or physical resource. Or the subject can defined as a blank node or it can be interpreted as a variable. The Predicate represents the relation between the Subject and the Object and is defined as a URI or if unknown as a variable. Finally, the Object is the second resource; this can be a URI, blank node, a literal, or a final variable. A BTP can be instantiated, meaning no element of the triple pattern has a variable, or uninstantiated, an element or multiple elements of the BTP have been replaced with a variable. With an uninstantiated BTP, we can ask a question to the knowledge graph and retrieve instantiated BTPs back. Alternatively, we can check with an instantiated BTP if it exists in the knowledge graph.

Basic Graph Pattern. A basic graph pattern (BGP) is a set of basic triple patterns and forms the basis of query matching. The basic graph pattern can consist out of BTPs that are connected, either through <Subject> and <Subject>, <Subject> and <Object>, or <Object> and <Object>. Alternatively, the BGP consists out of disconnected BTPs. Same as the BTP a BGP can be instantiated, without variables, or uninstantiated, thus parts of the BGP have been replaced with variables. With basic graph patterns, we can ask more informed questions to the knowledge graph, or find if a BGP exists in the knowledge graph.

Justification. The final part of essential information about the knowlegde graph is the justification[5]. A justification is a set of axioms that acts as an explanation for an entailment. Formally, a justification is a minimal subset of the original axioms which are sufficient to prove the entailed formula. In this paper we interpret a justification as an explanation of contradiction. Given that our knowledge graphs are sets of triples, our justification are instantiated BPGs and are always minimal set of axioms for a single contradiction.

2.2 Example introduction

Now that the basic notions of a knowledge graph are explained we will show a knowledge graph work in two examples.

2.3 Reasoner

Now that we understand how the data is represented in knowledge graphs we can move on to the next part, reasoning. Reasoning is a method to find new information about your dataset. Reasoning comes in a multitude of flavours, but the two most common are deductive and inductive reasoning. Before we can start reasoning we first need to define the framework.

The basic framework most reasoners work with consists out of three parts, RDF, RDFS, and OWL.

RDF [3] 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.

RDFS 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.

OWL[4], 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.

Now that we showed the framework we can explain reasoning.

2.4 Data

We introduce four datasets that we use in this work, namely the pizza-ontology, DBpedia, YAGO, and LOD-a-lot.

2.4.1 Pizza-ontology

2.4.2 LOD-a-lot

2.4.3 DBpedia

2.4.4 YAGO

3 Related Work

4 Problem formalization

5 Method

6 Implementations

7 Experiments

8 Conclusion

References

[1] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference*, ISWC'07/ASWC'07, pages 722–735, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] 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.

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

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

[5] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Explaining inconsistencies in owl ontologies. In *SUM*, 2009.

[6] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *Web Semant.*, 6(3):203–217, September 2008.

9 Appendices

9.1 Appendix A