MSc Artificial Intelligence
Master Thesis

Designing custom inconsistent knowledge graphs

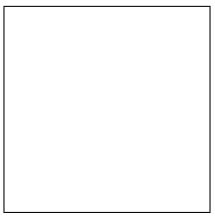
by
Thomas de Groot
11320303

May 1, 2019

36 EC
September 2018 - March 2019

Supervisor:
Dr A Person

Assessor:
Dr A Person



institute name

Acknowledgments

Abstract

Based on formal semantics most of the knowledge graphs on the Web of Data can be put to practical use. Unfortunately, a significant number of those graphs is logically inconsistent. This makes reasoning impossible and the knowledge formally useless. While methods exist to deal with contradictions, no systematic way exists to evaluate algorithms for repair or reasoning with inconsistency on large realistic knowledge graphs. Even worse, there is not even a methodology for analysing inconsistency in graphs in the first place.

While large inconsistent knowledge graphs are available, their size often prohibits their usage for benchmarking. Thus, current research often resorts to synthesised knowledge graphs. In this paper, we present a formal notion of 'anti-patterns', generalised basic graph patterns, describing contradictions in knowledge graphs. We present an extraction pipeline that retrieves explanations for contradictions in the LOD-a-lot collection of knowledge graphs and generalise them into what we call 'anti-patterns'. We enlist an (almost) complete set of 'anti-patterns' found in the LOD-a-lot. Next, we evaluate the set 'anti-patterns' on their intrinsic characteristics. Finally, we introduce two usages of those 'anti-patterns': first, for knowledge graph analysis and, secondly, for generating systematic samples from knowledge graphs. We showcase the two implementations by analysing a set of knowledge graphs and generating configurable inconsistent subontologies of those of variable size.

Contents

1 Introduction

Background. Large open knowledge graphs have increased in size over the last couple of years and are frequently used. Multi-billion statements being the standard instead of the exception. With these enormous knowledge graphs, we try to overcome challenges that have been set out, such as data cleaning, contradiction removal, improving linking between entities, relation linking, or even reasoning over multi-billion statements.

Reasoning with large knowledge graphs is hard due to the size and knowledge graphs are often inconsistent. Even if only one contradiction is stated within the knowledge graph the knowledge graph is inconsistent and any statement is semantically entailed (ex falso quodlibet). While some methods to still be able to reason over inconsistent knowledge graphs, such as paraconsistent reasoning[14], most reasoners can not. If even reasoners try to reason with inconsistent knowledge graphs, then the reasoner is only able to return that the graph is inconsistent. With some reasoners being able to find the justifications if the knowledge graph is small enough.

Motivation. Understanding inconsistency and the types of contradictions that occur in large knowledge graphs can give us a better overview of how mistakes are made. This can also help us create techniques to prevent mistakes being made in the first place. While inconsistency still blocks reasoning, giving qualitative and quantitative information about a knowledge graph contributes to a better understanding of large knowledge graphs. Moreover, once inconsistency in knowledge graphs is better understood we can systematically build benchmarks for tools such as reasoning under inconsistency (based on default or defeasible logics) or debugging tools.

Method. In this paper, we developed the formal notion of 'anti-patterns', describing contradictions that have been generalised to a minimal set of uninstantiated basic triple patterns that we use to describe a set of contradictions in the knowledge graph. We have designed an extraction pipeline that for an arbitrary, inconsistent knowledge graph can retrieve the 'almost' complete set 'anti-patterns'. We show this by retrieving such 'anti-patterns' from the LOD-a-lot[5]. We also use the set as input for the two implementations we designed. Finally, we designed two implementations, analysis for inconsistent knowledge graphs and a sampling method, that make extensive use of the found 'anti-patterns'.

Research questions. The above method informally describes our method. 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 knowledge graphs, by giving qualitative and quantitative information about a knowledge graph?
  - How can we improve sampling, to sample from large inconsistent knowledge graphs with general contradiction patterns, and can we keep we the characteristics invariant from sampling?

Experiments and implementations. We have selected two implementations to evaluate our method with real-world examples. Firstly, we have designed a system that can analyse Knowledge graphs. 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 knowledge graph. Then, the system return a detailed report of characteristics of knowledge graph. Secondly, we show the implementation that samples knowledge graphs and then generates subgraphs of the original knowledge graph with a user-specified sample-size and the number of contradictions.

Findings. We have tested our pipeline by extracting an 'almost' complete set of 'anti-patterns' from the LOD-a-lot[5], 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 or no 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[21]. 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. We give a definition of 'anti-patterns' with respect to inconsistent knowledge graphs. We design an extraction pipeline that can extract 'anti-patterns' from any arbitrary knowledge graph. Finally, we show that 'anti-patterns' can be used to analyse inconsistent knowledge graphs and to systematically sample inconsistent knowledge graph.

