

# Entity Reconciliation In Knowledge Graphs

**Haoran(Rohan) Song**

A report submitted for the course  
COMP4560 Advanced Computing Project  
Supervised by: Sergio Rodríguez Méndez  
Armin Haller  
The Australian National University

November 2020

© Haoran(Rohan) Song 2020

Except where otherwise indicated, this report is my own original work.

Haoran(Rohan) Song  
6 November 2020

---

# Acknowledgments

---

Thanks to Sergio Rodríguez Méndez and Armin Haller for helping me with this project.

---

# Abstract

---

This paper studies the method of Knowledge Graph(KG) Entity Reconciliation by exploring the existing popular knowledge graph. At this stage, there are many popular KGs on the Internet, and with the optimization and popularization of automated entry tools, most of the KGs use automated tools for data entry, which will cause some problems, such as attribute redundancy, the same entity data redundancy and so on. The purpose of this project is to classify and find the attribute set that can uniquely identify entities in certain classes so that these attribute sets can be used in subsequent algorithms for Entity Reconciliation. This project uses Wikidata, Dbpedia and NHMRC as data sources. In addition, SPARQL is used to extract the data from KG, and then python is used to extract and transform the data. This project has found an algorithm that can find the attribute set of each class in the KG that can uniquely identify the most entities, and the corresponding algorithm optimization is performed in time and space. The algorithm can meet the search for basic attributes, and after testing, most of the results are logical. But the disadvantage is that there is currently no way to test the operation of the algorithm under super-large dataset, and the algorithm needs to be started through many external inputs to ensure that the algorithm can be more general. The operation effect of the algorithm under super-large data can be optimized in the future work.

---

# Contents

---

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Problem Statement . . . . .	1
1.2 Motivations . . . . .	1
1.3 Project Scope . . . . .	1
1.4 Report Outline . . . . .	1
<b>2 Background and Related Work</b>	<b>2</b>
2.1 Background . . . . .	2
2.2 Related work . . . . .	2
2.3 Summary . . . . .	4
<b>3 Design and Implementation</b>	<b>5</b>
3.1 Language and Tools . . . . .	5
3.2 Basic concepts . . . . .	5
3.3 Design . . . . .	7
3.4 Optimization . . . . .	11
3.5 Implementation . . . . .	12
3.6 Summary . . . . .	14
<b>4 Experimental Methodology</b>	<b>15</b>
4.1 Experiment Purpose . . . . .	15
4.2 Experiment content . . . . .	15
<b>5 Results</b>	<b>16</b>
5.1 NHMRC . . . . .	16
5.2 Wikidata and Dbpedia . . . . .	16
5.3 Summary . . . . .	19
<b>6 Conclusion</b>	<b>20</b>
6.1 Future Work . . . . .	20
<b>Bibliography</b>	<b>21</b>

<b>Appendix</b>	<b>22</b>
.1 Project description . . . . .	22
.2 Study contract . . . . .	22
.3 Description of software . . . . .	26
.4 README file . . . . .	26

---

# List of Figures

---

2.1	Similarity calculation formula . . . . .	3
2.2	Pipeline for HolisticEM framework . . . . .	3
2.3	Similarity calculation formula . . . . .	3
3.1	SPO triplet . . . . .	6
3.2	Basic concepts . . . . .	6
3.3	Inputs and output of the algorithm . . . . .	7
3.4	Flow chart . . . . .	8
3.5	Flow chart . . . . .	9
3.6	BM25 . . . . .	10
3.7	BM25 flow chart . . . . .	10
3.8	Combination() . . . . .	11
3.9	Main . . . . .	11
3.10	Config file . . . . .	12
5.1	The results in wikidata and dbpedia . . . . .	17

---

# List of Tables

---

3.1	Table of language and tools . . . . .	5
5.1	Table of test results on NHMRC . . . . .	16
5.2	Table of test results on wikidata and dbpedia . . . . .	18
1	Table of description . . . . .	26



# Introduction

---

## 1.1 Problem Statement

At this stage, KGs are published in public form and are widely used in information retrieval, recommendation system and other fields. As an important part of semantic data, KG usually contains a large number of overlapping RDF triples. This has a lot to do with the different standards of multi-party cooperation of knowledge graph and the emergence of automatic input systems. Therefore, if multiple interrelated KGs are used at the same time, the same entities in different KGs need to be processed , which is Entity Reconciliation.

## 1.2 Motivations

Since different knowledge graphs belong to the entity's attributes, expression methods, semantics, etc. are not the same, and this project wants to reconciled entities in different KGs, so it needs to find the "same" substitution between them.

## 1.3 Project Scope

The similarity between the different knowledge graphs that this project is looking for is the smallest attribute set that can uniquely identify the most entities of a certain class. This project uses SPARQL and Python to implement a minimum attribute set query algorithm that is widely applicable to most KGs.

## 1.4 Report Outline

This paper have Chapter 2, Chapter 3, Chapter 4, Chapter 5, and Chapter 6.

---

# Background and Related Work

---

The purpose of this chapter is to highlight the innovative value of the work in this article by sorting out existing work. It focuses on the methods used in the Entity Reconciliation of the two papers, and introduces their innovations and shortcomings.

Section 2.1 gives background material necessary in order to read this report ,and related work is given in Section 2.2.

## 2.1 Background

Entity Reconciliation is an operational intelligence process through which organizations can unify different and heterogeneous data sources in order to correlate possible matches of non-obvious entities[Ontotext]. In order to fit the actual scenarios of each business, different knowledge graphs will be independently provided for each business party, which is convenient to manage data with the business side. With the deepening of the business, it will soon be discovered that a single business knowledge graph is very limited in text semantic understanding tasks due to its small scale. At this time, multiple knowledge graphs need to be merged to open up the knowledge boundary. How to enrich and better abstract the external semantic environment of the new entity and its associated structure in the graph are the focus of research.

