Bachelor Thesis

Detecting Duplicate Entities for Ontology Reconciliation

Marius Stefan Bethge

05.12.2013

Albert-Ludwigs-University Freiburg im Breisgau
Faculty of Engineering
Department of Computer Science
Chair of Algorithms and Data Structures

Bearbeitungszeitraum
05. 09. 2013 – 05. 12. 2013

Gutachter
Prof. Dr. Hannah Bast

Betreuer
M.Sc. Björn Buchhold

There is nothing more deceptive than an obvious fact.

Sherlock Holmes

Declaration

I hereby declare, that I am the sole author and composer of my Thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work. I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Freiburg, December 5th, 2013

Marius Stefan Bethge

Contents

Acknowledgments

First and foremost I would like to thank my supervisor Prof. Hannah Bast, for making this thesis possible. Her lectures have sparked my ambitions for optimization and efficiency and she introduced me to a now dear but moody friend, C++. A great many thanks go to Björn Buchhold for his guidance over the past three months and his always encouraging support and ideas.

I also would like to thank my parents and my sister for their never ending support, motivation and love.

Last but not least I want to thank my close friends in Freiburg. Particular gratitude goes to Alex and Ai for their hospitality. Their warmhearted, genuine and easy going nature is out of this world.

Zusammenfassung

Die stetig wachsende Nachfrage nach Ontologien, in Verbindung mit dem Vorhaben mancher Ontologien das menschliche Wissen über alle Gebiete hinweg zu vereinigen, führt zur immer größer werdenden Datenmengen. Diese domänenübergreifenden Ontolgien, wie z.B. Freebase, sammeln Daten von diversen Quellen, welches jedoch leicht dazu führen kann, dass mehrere Einträge für ein einzelnes Objekt der reellen Welt stehen können. Wir stellen einen Algorithmus vor, welcher durch Identifizierung und Vereinigung von duplikaten Entitäten die Anzahl an duplikaten Einträgen in einer Ontologie vermindert. Es war uns möglich eine hohe Präzision mit einer beachtlichen Anzahl an Vereinigungen zu erreichen. In einer angepassten, auf Freebase Daten basierenden Ontologie mit 50 Millionen Entitäten war es uns möglich 1,2 Millionen Duplikate zu identifizieren.

Abstract

The ever increasing demand for ontologies coupled with some ontologies' intention of accumulating human knowledge across all domains, generates ever larger data sets. Cross-domain ontologies, like e.g. Freebase, aggregate data from various sources, which in turn can easily lead to entries describing the same real world entity. We introduce an algorithm to lower the number of duplicates of an ontology by, based on entity names, identifying and merging duplicate entities. Our approach is not limited by ontology size and uses a relation-based scoring schema to detect duplicates. We have been able to accomplish high precision with a considerable of number of merges. For an adjusted ontology based on data from Freebase with 50 million entities we were able to find 1,2 million duplicate entries.

1 Introduction

The demand for comprehensive ontologies has been strongly growing with the gain in popularity of semantic search. Various projects curate ever expanding ontologies, which are formal representations of knowledge i.e. knowledge bases. Some of them focus on domain specific user generated content, like Geonames[1], IMDB[2], MusicBrainz[3] and Open Library[4], just to name a few. Others focus on various forms of content and information extraction from websites like Wikipedia, some of examples are the DBpedia[5], WordNet[6] and BabelNet[7] projects. Besides these project there are even more ambitious ones like Freebase[8] and YAGO[9]. They accumulate data from domain specific knowledge bases and extend it by means of information extraction from internet ressources. On top of that, in the case of Freebase, user contributions further expand the extent of information stored.

These unimaginably big troves of accumulated information with millions of entities and tens of millons of facts are used by semantic search engines to answer incomming queries. The ontology is the underlying knowledge the engine is being built upon. For instance, Google's search engine supplements its search result using their *Knowledge Graph* knowledge base, which is based on information from Freebase, the CIA World Factbook, Wikipedia and other sources[10]. The quality of an ontology is thus an essential part in correctly answering user queries.