Structure. 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 Related Work

Justifications are first introduced by Horridge et al. in their paper explaining inconsistent OWL ontologies [13]. In this paper the writers describe the framework used to retrieve the justifications from inconsistent knowledge graphs as minimal subsets of the graphs preserving the inconsistency. This forms an integral part of our algorithm. In the paper, they also explain why it is often time-consuming to retrieve all justifications for ontologies. In the paper by Töpper et al. [22] they, propose a solution to identify contradictions in DBpedia, with handcrafted 'anti-patterns'. With the extraction of 'anti-patterns' from the Lod-a-lot we have a generalised approach that works on any knowledge graph.
Paulheim [17] showcases the need for a standardised evaluation method. In the survey, they show that researchers sometimes choose different knowledge graph(s) according to their needs, this makes it harder to compare different algorithms. Removing this discrepancy would benefit all of us.
Fárber et al. [8] give an in-depth comparison of several large knowledge graphs, and demonstrates that knowledge graphs hold different metrics. The paper by Färber et al. [8] is expanded upon by Debattista et al. in [4], in which they analysed 130 datasets from the Linked Open Data Cloud using 27 Linked Data quality metrics. Both papers show that each graph has a different underlying structure and in theory, this even can result in different behaviour of algorithms.

2.1 Sampling

In the paper by Jure Leskovec and Christos Faloutsos [15] several sampling techniques are proposed and compared, and with it they show that even naive sampling such as random walks and 'forest fire' show accurate results, even sampling to 15% of the original size. Finally Rietveld et al.[18] shows that sampling for targeted use, in their case SPARQL coverage is possible, this shows that sampling a knowledge base, for targeted cases can generally be relevant for multiple reasons.

3 Background

In this section, we provide a set of basic notions that are used throughout the paper, and have already been defined in previous papers.

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 entities. The Subject can be defined as a URI, 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. A Justification[13] 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.

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 [9] 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 [11], 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. [19]

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 [5] 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 [6].

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. [2]
- 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 Problem formalization

4.1 Formal definition of an 'anti-pattern'

Justifications describe a single contradiction, with each justification being an instantiated BGP. Due to the instantiation, the justification can only describe one contradiction. Anti-patterns are generalisations over justifications as uninstantiated patterns, that match multiple inconsistent justifications in the knowledge graph.

Definition 1. *An 'Anti-pattern' of a knowledge graph G is a minimal set of uninstantiated Basic Triple Patterns that match a justification of an inconsistent subgraph of G.*

The conversion of a justification(instantiated BPG) to an 'Anti pattern' in figure (uninstantiated BPG) is shown in figure 3b.

4.2 Problem Description

At the moment most large knowledge graph have contradictions, but with reasoners only being able to reason on smaller knowledge graphs, we do not know much about the composition of the contradictions in the knowledge graphs. Understanding how the contradictions have formed, or where the contradictions occur, is crucial information when we want to develop better tools that can handle larger knowledge graphs. The second problem is that, while implementing solutions on a large scale is preferable, testing and benchmarking on smaller knowledge graphs that match existing knowledge graphs would be more accurate. As natural knowledge graphs and that have known properties and contradictions instead of synthesised knowledge graphs is preferable.

To solve the issues, we split it into three parts. The first part is to locate the contradictions within the knowledge graph. Only contradictions that are known can be treated as such, as sampling later in the pipeline can only work with contradictions that are known beforehand to sample accordingly. The second part of the problem is the knowledge graph analysis. Analysing the graph gives useful statistics about the knowledge graph, and helps us understand the knowledge graph also with respect to the found inconsistencies in the first part. The final part of the problem is the sampling of the knowledge graph. Is it even possible to sample from the original knowledge base and keep the characteristics we measured in the second part of the problem on the same level as the original large knowledge graph? Moreover, can we make sure that the sampled graph still holds the number of inconsistencies we want the graph to have? We examine this by experimenting with a set of large knowledge graphs and test if we have replicated the original knowledge graph in a smaller variant, as well as a measure if the number of inconsistencies matches the number of inconsistencies we have set.