## 2.2 Related work

### 2.2.1 Holistic Entity Matching Across Knowledge Graphs[Pershina, 2016]

Pershina [2016]’s research uses graphic similarity to improve entity alignment performance. The author proposes a knowledge base instance alignment algorithm (HolisticEM) based on Personalized PageRank. HolisticEM uses the structured information in the knowledge graph to realize the vectorized representation of triple knowledge and iteratively realize multi-source entity alignment. In addition to the classic Personalized PageRank, the author’s innovation has two points:

(1) Considering the semantic contribution of each word to the entire entity in each entity, the lower the IDF score of a word, the more entities the word shares. Therefore,

the difference between such a word and its entity is smaller.

$$\langle e_1, e_2, \text{sim}(\langle e_1, e_2 \rangle) \rangle = \frac{1}{\|e_1\| \cdot \|e_2\|} \sum_{w \in e_1 \cap e_2} \text{idf}_1(w) \times \text{idf}_2(w)$$

Figure 2.1: Similarity calculation formula

(2) Pairs Graph construction: First, select the seed pair set by calculating the IDF-based entity attribute similarity, and secondly, use the seed pair to connect the entity, and add the necessary new entity pairs and edges to the Pairs Graph to expand The original seed pair. This kind of seed pair screening method can obtain more semantic information of the entity than the n-hop method.

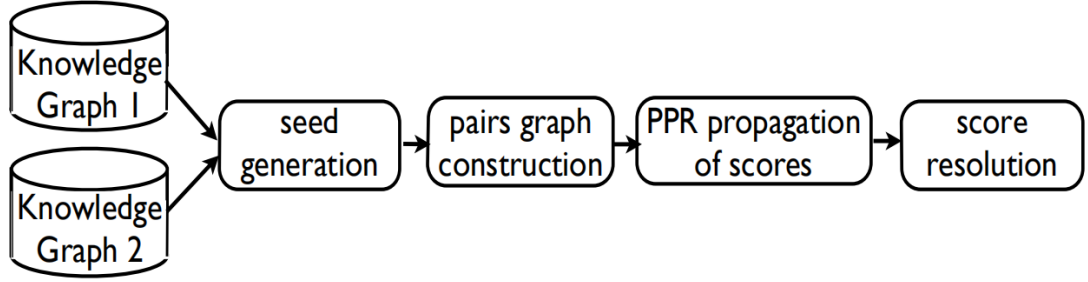


Figure 2.2: Pipeline for HolisticEM framework

### 2.2.2 RDF-AI: an Architecture for RDF Datasets Matching, Fusion and Interlink[Scharffe, 2009]

Scharffe [2009] implements an alignment framework composed of preprocessing, matching, fusion, interconnection and post-processing modules, and proposes an attribute-based entity pair matching algorithm: fuzzy string matching algorithm and word sense similarity algorithm based on sequence alignment. There are two implementations of the word sense similarity algorithm: WordNet-based synonym comparison algorithm and SKOS-based taxonomy similarity algorithm, these two algorithms can also be used in combination[W3C, 2005]. Among them, the WordNet-based synonym comparison algorithm is:

$$\text{Similarity}(SW_i, SW_j) = \frac{1}{\text{No}(SW_i) \times \text{No}(SW_j)} \times \frac{\sum_{w_i \in \{WS_i\} \cap \{WS_j\}} K_s \times \text{IDF}(w_i)^2 + \sum_{w_i \in \{WC_i\} \cap \{WC_j\}} K_c \times \text{IDF}(w_i)^2 + \sum_{w_i \in \{WE_i\} \cap \{WE_j\}} K_e \times \text{IDF}(w_i)^2}{\sqrt{\sum_{i \in Q_U, K \in \{K_s, K_c, K_e\}} K \times \text{IDF}(w_i)^2} \times \sqrt{\sum_{j \in Q_U, K \in \{K_s, K_c, K_e\}} K \times \text{IDF}(w_j)^2}}$$

Figure 2.3: Similarity calculation formula

The above method is used to calculate the attribute matching similarity to obtain all possible aligned attribute pairs in the two images, and the entity similarity is obtained by summing the attribute pair similarities. The one with the highest degree of final entity similarity is considered an entity. But the shortcomings of RDF-AI are also very obvious, that is, there are external maps such as WordNet and SKOS, and the words in the attributes must exist in the external maps.

## **2.3 Summary**

In general, one of the authors of the above two articles used some research to improve entity alignment performance using graphic similarity. The other is the similarity of semantic strings related to entities. However, no matter what kind of entity alignment method is based on, the authors of the two articles have chosen to use attribute pair matching: in the HolisticEM algorithm, the Pairs Graph needs to be constructed through attribute pairs; and in RDF-AI, it needs to be aligned. Attribute pairs are used to calculate entity similarity. However, although the method based on attribute pairs is a good choice, for a huge KG, attributes represent different functions in different categories, that is to say, there are many attributes that have no effect in some categories. Yes, and in some classes, only a few attributes may be needed to represent a class. If the algorithm can filter the attributes before matching the attribute pairs, it will greatly improve the efficiency of the algorithm, which not only saves the cost in time, but also improves the accuracy of the algorithm. Therefore, this article discusses how to find a collection of attributes that uniquely identify most entities of a class.

---

# Design and Implementation

---

The purpose of this chapter is to introduce the creative ideas and implementation process of the algorithm. Including the specific implementation of the algorithm, some optimization methods, time complexity analysis and so on.

Section 3.1 gives introduce to the main languages and tools used by the algorithm ,and the basic concepts of algorithms is given in Section 3.2, Section 3.3 gives introduce to the idea of the algorithm and the four main parts ,and some key optimization methods of the algorithm is given in Section 3.4, Time complexity analysis and config file are in Section 3.5

## 3.1 Language and Tools

The algorithm uses SPARQL to obtain relevant data from KG, and uses python for analysis and processing.

**Table 3.1:** Table of language and tools

Section	Language	Tools
<b>Get dataframe</b>	SPARQL, Python	SPARQLWrapper, pandas
<b>Find combinations</b>	Python	itertools.combinations
<b>Correlation</b>	Python	nlTK, sklearn
<b>Main</b>	Python	configparser

## 3.2 Basic concepts

The knowledge graph is composed of pieces of knowledge, and each piece of knowledge is represented as an SPO triplet(Subject-Predicate-Object)[Sowa, 2014].The semantic network is a way of expressing knowledge through points and edges in a graphical form[Sowa, 2014]. In other words, the semantic network will concatenate these SPO triples.