Yet due to the way ontologies for semantic search are often assembled, the consistency of the information it contains can not be guaranteed. Having fully validated facts is generally impossible to achieve, but having as little as possible inconsistencies and duplicate information is very much expected. A single object from the real world may easily appear in multiple of the domain specific ontologies the search engine's ontology is constructed of. Although the curators of an ontology generally attempt to merge the various entities when combining knowledge bases, they do not always succeed, or have ulterior motives for not merging very likely duplicates like the off

[1] http://www.geonames.org/
[2] http://www.imdb.com/
[3] http://musicbrainz.org/
[4] https://openlibrary.org/
[5] http://dbpedia.org/
[6] http://wordnet.princeton.edu/
[7] http://babelnet.org/
[8] https://www.freebase.com/
[9] http://www.mpi-inf.mpg.de/yago-naga/yago/
[10] http://googleblog.blogspot.co.uk/2012/05/introducing-knowledge-graph-things-not.html

chance of data corruption. Although the fear of corruption is justified, this behavior limits the usability of the data for the purposes of semantic search. In our analyzed ontology we have found copious amounts of duplicate entities that contain identical information.

In addition to the merging of ontologies being a source of duplicate generation, the fact that many projects of knowledge databases also allow and encourage individual contributions adds an additional source of duplicate information and may furthermore cause inconsistencies to arises.

The issue of defining duplicity for a single object adds to the problem of ontology reconciliation. For example, there can be various ways to define a song's uniqueness, like its length, the album it appeared on, the type of recording it is, whether it has been equalized and the selection of audio tracks used. Depending on the engineer of the search engine and its intended use, the characteristics that make an object unique may thus easily deviate.

To solve these challenges, we introduce an algorithm to find and merge duplicate entities in ontologies. The developed algorithm finds duplicate entities based on their name and, if the correlation of their respective relations surpasses a predefined threshold, merges the entities into one. To furthermore allow the operator control over the unification process, she is able to put emphasis on the gravity of any relation's impact. Our approach operates on ontologies of various sizes and with diverse structure. By using relative similarity measures, it is able to handle sparse as well as rich entities and through continuous processing has no upper limit to the number of facts or relations of the ontology being processed.

1.1 Structure of this Thesis

We have divided our paper into the following five chapters:

- The first chapter has shown our motivation for dealing with the topic of entity duplication.
- In the second chapter we display related work and in which way it differs from our approach.
- Our algorithm including the inner workings of the scoring schema is presented in the third chapter.
- The forth chapter covers the application of our algorithm onto a full-sized ontology and our findings and results of it.
- Lastly, the fifth chapter gives a conclusion and points out future work.

2 Related Work

The issue of improving the quality of a database by means of duplicate detection and has been a constant field of research for many years. This includes the usage of various forms of similarity measures.

A groundlaying approach has been [DSD98] with the goal of merging two customer databases. In each the entities have their respective identifiers, yet two entities may describe the same real world object. Their approach has been to analyze the distance between data available about the two entities from their respective database. They used different distance functions for various types of data and a weighting schema, filled out by people familiar with the data sets. Their approach furthermore strongly focuses on incoherent data and errors in the datasets, which had to be overcome. Although their approach coincides to some degree with ours, the algorithms themselves vary strongly due to the different underlying data, which in our case is an ontology with more complex relations, including linkages of entities and with a multitude of different relations.

Another, more recent approach from [KI13] has used Formal Concept Analysis to tackle the problem of data redundancy in ontologies. FCA, for their use, describes methods of creating biclusters of the form object-attribute. The ontology is initially being transformed into formal context and based on the proximity of the objects within their cluster, duplicates are being detected. Their approach has focused on finding duplicates in ontologies containing people and companies with tens of thousands of entities. In contrast to this stands the enormity of our data set of more than 50 million entities of various domains, making their proposed approach become no longer viable.

From the Data Quality Management perspective, removing duplicates is also an important aspect. The approach presented in [BG09] is based on a two step procedure, with the first being a learning phase and the later being the application phase. In the learning phase the algorithm is presented with pre-labeled duplicates and non-duplicates, through which it decides the weight of relation similarities. These weights are used in the application phase of the algorithm to make decisions about duplicates. In our case, a learning phase may be very large scale and would take a large amount of work, as every relation's weight needs to be learned, which with multiple thousands of relations becomes less feasible.