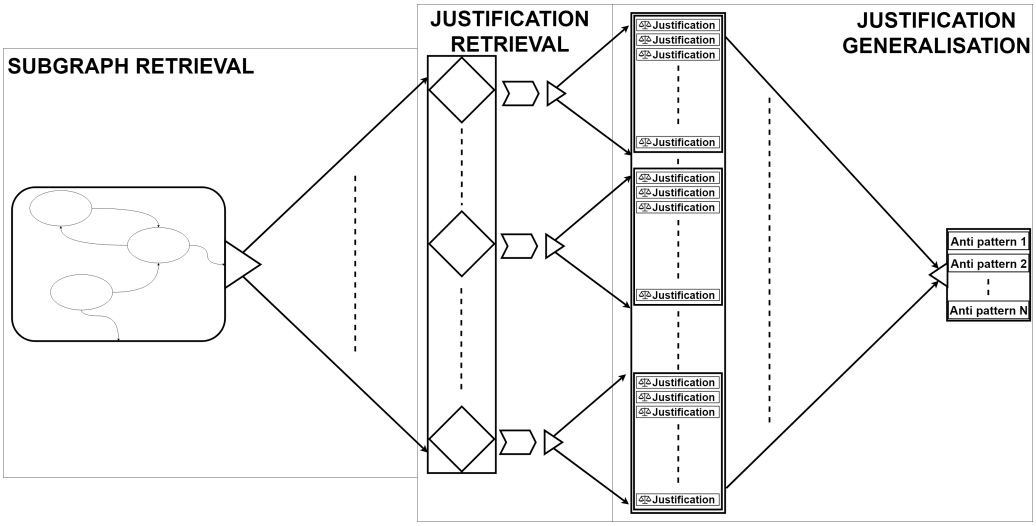
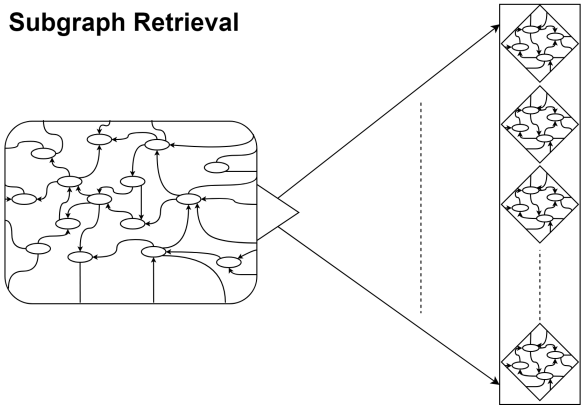
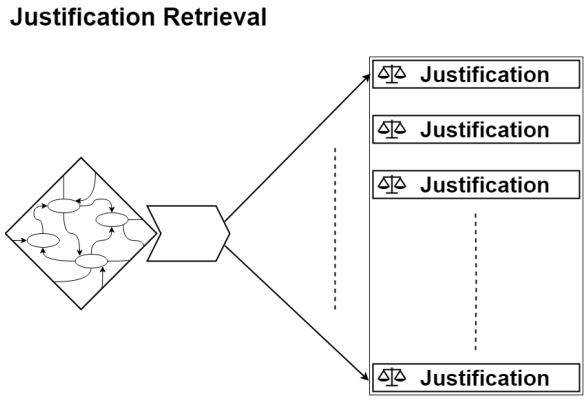


Figure 1: A schematic diagram that shows the pipeline used to extract subgraphs, find justifications and their 'anti-patterns'. Finally we use the information that we retrieved, to analyse and sample the Knowledge graph.



(a) A diagram that shows the retrieval and splitting of subgraphs from the knowledge graph.



(b) A diagram that shows the retrieval of justifications from a single subgraph.

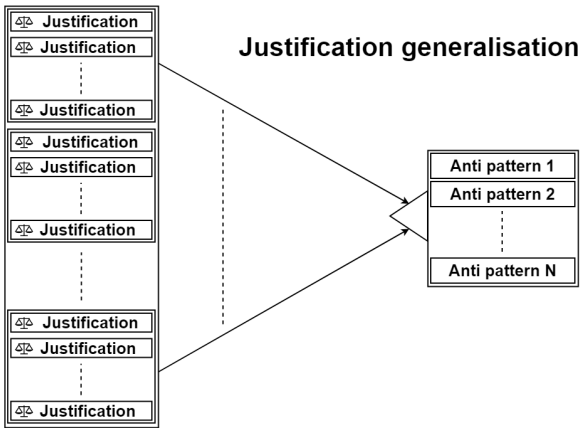
Figure 2: zoomed in diagrams of the first part of the pipeline

5 Method

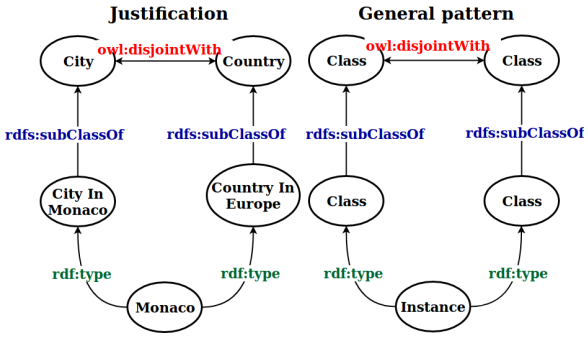
Our extraction method consists of three aspects: Firstly, we retrieve smaller subgraphs from the knowledge graph from which we want the contradictions. Secondly, from each subgraph, we check if the graph is inconsistent and retrieve the justifications. Finally, with the justifications, we create the 'anti-patterns'. The entire pipeline is shown in figure 1.
Our pipeline is designed to find 'almost' all the 'anti-patterns' in any knowledge graph. We implemented techniques to find these smaller inconsistent subgraphs with OWLAPI[12] and Openllet[10] which based on work of Pellet[20].