Figure 3.1: SPO triplet

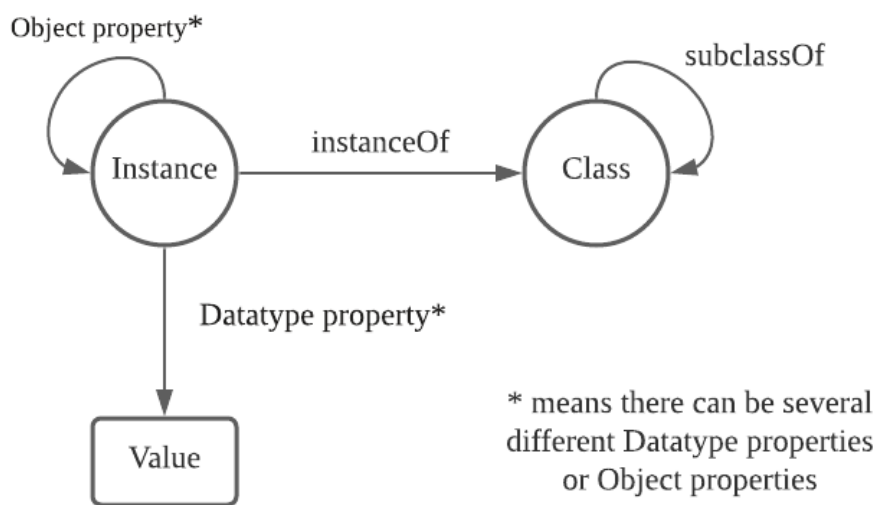


Figure 3.2: Basic concepts

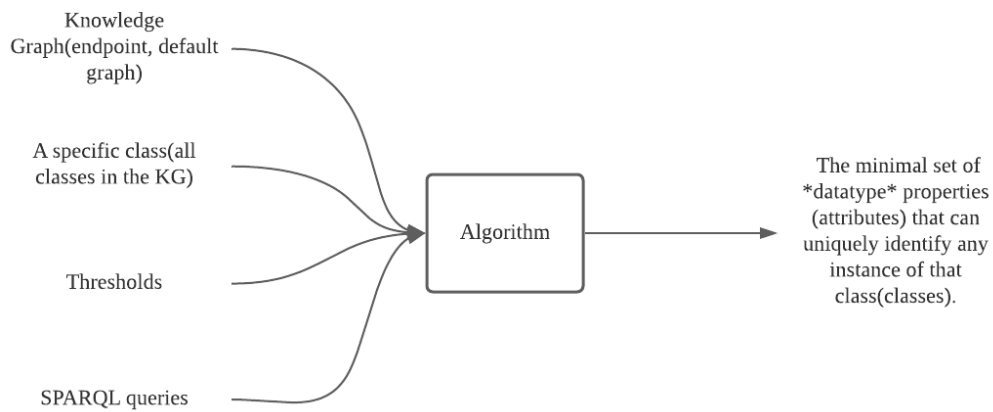
- Instance: something that can exist independently, as the basis of all attributes and the origin of all things[Hegel, 2013], attributes must exist depending on the entity (e.g., Harry Potter 1, Logitech Mouse).
- Class: A class of items has the same description template, forming a class or concept (for example, magazines, movies...).
- Object property: relate individuals to other individuals(e.g., hasage,hasrole, etc.).
- Datatype property:relate individuals to literal data (e.g., strings, numbers, date-times, etc.).
- Value: the value corresponding to the instance's datatype property.

This algorithm focuses on the relationship between class and datatype properties. It finds the smallest set of datatype properties that can uniquely identify most entities of a class.

### 3.3 Design

#### 3.3.1 Input and output

- KG related information: endpoint, default graph.
- Class: the category in the searched KG (\* means to search all classes in the KG).
- SPARQL QUERIES: Use sparql query to obtain relevant information from KG.
- Threshold: limit the result standard, find the completeness of KG and other information.
- Output: The minimal set of \*datatype\* properties (attributes) that can uniquely identify any instance of that class(classes).



**Figure 3.3:** Inputs and output of the algorithm

#### 3.3.2 Flow chart

The purpose of this algorithm is to find the smallest attribute set that can uniquely identify more than 90% of different entities for any class in any KG. The idea of this algorithm is: first use sparql to obtain relevant data, and output them to dataframe, after preprocessing operations such as groupby, sort, and deduplication, find all possible combinations of attributes of this class, and traverse all these combinations to find appropriate attribute set. To determine how many entities can be uniquely identified by an attribute set, which can be converted to how many unique values there are, that is to say we can merge the values corresponding to all attributes of an entity into one line, and finally merge into one string and judge how many different entities this attribute combination can uniquely identify by judging the number of distinct strings. As shown in the following flow chart:

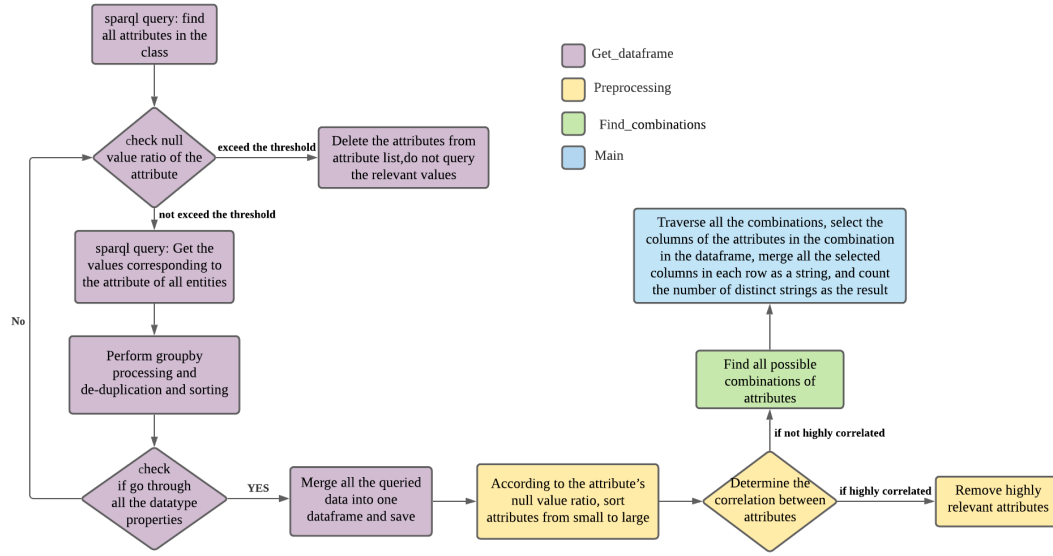


Figure 3.4: Flow chart

Therefore the algorithm is divided into four parts: `get_dataframe`, `preprocessing`, `find_combination` and `main`. The following describes the functions of these modules:

- `Get_dataframe`

This module mainly uses SPARQL statements to obtain data. First, a sparql query is needed to get the union of the attributes of entities of the class, and other SPARQL queries are needed to get all the values of all entities under all attributes. It can be found that there are many entities that do not have a certain attribute or do not have a corresponding value, so the algorithm uses OPTIONAL statements to ensure that even if an entity does not have any value in a certain attribute set, it will not be ignored.