The unification of entities of two different ontologies with a focus on location-based data, as covered in [Ste13], is related to our approach. Its decision process also uses

a relation based scoring schema and the underlying ontology format similar. Yet the size of the underlying ontologies and the focus on one domain limits it's application.

Although there are various concepts to finding duplicate entries, none of them have particularly matched our task at hand, as we are interested in an approach not limited by the size or diversity of the underlying data of the ontology.

3 Reconciliation

In the following we will display our approach to ontology reconciliation, the workings of our algorithm and the ground on which the algorithm makes it decision.

Briefly describing the sequential process of the algorithm, the initial step constitutes of the generation of a scoring configuration. This contains relation-based scores that are alterable by the operator. This is followed by the preprocessing of the given ontology, which bundles each entity's information, making it easily accessible once needed. The duplicate detection process filters on basis of the entity's name and executes a relation-based comparison process for each pair of possible duplicates using the previously defined scores. Successful identification of a duplicate results in the merging of the two entities. The merge consists of linking the entities and optionally merging its facts.

3.1 Central Concept

Our approach to solving the issue of duplicate entities is based on comparing entities in the context of their relations. Entities on their own, without their relations, are only unique identifiers, which in the most generic case are not in human readable form. If the name of an entity is part of its identifier, it is something to start with, yet without more information about two entities, merging on the basis of them them having the same name, would be risky and in many cases plain wrong. By using all of an entities relations for the reconciliation process, we use all relevant information available in the ontology, including relations where the entity is a fact of an relation. This allows the algorithm to make an educated decision for merging two entities.

On the basis of this comprehensive knowledge about any given entity that is in question of being a duplicate, the decision whether it is one is based on a scoring scheme. This scoring scheme, which builds on the foundations laid out in [Ste13], allows the user to adjust the importance of matching and mismatching relations. This scheme is of fundamental importance when it comes to merging entities with diverging facts to common relations. For entities with common facts to common relations, there are no discrepancies in their information and merging the two entities is the logical decision. Yet if two entities share a relation but have incoherent facts for it, correct merging becomes no longer trivial. The scoring system awards and penalizes consistency and divergence of a relation's facts, respectively. If the overall

score for all relations of two entities exceeds a predefined threshold the entities are being merged.

3.2 Underlying Ontology

The algorithm has been implemented to work on an ontology which contains triples of information. Each triple is of the turtle format, i.e. of the form (subject, predicate, object), as it is common form for ontologies. The predicate is the name of the relation and the object is a fact of the relation. Depending on the relation there may be just one or many facts to a relation.

We treat the subject and object of a triple each as the unique identifier of that entity. Each entity is regarded as a possible candidate for being a duplicate. This goes with the exception of relations whose facts are numbers or dates. For these only the subject is treated as an entity.

For our algorithm the subjects and objects are regarded as entities and consist of the form 'stem (unqiue-suffix)', e.g.: 'Phobos (z7)'. Each entity from the given input ontology is thus uniquely identifiable by its stem-suffix combination, e.g. as displayed in Table 3.1.

Phobos (z7)	is-a	moon (b1)
Phobos (z7)	is-a	celestial body (b4)
Phobos (z7)	date-of-discovery	1877-08-18
Phobos (z7)	mean radius (km)	11.27
Phobos (z7)	orbital period (days)	0.32
Phobos (z7)	alternative names	Mars I (c7)
Phobos (0r)	is-a	god (gg)

Table 3.1: Example Ontology Entry in Turtle Syntax

For our purposes, the source of the ontology data is the Freebase ontology. The data has been preprocessed in an earlier step to match the form as needed for the algorithm. Additionally, further processing has been done to reduce the amount of data, while keeping essential relations. Through this we have ended up with an ontology of ~307 million facts and ~50 million entities.

3.3 Internal Data Structure

To be able to make educated decisions in reasonable time, a data structure that allows easy access to each of an entity's facts is needed. This is being achieved by using additional layers of abstraction and indirection.