5.1 Subgraph retrieval

Due to the large size of most knowledge graphs, running a justification retrieval algorithm over the complete knowledge graph, to retrieve all contradictions would be impractical. To speed up this process, we decided to split the knowledge graph into smaller chunks to reduce the time the justification retrieval algorithm needs to find all justifications in the smaller subgraphs. Figure 2a shows the process of splitting the knowledge graph into smaller subgraphs. Each subgraph is generated by extending the root node. The root node is retrieved by taking a triple from the complete graph and taking the node that is in the subject position as the starting point. The graph is expanded by finding all the triples that have the root node as the subject, and we add these triples to the subgraph. Next, all the nodes that were in the object position are now used as expansion nodes, so now for each object, we now find all the triples that match where the object is put in the subject position.



(a) A schematic diagram showcasing how the justifications are transformed into Generalized inconsistency patterns.



(b) A diagram that shows the transformation from Justification to a 'anti-pattern'.

Figure 3: zoomed in the diagram of the third part of the pipeline, as well as a closer look into justifications and general graphs.

We keep expanding the graph until it can not expand any further or the maximum amount of triples of 5000 triples is reached. The value of 5000 triples is chosen because it is large enough to hold almost all justification patterns but small enough that it does not take long to retrieve all justifications.
While this method to sample subgraphs from the knowledge graph does not guarantee completeness in terms of finding all the contradictions that can occur, but we show that this method finds the 'almost' all occurring contradictions, with the help of redundancy, without occurring too many time-consuming calculations.

5.2 Justification retrieval

With the knowledge graph split into smaller chunks, we can now move on the next step in the extraction pipeline, as shown in figure 2b. With the newly formed subgraphs, we start with the check if the graph is consistent or inconsistent. If the graph is consistent, we can skip this graph, as the amount of contradictions is zero.
If the graph is inconsistent, then we find the reason or reasons why, a graph can be inconsistent due to a single contradiction, or it can have multiple contradictions. To find all the contradictions we use the justifications algorithm in the Openllet reasoner. The justifications algorithm walks through the graph and finds the minimal justification for each contradiction. The algorithm continues to search for justifications until no more justification can be found in the graph. This is done for each subgraph, and all the justifications are then pushed through the extraction pipeline to the next stage.

5.3 Justification generalization to 'anti-patterns'

While all justifications are different as each justification is a set of instantiated BTP, the underlying uninstantiated BGP does not have to be. The underlying BGP forms the basis of the 'anti-patterns'. The 'anti-patterns' describe a set of contradictions, that has been found in the inconsistent knowledge graph. The 'anti-patterns' can be used to locate inconsistencies in other knowledge graphs as well. In Figure 3a we show the last part of the pipeline, the conversion of all justifications to a set of 'anti-patterns'.

To get the 'anti-pattern' from the justification, we first generalise the justification by removing the instantiated subject and object on the nodes as shown in figure 3b. This gives the possibility to generalise the justification purely on its structure instead of its instantiated subject and object. While the information in the nodes is not essential, the information on the edges is. Graphs with different edges can be seen as different contradictions.
If we, for example, change the 'owl:disjointWith' into 'rdfs:subclassOf' and transform one of the 'rdfs:subclassOf' into an 'owl:disjointWith' We have a different inconsistency pattern. For this reason, we need to store the edges in the 'anti-pattern'.
With the justifications now devoid of information on the nodes we now group the justifications per 'anti-pattern'. This means that each justification that has the same underlying 'anti-pattern' is isomorphic, even with respect to the edges in the pattern.
To find these isomorphisms we have implemented a version of the VF2 algorithm[3], with the addition to the algorithm that we also match the edges of the two justifications. Checking if two graphs are not isomorphic is

NP-intermediate. Therefore we added in additional heuristics that need to match first before we apply the VF2 algorithm. Firstly we check if a graph has the same number of vertices, the same number of edges, the same amount of degrees, and in our case it also needs to have the same amount of edges based on the edge types we have. If all these matches then we apply the VF2 algorithm to the two 'anti-patterns'. If the algorithm matches a justification to the found 'anti-patterns', it adds this particular justification to this 'anti-patterns', but if no pattern can be matched to the justification, a new 'anti-patterns' is formed from this justification. This algorithm continues until all patterns have been matched to their correct 'anti-pattern' group.