After all the values are obtained, since one attribute of an entity may correspond to multiple values, which will be shown in multiple lines in the result returned by SPARQL. The algorithm finally needs to determine the number of distinct rows, and one entity occupies multiple rows will definitely affect the final judgment. Therefore, all values of an entity need to be merged, which requires groupby operation, and all values in a group need to be merged. In addition, after groupby, each row element of the merged dataframe needs to be de-duplicated, sorted, etc. An example of the pipeline of get dataframe is shown below:



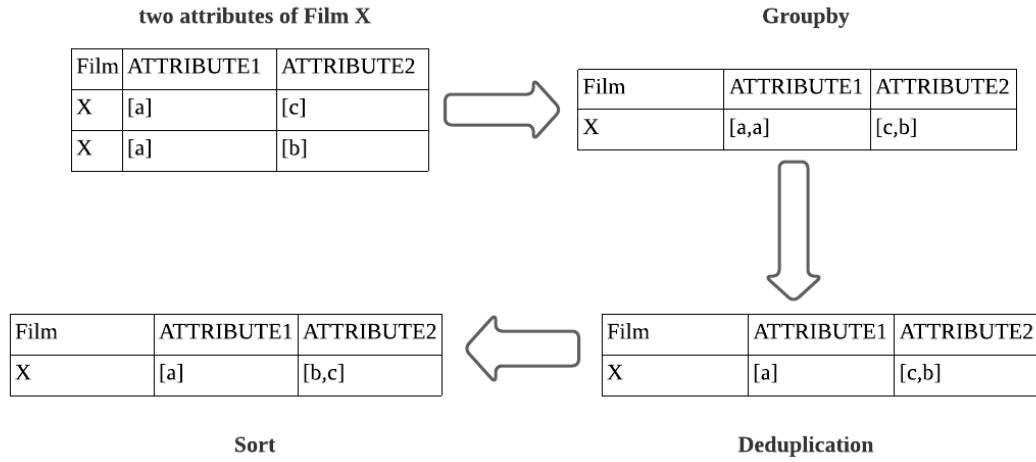


Figure 3.5: Flow chart

#### · Preprocessing

For all the acquired attributes, there are many attributes that are obviously useless, and these useless attributes need to be filtered out through preprocessing, such as attributes with a high proportion of NULL values, highly relevant attributes, and so on.

First, for attributes with high null values, the algorithm filters the attributes by calculating the proportion of NULL values and sets a threshold. If the proportion of NULL values in an attribute exceeds this threshold, the attribute and all corresponding values can be deleted. Because for different KGs, different classes will be different. For example, in one class, we can use 80% as the threshold to get 5 final attributes, while in another class with the same threshold, there are no attributes. Therefore, the threshold is set as an external parameter, and the user can adjust it according to the knowledge map (explained in detail in Section 3.5).

In addition, for highly correlated attributes, this algorithm studies the algorithm for judging the correlation between attributes. First of all, all the values of an attribute are regarded as a document, so that the correlation analysis between the attributes can become the correlation analysis between the documents. Secondly, the method used by the BM25 model to calculate the relevance of all texts and search terms is to compare the keywords of the document with the relevance of the text.

BM25/Okapi [Robertson & Walker 94]

$$b \in [0, 1]$$

$$k_1, k_3 \in [0, +\infty)$$

$$f(q, d) = \sum_{w \in q \cap d} c(w, q) \frac{(k+1)c(w, d)}{c(w, d) + k(1 - b + b \frac{|d|}{\text{avdl}})} \log \frac{M+1}{df(w)}$$

Figure 3.6: BM25

In this module, using BM25 to calculate the keywords of each document (the number of keywords can be set in the config file), and use these keywords as search terms for other documents. Finally, the correlation between the attribute and itself is used as a criterion to judge the correlation between different attributes. That is to say, for example, if the correlation between attribute a and attribute b is very high, then attribute a and its own score very close to the score between attribute a and attribute b. The flow chart is shown in the figure below:

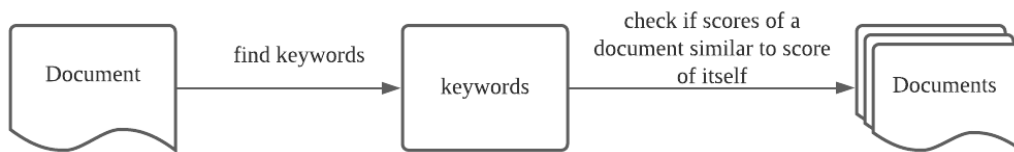


Figure 3.7: BM25 flow chart

Finally, the filtered attribute values are sorted according to the proportion of NULL values from small to large. Because the attributes that can uniquely identify all entities of a category must be attributes with a small proportion of null values. If the attributes with a small proportion of null values are traversed first, it will help the algorithm find the relevant attribute set in advance and reduce the execution time of the algorithm.

#### · Find\_combination

The algorithm uses *itertools.combinations* in Python to calculate all possible combinations of the attribute set, that is, all subsets of the attribute set. The subset obtained by the `Combinations()` method is order-independent, which means that `[a,b]` and `[b,a]` will be considered as a result, which meets our needs.