When processing the input ontology, each entity's facts are parsed and an entry is added to the list of all entities. Facts thats objects are entities are also processed and added to the list of entities.

Relations are stored separately, since their scoring values, read from the previously created score configuration, are needed for the later comparison process. Additionally, as each relation is only stored once, this allows the gathering of data about, for instance, the average and/or maximum number of facts per subject for a relation. This may be used as an indicator for the score value of a relation. As this topic unfortunately is beyond the scope of this paper, we have resorted to including it in the Future Work section.

For each of a subject's relations, either, if the object is a date or number, the value with a reference to the relation is stored; or, if the object is an entity, a reference to the relation and a reference to the entity are stored.

As the occurrence of an entity as an object is just as much an information as its occurrence as a subject, entities occurring as objects also store a reference to the relation and a reference to the subject. This though just with the variation that the reference is particularly marked as a reverse relation, which allows a independent scoring value for the reverse relation.

If two entities have been marked as duplicates and consequently merged, this holds relevant information for the following entity comparisons and improves the chance of finding more duplicates. To incorporate this into our design, we have added an additional layer of abstraction. This layer constitutes of each entity being accessed only through a proxy entity. As such, if a entity is merged with another entity, its proxy entity is redirected to the new entity. We thus gain the advantage of having consecutive comparisons profit from previous merges, since they already consider the two merged entities identical. This becomes particularly visible when iterating multiple times over a given data set, as mentioned in our Chapter chapter 4.

To accommodate ontologies with data sets too large to be processed all at once, we furthermore preprocess the ontology. First we explicitly add every entity's reverse relations to the existing ontology. Although this information is redundant, it allows us to gather all information about an entity, i.e. all facts it is part of, by only searching for its appearances as a subject. Secondly we sort this expanded ontology. This brings together all of an entity's information value. From this, we chunk-wise process the data, as above mentioned.

3.3.1 Entity Definition

Since our approach aims at being able to handle all kinds of ontologies of the required input format containing various information and not only information related to a specific domain. The kinds of relations that occur can consequently describe all sorts of facts. To overcome this issue, we introduce the classification of relations into

three types: Set-Relations, Float-Relations and Date-Relations. This gives us the ability to use different comparison approaches for each types. With this classification of an entity's relations, we can express an entity as a 5-tuple of the following form:

$$e = (id, stem, S, F, D)$$

Here, *id* is a unique identifier of the entity as used in the ontology. The *stem* is the result of a transformation function applied to the *id*, which will be used for the comparison process. Based on our underlying ontology the transformation trims off the suffix-identifier of the entry, such that only the stem remains. The relation information of the entity is split up into the three relation types, where Set-Relations, Float-Relations and Date-Relations are stored in S, F and D, respectively.

Anticipating the following chapter to give an idea of the concept of the 5-tuple, the entity $e_{\text{Phobos (z7)}}$ shown in Table 3.1 would be expressed as a 5-tuple in the following way:

$id_{\text{Phobos (z7)}} = \text{Phobos (12345)}$

$stem_{\text{Phobos (z7)}} = \text{Phobos}$

$S_{\text{Phobos (z7)}} = \left\{ \left(\text{is-a}, \{e_{\text{moon (b1)}}, e_{\text{celestial body (b4)}}\}\right), \left(\text{alternative names}, \{e_{\text{Mars I (c7)}}\}\right) \right\}$

$F_{\text{Phobos (z7)}} = \{(\text{mean radius (km)}, 11.27), (\text{orbital period (days)}, 0.32)\}$

$D_{\text{Phobos (z7)}} = \{(\text{date-of-discovery}, (1877, 8, 18))\}$

3.3.2 Relation Data

Each of the three relation sets of an entity consists of pairs, where the first entry is the name of the relation and the second a single fact or a set of facts, depending on the type.