6 Implementations

As shown in figure 4 we show the two implementations that make use of 'anti-patterns' that we can findwith the previous section. The first implementation is the use case of analysing the knowledge graph in its entirety, as well as looking at how the inconsistencies occur within the larger knowledge graph. We use the analytics of the knowledge graph later in the sampling implementation. Secondly, we showcase that the sampling technique, with respect to the found 'anti-patterns' in the graph that we used, produces similar albeit smaller knowledge graphs.

6.1 Analysing inconsistent knowledge graphs

In this paper, we show the implementation of analysing knowledge graphs on a range of characteristics. Our analysis of the knowledge graphs is split into two aspects. The first part is the retrieval of general statistics about the knowledge base:

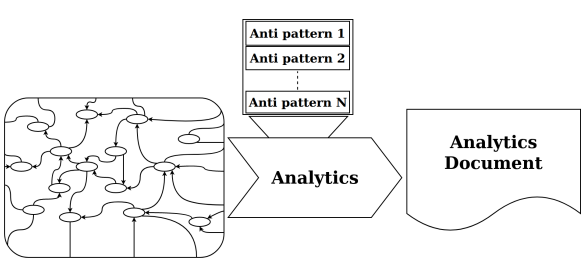
- The number of triples of the knowledge graph.
- The Expressivity of the logic language that is used in KB.
- The number of distinct namespaces in the knowledge graph.
- The number of distinct predicates used in the knowledge graph.
- The distribution of indegree over all the nodes.
- The distribution of outdegree over all the nodes.
- The distribution of the Clustering Coefficient over all the nodes in the graph.

The second part of the analysis is specifically aimed at inconsistency statistics. We locate all contradictions that match the 'anti-patterns' that have been found by retrieving 'almost' all 'anti-patterns' from the LOD-a-lot. With the 'anti-patterns' we can retrieve the needed statistics that characterise the knowledge graph. The metrics that we use in our analysis are:

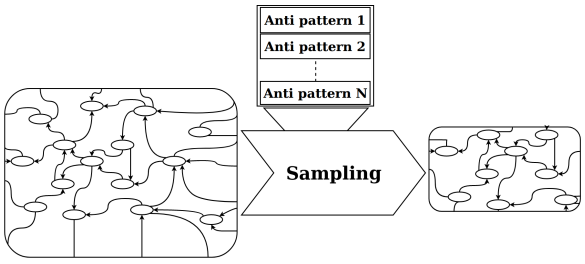
- Amount of 'anti-patterns' in the knowledge graph.
- Largest 'anti-patterns' by a number of basic triple patterns found in the knowledge graph.
- Distribution of the occurrences of contradictions found in the knowledge graph.
- Distribution of the sizes of 'anti-patterns' found in the knowledge graph.

6.2 Sampling inconsistent graphs

The second implementation that makes use of the 'anti-patterns' is the implementation of sampling a knowledge graph. In this paper, the goal of sampling is to sample a knowledge graph into a smaller partition with two constraints. the first constraint is that, we make sure that the knowledge graph stays connected. Secondly, we give the user the possibility to choose the 'anti-patterns' and the minimal amount of contradictions that can be in the sample. To achieve the goal of sampling while keeping the constraints in mind means that we can not use any techniques that start from a node,, or a set of nodes, such as random walks, or forest fire sampling. Because we then cannot guarantee connectedness or the minimal amount of contradictions.
Our algorithm instead uses random constrained deletion; this algorithm randomly deletes triples that can be safely deleted. Triples that are connected to contradictions, or hold the only connection between subgraphs cannot be deleted. The first set of triples that can not be deleted are the triples that describe the contradictions. To make sure that contradictions are not deleted, we start by building an HDT[7][16] that stores all the triples



(a) This diagram shows one of the usecases for the 'anti-patterns', namely Analytics.



(b) This diagram shows the second usecase for the 'anti-patterns', Sampling.

Figure 4: zoomed in diagram of the final parts of the pipelines.

that make up these contradictions. We use a SPARQL query to find the results of each of the 'anti-patterns', and retrieve the number of contradictions the user wanted as BTPs. The triples are then combined and converted to an HDT. The algorithm can now check if a triple can be safely deleted. By checking if the triple is not present in the HDT.

The second set of triples that can not be deleted are the triples that break the to sample graph into smaller subgraphs. To negate this problem the algorithm uses local connectedness. A triple may only be safely deleted if there is a second path that connects the object and the subject without having to pass through the severed link. There is one exception to the rule when either the subject or the object does not have any other links, the triple is then is also allowed to be deleted. With this, we can guarantee that the graph does not split into smaller subgraphs.