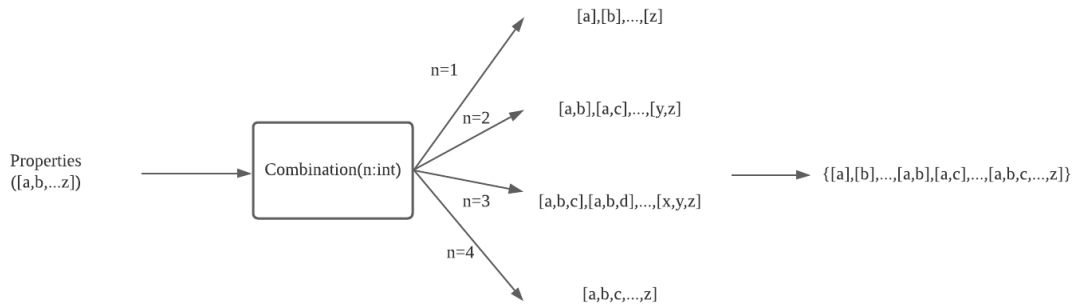


Figure 3.8: Combination()

#### · Main

The main part of the algorithm traverses all the attribute sets found in Find\_combination. And filter out all the values corresponding to the current attribute set in the dataframe obtained from Get\_dataframe. Subsequently, these values are combined into a string, that is, all the attribute values of this entity in the current attribute will appear as a string of one row in the dataframe. In this way, the algorithm judges how many entities this attribute set can uniquely identify by judging the number of non-repeated strings in dataframe.

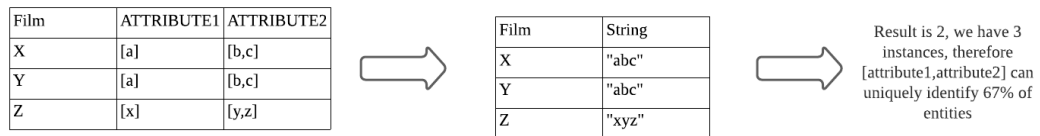


Figure 3.9: Main

## 3.4 Optimization

- When the original algorithm reads data from SPARQL, it uses the optional statement to read all the values of all attributes (including attributes without values, which are displayed as NULL). However, most entities' attributes correspond to more than one values. For example, most films have more than one actors, therefore when querying the value of the actors in a film, SPARQL will return multiple rows of data. That is to say, when the data is output to the dataframe, the form of expression is also multi-row. Therefore, for example, when a class with 200,000 entities is output to dataframe, it will be represented as more than 2 million rows of data. It is not good for the time and space of the algorithm. Therefore, when transferring the output of SPARQL to dataframe, the algorithm traverses by attribute, that is, output column by

column to sub\_dataframe, and then perform groupby, sort, and deduplication operations on each sub\_dataframe. It can ensure that the number of rows of each sub\_dataframe is the number of entities in the class, and finally merge all the sub\_dataframes to obtain the final dataframe.

- In the original algorithm, when preprocessing the data, one step is to delete attributes with high ratio of NULL values. If the algorithm is connected from the top to bottom, it will be found that the original algorithm actually inputs all attributes with high ratio of NULL values and corresponding values into dataframe and deletes these attributes and corresponding values. In other words, it can be found that these two steps are actually redundant. The attributes with high ratio of NULL values and the corresponding values can be deleted when inputting, so that unnecessary time for inputting attributes with high ratio of NULL values and corresponding values can be saved.

## 3.5 Implementation

### 3.4.1 Config file

The Config file is designed to make the algorithm more general. Users can change different variables according to the KG they want to test. The designed config file is shown in the figure below:

```
[main]
kg = nhmrc

[nhmrc]
endpoint_url = http://rsmsrv01.nci.org.au:8890/sparql/
defaultgraph = http://rsmsrv01.nci.org.au/agrif/nhmrc/grants/v0
class_name = http://linked.data.gov.au/dataset/nhmrc/grants#GrantResearcher
delete_ratio = 0.7
num_keywords = 10
class_properties_query = SELECT DISTINCT ?property WHERE { ?s a <%(class_name)s>. ?s ?property ?o. FILTER( !(ISIRI(?o)) )}
j = *
values_query = select ?item ?a { ?item rdf:type <%(class_name)s>. OPTIONAL {?item <%(i)s> ?a.}}
total_query = select (count(*) as ?count) { ?item rdf:type <%(class_name)s>. OPTIONAL {?item <%(i)s> ?a.}}
distinct_query = select (count(*) as ?count) { ?item rdf:type <%(class_name)s>. ?item <%(i)s> ?a }
number_of_instance = select distinct (count(*) as ?count) { ?item rdf:type <%(class_name)s> }
result_ratio = 0.95
corr_para = 1
```

**Figure 3.10:** Config file

The user can change the "kg" variable in the [main] section to tell the algorithm which KG to run. If there is no section about the KG in the config file, the user can add a section with the same name as the variable value of "kg". In this section, it is needed to include some variables below:

- endpoint\_url: sparql endpoint.
- defaultgraph: select the knowledge graph to query.

- 
- `class_name`: Query class IRI, if it is needed to query all classes in KG, fill in .
  - `delete_ratio`: The number of [0,1]. If the proportion of null values in an attribute exceeds this number, the attribute and the corresponding value will not be added to the dataframe.
  - `num_keywords`: How many keywords are extracted for each document when executing the BM25 algorithm.
  - `class_properties_query`: find a SPARQL query for all properties of a class.
  - `i`: In order to connect the variable set by the external attribute, it will be automatically assigned during the algorithm, so there is no need to fill in.
  - `values_query`: a SPARQL query to find all values corresponding to "i".
  - `total_query`: a SPARQL query to find how many entities "i" corresponds to.
  - `distinct_query`: a SPARQL query to find out how many non-null entities the "i" corresponds to.
  - `number_of_instance`: a SPARQL query to find the total number of entities in this class.
  - `result_ratio`: The number of [0,1]. If an attribute set can be found to uniquely identify the number of entities whose proportion is "result\_ratio", the algorithm will stop.
  - `corr_para`: When the score is compared in the BM25 algorithm, the difference between the two will be marked as high correlation.

### 3.4.2 Time complexity

In this section, the time complexity of the important part of the algorithm is analyzed:

- `Get_dataframe`:  $O(mn)$   
Where m: number of properties, n: number of instances. The algorithm needs to traverse each attribute to find the attribute whose proportion of NULL values meets the requirements, and the algorithm needs to traverse all the values of the attribute and input the json format result obtained by SPARQL query into the dataframe.
- `Find_combinations`:  $O(nrC_n^r)$   
Where r: the length of combination we are searching for, n: length of the list. The time complexity of `Combination()` is  $O(rC_n^r)$ , but `Combination()` specifies the length of searching for subsets, such as a subset of length 2, and so on. The algorithm needs to find all subsets, so it needs to traverse all possible subsets with length n.