The relations of the type Set-Relation generally make up the bulk of an entity's relations. Each of them may have a variable amount of facts. E.g. the entity '*Bob Marley*' has multiple facts to the *'is-a'* relation, them being among others: *'person', 'artist', 'musician'* and *'composer'*. More over, each of the facts of a Set-Relation is itself regarded an entity. As, this type is applicable to many relations, we used it for those relations which are not specifically labelled as Float- or Date-relations by the previously created relation-based scoring configuration.

The form of the Set-Relations of an entity can be described the following way: Let E be the set of all entities and P be the set of all relations names, then for an entity $(id, stem, S, F, D) \in E$ is S defined as a set of pairs:

$$S = \{(r, A) \mid r \in P, A \subseteq E \wedge \forall a \in A : (e, r, a) \in Facts\}$$

Float-Relations are relations that have as a fact a floating point number or an integer. Each of these relations may only have one fact. Examples are the relations '*height*', '*fleet size*' or '*atomic number*'. For some relations their values can only be stated up to a certain precision; this is being considered in the merge process by allowing the facts of Float-Relations to be within a predefined range. The set of Float-Relations F is consequently defined, using above's definitions, as:

$$F = \{(r, f) \mid r \in P, f \in R \wedge (e, r, f) \in Facts\}$$

The set of Date-Relations contains relations whose fact is in a date format. Here also only one fact may exist at a time for a given relation. For many entities these date relations have very defining information, e.g. *'date of birth'*, *'date founded'* or *'date of death'*; for each of these relations having multiple facts would generally not make sense, as only one date can be correct for a specific relation and be used to be a part of what defines an entity. Relations justifiably containing multiple dates are best be treated as Set-Relations, as applying the date specific comparison approach would not yield the intended results. The set of Date-Relations may be similarly described as above, with W being the set of all possible date triples (y, m, d), the set of date relations is of the form:

$$D = \{(r, d) \mid r \in P, d \in W \wedge (e, r, d) \in Facts\}$$

These sets of different relation types will be the central part of the duplicate detection process.

3.4 Comparison Process

To completely and thoroughly find every duplicate, one would have to naively compare every entity with every other entity in the ontology. This task would take enormous time and processing ressources as it is in $O(n^2)$ in the number of entities alone, not including the entities' relation comparisons; making it less and less feasible with the constant growth in size of modern day ontologies.

During our data analysis we were able to make out that many duplicate entries in fact have the same or a very similar name as their original, which is also pointed out by [FDA-PAPER]. Thus, to be able to get a hold on reconciling vast ontologies with more than 50 million entities, such as in our case, we focused the scope of duplicate detection to only those entities with the same name stem. This, if successful, invariably improves the ontology's usage for semantic search as it allows the search engine to respond with less hits containing higher degree information.

As such, our comparison process for duplicate detection is based on the pairwise comparison of entities of preprocessed ontological data. The preprocessing, done either by the underlying data structure, if memory size suffices, or alternatively on

disk, sorts the entities lexicographically such that entities with the same name stem are adjacent to one another.

Consequently this limits the comparison process only to adjacent entities with the same stem or, in the case of multiple entities, ranges of entities. For these ranges each combination of two entities undergoes a pairwise comparison process to determine the similarity of their relations. Based on the result of this process, which uses a predefined scoring schema to leverage similarities, the two entities are either unified, by merging their relations, or left separate. Recalling above's definition, the set of pairs that make out the pairwise comparison can be described more formally as:

$\{(e_1, e_2) \mid e_1, e_2 \in E : e_1.stem = e_2.stem, e_1.id < e_2.id\}$.

For our purposes, the name stem describes the full name of the object the entity represents. These stems may be as specific as *'6th Marine Division on Okinawa'* or as ambiguous as *'Introduction'*.

3.4.1 Pairwise Comparison

Deciding whether two entities with the same name stem are duplicates and to be merged is as aforementioned based on the similarity of their relations. Yet due to the various natures of the relations, the equivalence of some relations is more conclusive than that of others. For example the event of two entities in comparison having the same date to the *'date of birth'* relation, allows a strong tendency towards assuming them duplicates, provided there is are no valid discrepancies, since the chance of two persons having the same name as well as the same date of birth is quite small. Likewise the non-existence of common facts for some relations may not indisputably ensure each ones genuineness. For instance, the facts to the relation *'acted-in'* for one of the two entities may be a whole array of films: *'Goldfinger'*, '*Thunderball*' and '*You Only Live Twice*', while the other only holds '*Entrapment*'. These are all movies starring the same actor, *'Sean Connery'*, yet the sets of facts are disjoint. This implies that the relation *'acted-in'* should not be weighted too heavily a factor in deciding if two entities should be merged.