The sampling by random deletion now continues to delete triples until the sample reaches the size given by the user, or when it is no longer able to delete triples. This can happen either because the only triples that can be deleted split the graph into subgraphs, or the only triples that can be safely deleted belong to the non-deletable contradictions.

	Inconsistent?	Found inconsistencies
250 triple subgraphs	Yes	9
500 triple subgraphs	Yes	9
1,000 triple subgraphs	Yes	9
5,000 triple subgraphs	Yes	9
Pellet	Yes	6
HermiT	Yes	6
Pellet	Yes	6

Table 1: table showing the reasoners to test the pizza ontology.

	Size	Expressivity	Namespaces	Distinct predicates	Amount of 'Anti-patterns'	Largest Inconsistency
DBpedia	1,040,358,853	SHOIN	20	18	13	19
YAGO	158,991,568	SHOIN	11	5	135	19
pizza	1,946	SHOIN	29	6	2	6

Table 2: table showing several statistics about graphs.

7 Experiments

Datasets. We have implemented the pipeline to retrieve the 'anti-patterns' from the LOD-a-lot. We chose the LOD-a-lot, for its size, with 28,362,198,927 triples and the 650,000 different datasets. the 'anti-patterns we found form the basis for the analysis and sampling of the test cases, YAGO, Freebase, and DBpedia.
We chose these three knowledge graphs, because all three have numerous different characteristics as shown in Färber[8]. With YAGO having over 500.000 classes, and DBpedia having only 736 classes. The number of relations between both YAGO and DBpedia is different, with only 106 relations for YAGO and more than 55,000 relations for DBpedia. With Freebase having both a large number of relations and classes.

Experimental setup. The implementation was written in Java and is the program has been made available here: ¿¿LINK¡¡.

7.1 Anti-pattern Analysis

7.2 Experiment 1: RQ1: Can we design a pipeline that can retrieve a (sub-)set inconsistencies and translate the inconsistencies to 'anti-patterns'?

Experiment description. The purpose of this experiment is to show that we can indeed extract inconsistencies from an arbitrary knowledge graph, with the pipeline. To show that the pipeline works we have tested the algorithm on two extreme cases the first case being the pizza ontology and the second case the LOD-a-lot. The pizza knowledge graph is great for measuring the completeness of the inconsistencies found and shows it is measurable if the reasoner can find the different inconsistencies. Which can be easily checked by hand.
Because we can not prove that our pipeline guarantees the completeness, we test if the pipeline still retrieves all 'anti-patterns' even when the size of the subgraph is smaller than the size of the knowledge graph.
analysis. We compare the results of our method for retrieving the 'anti-patterns' from the pizza ontology with the inconsistencies found by the reasoners in protégé. The results are summarized in Table 2. Results of our method show that even with

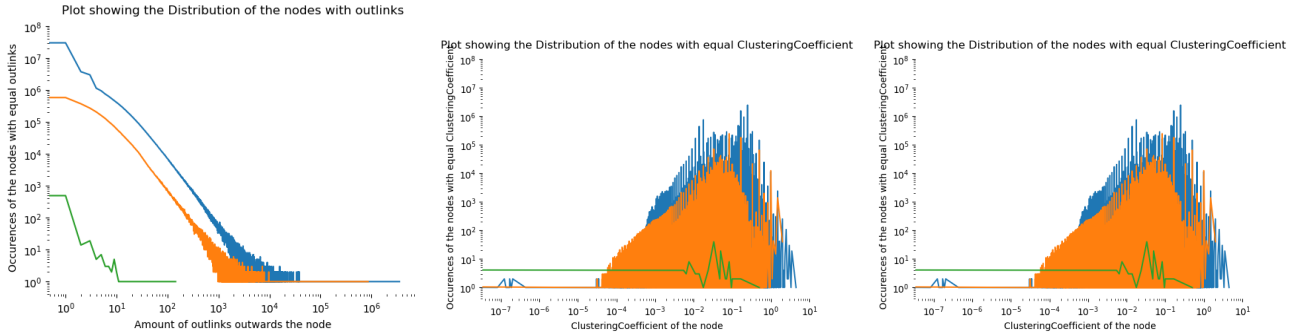
7.3 Experiment 2: RQ3: Implementation 1, Analysing inconsistent graphs

Experiment description. We retrieved the statistics of YAGO, DBpedia knowledge graphs as explained in section 6.

Analysis. Table 2 shows the analytics about the knowledge graph. As noticed the results show several distinctions between the three different knowledge graphs. Even though their expressiveness and the size do roughly match their number of namespaces and distinct predicates differ between the three test cases.