- Correlation:  $O(n^2)$

Where  $n$ : the length of doc. The algorithm first needs to traverse all the words in a doc to determine keywords, and then needs to traverse all the words in the doc to find whether they are related to keywords.

- Traverse:  $O(cn)$

Where  $c$ : number of combinations,  $n$ : number of instances. The algorithm needs to traverse all the combinations, and in every possible combination, change the string of each row of the dataframe.

It can be seen that among these key algorithms, the worst time complexity of the algorithm is that the time complexity of Correlation reaches  $O(n^2)$ . The time complexity of the Correlation algorithm can still have a lot of room for improvement, and it can be expected to run Correlation will take more time.

### 3.6 Summary

This chapter introduces the implementation process of the algorithm through the flow chart, as well as the use of the config file and the time complexity of the algorithm. The next chapter will conduct algorithm testing and introduce the method of algorithm testing.

# Experimental Methodology

---

## 4.1 Experiment Purpose

In order to explore the effect of the algorithm on different KGs, public KG (wikidata, dbpedia) and custom KG (NHMRC) were selected for testing. When testing in a public KG, it mainly explores the execution time of the algorithm to test whether the algorithm can run under a large KG; for custom KG, because the number of entities is relatively small, and the "primary key" created by each class can be known, therefore it is a good opportunity to test the accuracy of the algorithm.

## 4.2 Experiment content

- Set result\_ratio to 0.9, that is to say, if an attribute set can be found to uniquely identify the number of entities accounting for 90%, the algorithm stops.
- Tested all classes of NHMRC KG and got the results.
- Tested the four representative classes of Wikidata and Dbpedia: film, book, animals, country. Get test results and analyze the execution time of each part of the algorithm.

# Results

## 5.1 NHMRC

- Result

Table 5.1: Table of test results on NHMRC

Class Name	Results	Examples
<b>FieldOfStudy</b>	name	"Modes of convergence"
<b>ConferencePaper</b>	title	"CP-Miner: a tool for finding ..."
<b>Investigator</b>	fullName	"A/Pr Ingrid van der Mei"
<b>GrantApplication</b>	applicationID	"APP1185426"
<b>BookChapter</b>	doi	"10.1007/978-94-007-1333-8 <sub>83</sub> "

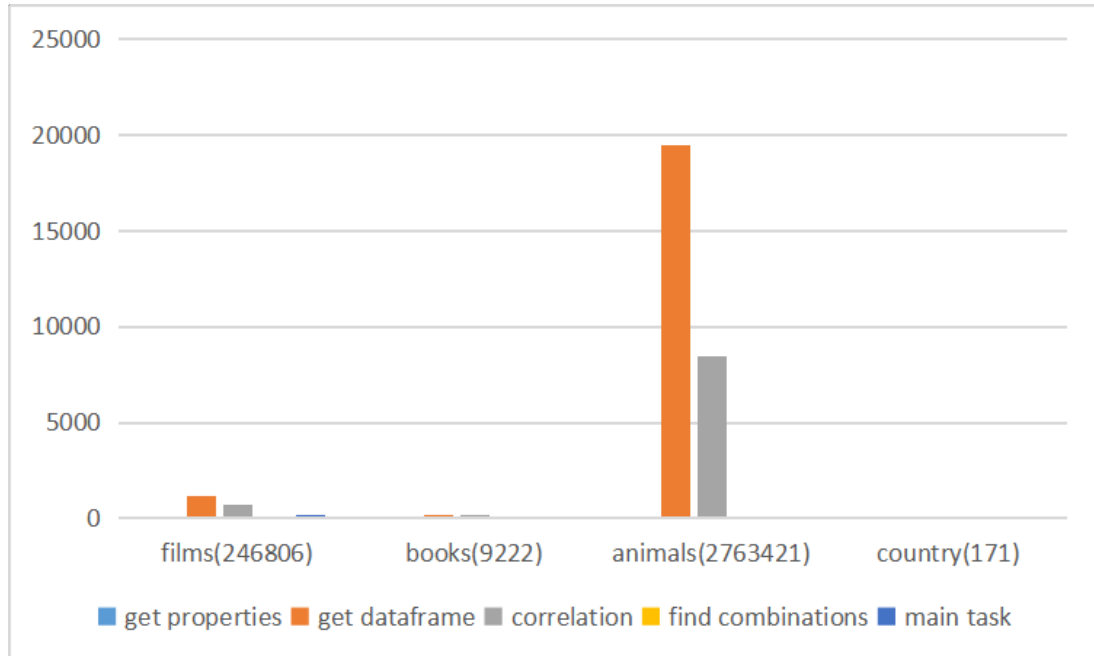
Taking the GrantApplication class as an example, the result of the algorithm is applicationID, which means that the algorithm believes that the attribute applicationID can uniquely identify more than 90% of entities in GrantApplication. Each GrantApplication has a unique ID to identify this entity, so applicationID is a correct result. After the same comparison, most of the results are reasonable.

However, some results seem unreasonable. For example, the result obtained in the ConferencePaper class is citation count. Since the number of citations varies greatly, it seems that a number can represent a paper, but if the meaning it represents is being analyzed, it is found that the number of citations is used as an attribute of a class. This result is obviously unreasonable.