Thus, some relations are more defining than others. To accommodate this behavior we introduce a scoring schema, which allows the operator to in advance give each relation a score for a match or a mismatch of its facts. For each entity comparison the algorithm iterates over all common relations, and a total score is aggregated across all relation comparisons. If this score reaches a predefined threshold, the entities along with their relations are merged.

3.4.1.1 Relation Type Based Scoring

The introduced classification of relations evidently demands for each relation type a separate formula for calculating similarity, and rewarding coherence and punishing divergence.

For SET-RELATIONS, i.e. relations with a set of facts, the overlap of the two sets is being used to compute the score. Thus, for entities $e_1 = (id_1, stem_1, S_1, F_1D_1), e_2 = (id_2, stem_2, S_2, F_2D_2)$ is the score for relation r with $(r, A_1) \in S_1, (r, A_2) \in S_2$:

$$score_r = \frac{|A_1 \cap A_2|}{min(|A_1|, |A_2|)} \cdot (|match_r - mismatch_r|) + mismatch_r$$

This way, if one fact set is a subset of the other: $\frac{|A_1 \cap A_2|}{min(|A_1|,|A_2|)} = 1$, i.e. one relation's set of facts fully covers the other relation's set, a full score is awarded. If, on the other hand, the two sets are disjoint, the score has a penalizing function by giving the mismatch score for the relation. In cases where both sets have unique and common elements, the fraction of the overlap determines the generated score.

We generally consider mismatch scores to be below the score threshold, such that applying it lowers the chances of two entities being merged.

For FloatRelations the comparison allows the two relation's facts to be within a certain, predefined threshold. Taking the above declared entities e_1, e_2, the score of a FloatRelation r with $(r, f_1) \in F_1, (r, f_2) \in F_2$ is:

$$score_r = \begin{cases} match_r & |f_1 - f_2| \leq threshold_r \\ mismatch_r & |f_1 - f_2| > threshold_r \end{cases}$$

DATE-RELATION comparison aims at rewarding the coherence of the relation's date-describing facts. They are regarded as equal as long as they do not contradict one another. Non-contradicting facts are if one fact is more or equally precise as the another, such as one describing only a year and the other a specific month of that same year. For e_1, e_2 with DATE-RELATION r such as $(r, (y_1, m_1, d_1)) \in D_1, (r, (y_2, m_2, d_2)) \in D_2$, is the score:

$$score_r = \begin{cases} mismatch_r & y_1 \neq y_2 \wedge m_1 \neq m_2 \wedge d_1 \neq d_2 \\ match_r & else \end{cases}$$

3.4.2 Merge Decision

With the computed scores from various relations, we have a similarity measure for each common relation. Since each relation's mismatch and match score can also be adjusted to express the gravity of the relation comparison's outcome, the individual scores now need to be put into perspective.

As earlier mentioned, each relation comparison needs to overcome a predefined threshold in order to be merged. An initial approach of using the sum of the comparison's scores yielded too little flexibility for the various kinds of entities we encountered. Since the entities may be of various origins, and thereby diverse in

composition, the number of relations from entity to entity can strongly vary. Thus the sum of some entity comparisons may, caused by their scarcity in relations, not be able to reach an absolute threshold, even though their relations are identical.

We adjusted the algorithm to better handle scarce relations by normalizing the generated score and the predefined threshold. This allows merging decisions to be based on relative similarity of the two entities.

3.4.2.1 Overlapping Relation Sets

For two entities sharing the same sets of relations, the average similarity of these relations composes the total score, and as such does not take more consideration, as all of their relations are being compared and considered in the merge decision.