	Size	Expressivity	Namespaces	Distinct predicates	Amount of 'Anti-patterns'	Largest Inconsistency
DBpedia	1,040,358,853	SHOIN	20	18	13	19
Yago	158,991,568	SHOIN	11	5	135	19
pizza	1,946	SHOIN	29	6	2	6

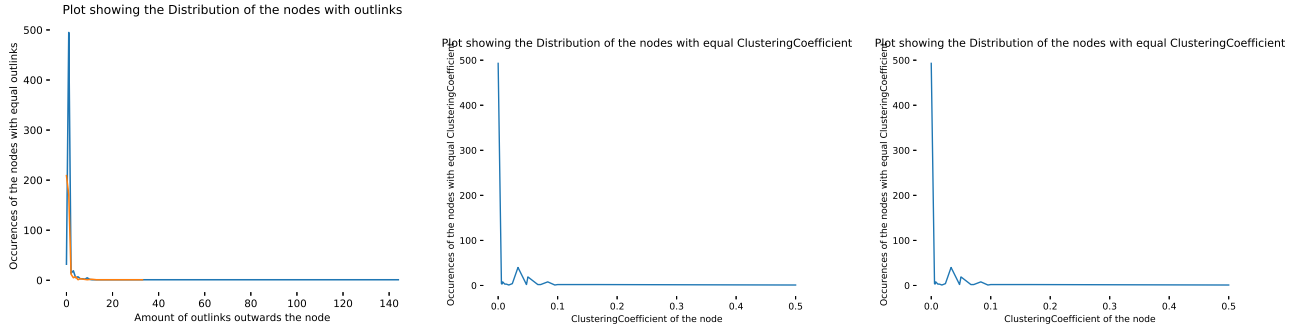
Table 3: table showing several statistics about graphs.



(a) The distribution of indegree over all nodes.

(b) The distribution of outdegree over all nodes.

(c) The distribution of the Clustering Coefficient over all nodes.



(d) The distribution of indegree over all nodes of the sample.

(e) The distribution of outdegree over all nodes of the sample.

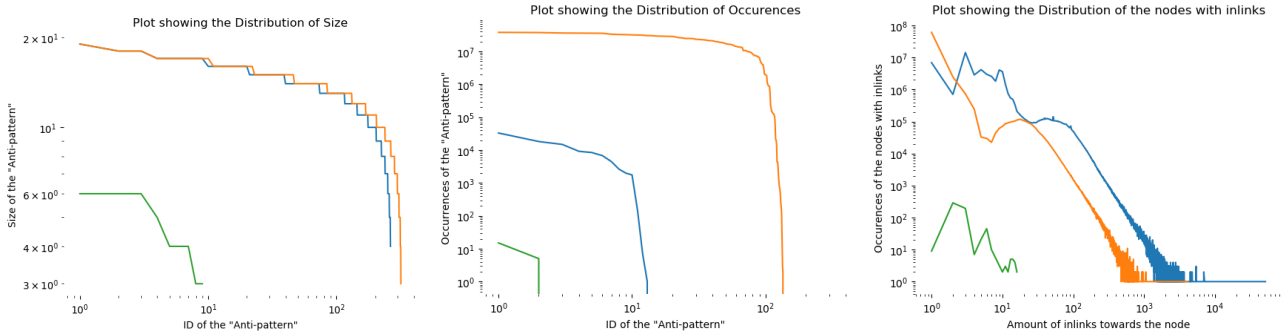
(f) The distribution of the Clustering Coefficient over all nodes of the sample.

Figure 5: Figures showing several statistics about graphs.

7.4 Experiment 3: RQ3: Use case 2, Sampling inconsistent graphs

Experiment description. To test the sampling a sample size of 20% is taken, as the paper by Jure Leskovec and Christos Faloutsos [15] shows that samples up to 15% still hold the characteristics of the original graph well, even with simpler sampling methods.