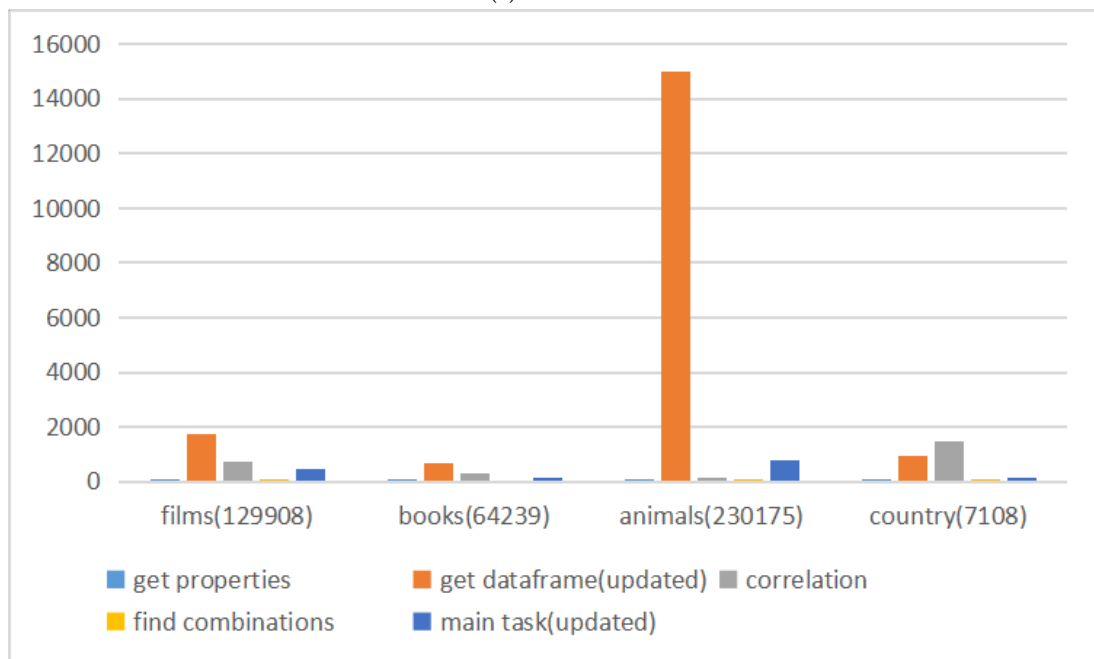
## 5.2 Wikidata and Dbpedia

- Execution time(Unit: Second)





(a) Wikidata



(b) Dbpedia

**Figure 5.1:** The results in wikidata and dbpedia

It can be seen that the main execution time of the algorithm is spent in the two parts of `get_dataframe` and `correlation`, and the time spent by `get_dataframe` far exceeds the execution time of `correlation`. In the previous time complexity analysis, the `correlation` part will spend the most execution time, but the test result is not the case. In addition to the number of attributes contained in the class and the number of class entities will affect the execution time of the algorithm, the main factors affecting the execution time of `get_dataframe` are as follows:

- Network download speed: As the algorithm needs to use sparql query to download data from KG, the network environment will greatly affect the execution speed of the algorithm.
- The stability of the endpoint and the local connection: Most of the categories in the KG are not only the category of the KG, but also many external KGs are connected. During the test, for example, the local connection access to "http://purl.org/" will be very slow, and sometimes even 500 error will appear.
- Result

**Table 5.2:** Table of test results on wikidata and dbpedia

Class Name	Wikidata Results	Depedia Results
<b>Films</b>	director, IMDB ID, publication date	IMDB ID, director, runtime
<b>Books</b>	author,title	isbn, author, numberOfPages, publisher
<b>Animals</b>	taxon name	family, taxonomy, binomialAuthority, genus
<b>Country</b>	highest point, GS1 country code	capital, dissolutionDate

In the table, the test results are compared with the expected results, and it can be found that the difference between the two is still very large. In some classes, the two are completely different, and some classes are only partly the same. The reasons for this result are as follows:

- The NULL value of the attribute will make the result different. If an attribute can uniquely identify more than 90% of entities in entities with non-empty values of this attribute, then this attribute will be expected as the result of the search, but if the attribute value is null in 20% of the entities, then it can not be used only to uniquely identify this class. For example, for the film category, the imdb id is expected as the result, because it seems that every movie will have a different imdb id. But after testing, in wikidata, there are 246806 movies, and only 199747 movies have imdb id, which means that only 80.9% of the movies have imdb id. In this way, only through the imdb id attribute, it is impossible to uniquely identify more than 90% of the entities, so the director and publication date are also added to the test results to help identify the entities.

- 
- Different understanding of classes. Different KGs have different classification standards, which will cause differences. For example, the country class is considered as countries like Australia and America when the result is expected. So the expected result is Geographical location. In Wikidata, the classification standard of country is distinct region in geography[Dbpedia], so the result will be updated with historical changes, that is, it will be closer to the current situation. But in Dbpedia, country refers to A country is a region that is identified as a distinct national entity in political geography[Wikidata], which will include all historical "country", so in the final result A result like dissolutionDate appears.

### 5.3 Summary

This chapter introduces the algorithm test on NHMRC, Wikidata and Dbpedia, analyzes the test results and the execution time of the algorithm. This paper will be summarized in the next chapter.

---

# Conclusion

---

In the work of predecessors, many people use attributes for Entity Reconciliation, but the lack of initial screening of attributes will greatly increase the workload. This article introduces the optimization algorithm in the preparation phase of Entity Reconciliation: finding the attribute set that can uniquely identify most entities of a category. The algorithm is roughly divided into four parts: `get_dataframe` to obtain data, preprocessing to filter attributes, `find_combination` to calculate all possible combinations of attribute sets, and `main` to traverse all attribute set combinations to find the final result. In addition, the algorithm also provides a config file that allows users to modify some variables to adapt to different kg. This article analyzes the test results of the algorithm on NHMRC and the three KGs of Wikidata and Dbpedia. This algorithm is useful in "schemaless" KGs. That is, KGs that don't have a specific schema and, also, in KGs with incomplete and noisy data. This algorithm is a mechanism to uniquely identify the instances. Most of the results of the algorithm are reasonable, but the running speed still needs to be greatly improved.

## 6.1 Future Work

- As mentioned above, although the algorithm can run on most classes, timeout will occur when the amount of data reaches one million, so the time complexity of the algorithm needs to be further optimized to meet more general requirements.
- Determine the value of attributes through Semantic Parsing and Syntactic Parsing. When filtering attributes, it is also needed to consider whether the attribute value is meaningful. For example, the result obtained in the `ConferencePaper` class is citation count. Due to the large difference in the number of citations, it seems that a number can represent a paper, but if the meaning it represents is being analysed, this result is unreasonable. Therefore, in the future, it is hoped that the algorithm can use Semantic Parsing and Syntactic Parsing to analyze the meaning of attribute values.