With entities with overlapping relation sets on the other hand, one of the entities may have more relations than the other. These relations, due to being unique to one entity, were not being compared in the relation comparison progress, as there are no grounds for a comparison. Yet, due to the fact that only one of the entities has these additional relations, there is no direct source of conflict. All relations of the smaller entity are being compared, making the basis of the decision stay the same. This does imply that these additional information does not alter compared relations validity. This can in our view only occur if there is inconsistent information in the ontologies data.

As such the final score for $e_1 = (id_1, stem_1, S_1, F_1, D_1), e_2 = (id_2, stem_2, S_2, F_2, D_2)$ with $R_1 = S_1 \cup F_1 \cup D_1, R_2 = S_2 \cup F_2 \cup D_2$ under the condition of $R_1 \subseteq R_2$ or $R_1 \supseteq R_2$ is:

$$score(e_1,e_2) = \frac{\sum_{r \in R_1 \cap R_2} score_r}{|R_1 \cap R_2|}$$

3.4.2.2 Partially Disjoint Relation Sets

Additional attention must be directed towards the merge decision if both entities have unique relations. These relations can be to various degrees defining; for entities with few relations they can be of especially high importance.

E.g. table 3.2 displays two entities with very limited information, whose only common relation is '*is-a*', while each has one and two unique relations, respectively. Having no other information makes the decision whether they should be merged a coin toss. There is no contradiction in the information of the entities, but due to the scarcity of any other information there is no convincing factor for the two entities to be identical either, except for their common name. As precision is of much higher importance than recall, we penalize each mutually disjoint relation, such that the

Walter Mitty (e21)	
relation	fact
is-a	Person (d92)
is-a	Soldier (3f3)
profession	Adventurer (203)
cause of death	Firing Squad (2r1)

Walter Mitty (r03)	
relation	fact
is-a	Person (d92)
is-a	Daydreamer (zZz)
spouse	Mrs. Mitty (30f)

Table 3.2: Example of Partially Disjoint Relation Sets of Two Entities

fewer common, comparable relations two entities have in contrast to each's total number of relations, the higher the penalty.

To include this behaviour in the score calculation, the score is based not only on the mean score of their common relations, but also accounts for mutually disjoint relations. This is being done by treating each combination of relations, that both entities exclusively have, as a non scoring relation comparison. Differently put, the divisor for the calculation of the mean score over the two entities is the number of common relations plus the number of relations both have exclusively.

Be e_1, e_2 and R_1, R_2 defined as above mentioned, and $R_1 \nsubseteq R_2 \wedge R_1 \not\supseteq R_2$, the score of e_1 and e_2 is equal to:

$$score(e_1,e_2) = \frac{\sum_{r \in R_1 \cap R_2} score_r}{min(|R_1|, |R_2|)}$$

In the case of *Walter Mitty* of table 3.1, the total score is:

$score(e_{\text{e21}}, e_{\text{r03}}) = \frac{1}{2}|match_{\text{is-a}} - mismatch_{\text{is-a}}| + mismatch_{\text{is-a}}$ / 2

Eventually, if $score(e_1, e_2)$ exceeds the predefined global threshold, the two entities are merged.

3.5 Merging Entities

The merge process unifies the information of the two entities into a single entity and, by design of the underlying data structure, thereby implicitly redirects all references to the new entity.

The merging of some relations may cause a certain loss of data, as e.g. unifying two entities' *'height'* relations results into a single value for the new entity. The loss of the original value of the two merged entities' *'height'* relation may not always be desirable. We thus offer an optional merging system. The operator has the option to decide if the entities should be merged and the resulting ontology be returned

by the algorithm; or if only the mapping of duplicate entities should be returned, without further processing. The later allows the operator to subsequently apply a merging technique of their choosing onto the mapped entities.

In the following we shortly present our approach to merging two entities' relational data.

3.5.1 Relation Merging

In case of disjoint relations, the union of the two entities' relations is being used for the new entity. Yet for mergin common relations with disjoint facts, depending on the relation type, different strategies are being used.

As Set-Relations are relations consisting of sets of entity references, the merge process unifies the two sets of references. This ideally increases the number of facts for the relation, increasing the information value of the relation.