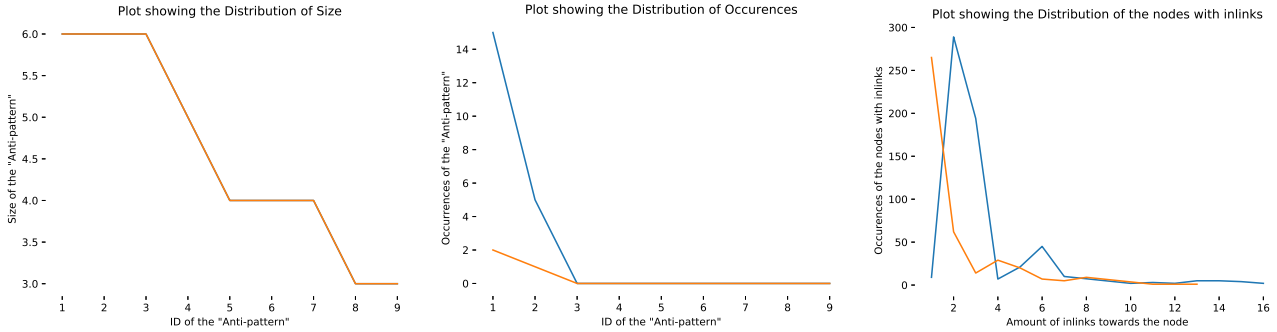
Analysis. Table 2 shows the of the analytics about the knowledge graph before the sampling. As noticed the results, shows several distinctions between the three different knowledge graphs. Even though their expressiveness and the size do roughly match their number of namespaces and distinct predicates differ between the three test cases.
Table 3 shows the analytics of the knowledge graphs after the sampling has been applied. Even though the sample size has been reduced to one-fifth of the original size. The statistics of the sampled knowledge graph still match the original knowledge graph, within the same ballpark scores. Figures 5a, 5b, 5c, show the distribution of statistics of the knowledge graph. Figures 5d, 5e, 5f show the distribution of the statistics sampled knowledge graph. As noted, the distribution of the statistics match the original knowledge graph.



(a) Plot showing the Distribution of size.

(b) Plot showing the Distribution of occurrences.

(c) Plot showing the Distribution of occurrences.



(d) Plot showing the Distribution of size of the Sample.

(e) Plot showing the Distribution of occurrences of Size of the Sample.

(f) Plot showing the Distribution of occurrences of Size of the Sample.

Figure 6: Figures showing several statistics about the Anti patterns.

8 Conclusion

In this paper, we have created a formal notion for 'anti-patterns', patterns describing an error in a knowledge graph that makes the knowledge graph inconsistent. We have constructed a pipeline that can extract the 'anti-patterns' from any knowledge graph. We then tested the extraction pipeline by extracting 'almost' all 'anti-patterns' from the LOD-a-lot.

We have shown two implementations that make extensive use of the 'anti-patterns', knowledge graph analysis and knowledge graph sampling with respect to 'anti-patterns'. With knowledge graph analysis, we can now give qualitative and quantitative information about a knowledge graph with respect to its inconsistency. We observed that 'anti-patterns' follow the same distribution in each of the large knowledge graphs. the Analysis of the 'anti-patterns' also showed that most 'anti-patterns' consist out of 'rdfs:subclassOf' and 'owl:equivalentClass'. 'rdfs:range' and 'rdfs:domain' do not occur in the 'anti-patterns'. Finally, we showed that knowledge graph sampling with respect to 'anti-pattern' is possible. We demonstrate this by sampling knowledge graphs by random deletion. We show that sampled knowledge graphs still have the same characteristics with original knowledge graphs.

Future Work. We want to evaluate why 'rdfs:range' and 'rdfs:domain' do not occur in the 'anti-patterns' and we would improve the generalisation of 'anti-patterns', by creating more general types for 'anti-patterns'.

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

[3] 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.

[4] Jeremy Debattista, Christoph Lange, Sören Auer, and Dominic Cortis. Evaluating the quality of the lod cloud: An empirical investigation. *Semantic Web*, 9:859–901, 09 2018.

[5] 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.

[6] 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.

[7] 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*, 19:2241, 2013.

[8] 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.

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

[10] Galigater. Openllet.

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

[12] Matthew Horridge and Sean Bechhofer. The owl api: A java api for owl ontologies. *Semant. web*, 2(1):11–21, January 2011.

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

[14] Tobias Kaminski, Matthias Knorr, and João Leite. Efficient paraconsistent reasoning with ontologies and rules. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 3098–3105. AAAI Press, 2015.

[15] Jure Leskovec and Christos Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 631–636, New York, NY, USA, 2006. ACM.

[16] Miguel A. Martnez-Prieto, Mario Arias, and Javier D. Fernndez. Exchange and consumption of huge rdf data. In *The Semantic Web: Research and Applications*, pages 437–452. Springer, 2012.

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

[18] Laurens "Rietveld, Rinke Hoekstra, Stefan Schlobach, editor="Mika-Peter Guéret, Christophe", Tania Tudorache, Abraham Bernstein, Chris Welty, Craig Knoblock, Denny Vrandečić, Paul Groth, Natasha Noy, Krzysztof Janowicz, and Carole" Goble. "structural properties as proxy for semantic relevance in rdf graph sampling". In *"The Semantic Web – ISWC 2014"*, pages "81–96", "Cham", "2014". "Springer International Publishing".

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

[20] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical owl-dl reasoner. *SSRN Electronic Journal*, 01 2007.

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

[22] Gerald Töpper, Magnus Knuth, and Harald Sack. Dbpedia ontology enrichment for inconsistency detection. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 33–40, New York, NY, USA, 2012. ACM.

9 Appendices

9.1 Appendix A