---

# Bibliography

---

DBPEDIA. About: Country. <http://dbpedia.org/page/Country>. (cited on page 19)

HEGEL, G. W. F., 2013. *Wissenschaft der Logik*. CreateSpace Independent Publishing Platform. ISBN 9781484031834. (cited on page 6)

ONTOTEXT. What is a knowledge graph? <https://www.ontotext.com/knowledgehub/fundamentals/what-is-a-knowledge-graph/>. (cited on page 2)

PERSHINA, M., 2016. Graph-based approaches to resolve entity ambiguity. *A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy Department of Computer Science New York University*, (2016). (cited on page 2)

SCHARFFE, F., 2009. Rdf-ai: an architecture for rdf datasets matching, fusion and interlink. *Semantic Technology Institute*, (2009). (cited on page 3)

SOWA, J. F., 2014. *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann. ISBN 9781483207711. (cited on page 5)

W3C, 2005. Skos core guide. <https://www.w3.org/TR/2005/WD-swbp-skos-core-guide-20051102/>. (cited on page 3)

WIKIDATA. country(q6256). <https://www.wikidata.org/wiki/Q6256>. (cited on page 19)

---

# Appendix

---

## **.1 Project description**

Knowledge Graphs (KGs) proliferating on the Web are well known to be incomplete and inaccurate, as many of the facts in the graph are created using automated tools. One particular problem with these tools is that they often create duplicates for equivalent entities. The same entity is duplicated many times, because it is challenging to decide which properties of a specific entity type are functional (e.g. as an email address or passport number is for a human entity). This project aims at analysing an existing knowledge graph and the entities contained within to determine which properties can be used to reconcile entity types within the graph. The student will be able to use different types of techniques to determine which works best to reconcile entities in an RDF-based graph model.

## **.2 Study contract**

# INDEPENDENT STUDY CONTRACT

## HONOURS

*Note: Enrolment is subject to approval by the projects co-ordinator*

### SECTION A (Students and Supervisors)

UnID: U6688461

SURNAME: Song FIRST NAMES: Haoran

PROJECT SUPERVISOR (*may be external*): Armin Haller

COURSE SUPERVISOR (*a RSCS academic*): Eric McCreath

COURSE CODE, TITLE AND UNIT: COMP4560, Advanced Computing Research Project, 12

SEMESTER ☐ S1 ☒ S2 YEAR: 2020

#### PROJECT TITLE:

Entity reconciliation in knowledge graphs

#### LEARNING OBJECTIVES:

The student will be able to use different types of techniques to determine which works best to reconcile entities in an RDF-based graph model.

#### PROJECT DESCRIPTION:

Knowledge Graphs (KGs) proliferating on the Web are well known to be incomplete and inaccurate, as many of the facts in the graph are created using automated tools. One particular problem with these tools is that they often create duplicates for equivalent entities. The same entity is duplicated many times, because it is challenging to decide which properties of a specific entity type are functional (e.g. as an email address or passport number is for a human entity). This project aims at analysing an existing knowledge graph and the entities contained within to determine which properties can be used to reconcile entity types within the graph. The student will be able to use different types of techniques to determine which works best to reconcile entities in an RDF-based graph model.



**ASSESSMENT** (as per course's project rules web page, with the differences noted below):

Assessed project components:	% of mark	Due date	Evaluated by:
Thesis	_____ (85%)		
Presentation	_____ (10%)		
Critical Feedback	_____ (5%)		

**MEETING DATES (IF KNOWN):**

**STUDENT DECLARATION:**

I agree to fulfil the above defined contract

..... 20-07-20  
Signature Date

**SECTION B (Supervisor):**


I am willing to supervise and support this project. I have checked the student's academic record and believe this student can complete the project. I nominate the following reviewers and have obtained their consent to review the completed thesis (through signature or attached email)

..... 20-07-20  
Signature Date

**Reviewer 1:**

Name: ..... Pouya Omran Signature.....  .....

**Reviewer 2:**

Name: ..... Sergio Rodríguez Méndez Signature.....  .....

\*Nominated reviewers may be subject to change on request by the supervisor.

**REQUIRED DEPARTMENT RESOURCES:**

**SECTION C (Course coordinator approval)**

.....  
Signature Date

**SECTION D (Projects coordinator approval)**





.....  
Signature

.....  
Date

### .3 Description of software

- The algorithm is divided into four py files: main.py, corrleation.py, sparql.py, preprocessing.py. All modules were implemented by the student.
- Algorithm testing only needs to modify the variables in config.fig, there is no separate test code.

**Table 1:** Table of description

Section	Description
<b>Hardware</b>	Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz (4 CPUs), 2.3GHz
<b>dataset</b>	Wikidata, Dbpedia, NHMRC
<b>compiler / version details</b>	Python 3.7.3, SPARQL 1.1

### .4 README file

- Users can modify external attributes through *config.fig*, the specific explanation of *config.fig* is as follows:  
*kg* = tell the algorithm which KG to run  
*endpoint\_url* = sparql endpoint  
*defaultgraph* = select the knowledge graph to query  
*class\_name* = Query class IRI, if it is needed to query all classes in KG, fill in \*.  
*delete\_ratio* = The number of [0,1]. If the proportion of null values in an attribute exceeds this number, the attribute and the corresponding value will not be added to the dataframe.  
*num\_keywords* = How many keywords are extracted for each document when executing the BM25 algorithm.  
*class\_properties\_query* = find a SPARQL query for all properties of a class.  
*i* = In order to connect the variable set by the external attribute, it will be automatically assigned during the algorithm, so there is no need to fill in.  
*values\_query* = a SPARQL query to find all values corresponding to "i".  
*total\_query* = a SPARQL query to find how many entities "i" corresponds to.  
*distinct\_query* = a SPARQL query to find out how many non-null entities the "i" corresponds to.  
*number\_of\_instance* = a SPARQL query to find the total number of entities in this class.  
*result\_ratio* = The number of [0,1]. If an attribute set can be found to uniquely identify the number of entities whose proportion is "result\_ratio", the algorithm will stop.  
*corr\_para* = When the score is compared in the BM25 algorithm, the difference

---

between the two will be marked as high correlation.

In `main.py` the user can start the algorithm and wait for the result to be displayed.

- The project has been tested in the following KGs endpoints:  
<https://query.wikidata.org/>  
<http://dbpedia.org/sparql>  
<http://rsmsrv01.nci.org.au:8890/sparql/>