For Float-Relations, if the two values differ, the mean is of the two values is used henceforth for the relation. This works well for coordinates or other floating point number based relations. Though there are drawbacks for natural numbers.

Furthermore this does imply that, if one of the two relations has incorrect or even corrupt data, while the other has correct data, the new relation data is is corrupted as well. Take for example two entities that are being merged due to overwhelming similarities. If there has been a wrong value for the *'height'* relation of one of the two entries, say '1700', while the other holds '170', the mean of '935' is still erroneous.

For Date-Relations the more precise of the two dates is being taken. Yet, again, if the two relations do not match, and the entity still got merged, the question of which date is correct occurs. In our experience, for relations where a date is given it becomes one of the defining relations of the entity. A mismatch, in almost all cases we have seen, implies the that entities are not relating to the same object. Those that do have multiple date values we have treated as Set-Relations, therby keeping both values upon merging.

Visibly, a trade-off between generality and number of relation types has to be made when processing thousands of different relations. Going beyond the scope of this thesis, more research in this direction may lead to more comprehensive merging rules.

4 Evaluation

In the following we describe our findings of applying the introduced algorithm onto an actual data set. For this we have implemented the algorithm as a C++ program. We have complied it using g++-4.7 with c++11 support and the -O3 and -Wall flags. Testcases have been implemented using the Google C++ Testing Framework[1] and the format follows the Google C++ Style Guide[2].

4.1 Ontological Data

The ontology we have tested our implementation of the algorithm on is as previously described in turtle format and is based on data from the Freebase project. Certain relations have been transformed to suit our needs of the ontology being part of the semantic search engine of the Broccoli project of the Chair of Algorithms and Data Structures at the University of Freiburg[3]. The key characteristics of the ontology are described in Table 4.1.

(a) General

Number of Entities	**Number of Triples**	**Number of Relations**	**Filesize**
50,026,164	306,739,175	5,003	22,4 GB

(b) Duplicate Related

Number of Entities with Unique Names	**Number of Entities with Non-Unique Names**	**Number of Names**	**Number of Non-Unique Names**
24,259,597	25,766,567	29,614,530	5,354,933

Tabelle 4.1: Ontology Characteristics

Table 4.1 (b) shows that 52% of all entities have a name also used by another entity. Yet among these entities sharing common names, each name is used by on average 4.8 entities. This can be explained by the in Table 4.2 (a) displayed list of most used

[1] https://code.google.com/p/googletest/
[2] http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml
[3] [BBBH12]

names. It shows that the top ten entries alone make up 154,770 entities, which can be explained by the fact that certain names are very common, especially so among song titles.

To visualize the other end of the spectrum, Table 4.2 (b) shows how many duplications occur how often. E.g., there are 3,633,033 names, for each one of which there are exactly two entities.

(a) Most Commonly Used Names

Number of Entities	Name
43,561	[silence]
40,875	[untited]
30,075	Intro
6,328	Outro
6,026	Summertime
5,971	Home
5,883	Interlude
5,868	Silent Night
5,242	Untitled
4,941	Time

(b) Distribution of Duplicate Name Occurrences

Number of Entities per Name	Number of Occurrences
2	3,633,033
3	594,570
4	398,680
5	153,909
6	126,583
7-9	166,722
10 - 19	158,028
20 - 99	102,247
100+	21,161

Tabelle 4.2: Name Distribution

4.2 Quality

To allow for proper and easier quality control, we equipped our program with a particularly verbose logger, which gives detailed information about the origin of each merge decision. This makes it easier to inspect the reason for why each pair of entities has or has not been merged. This, though, is only applicable for ontologies of smaller size, as the log files for the inspected ontology go beyond proportions with a size of more than a terabyte each.

In order to still make use of the logger, we only logged a comparison with a $\frac{1}{1000}$ chance. The resulting log files give us a cross section of the entity comparisons. As such, we use these log files to investigate the quality-wise performance of our algorithm.

The first and foremost goal has been a as high as possible precision, while not ending up with a diminishingly small recall. Thus, we have composed and tested various scores for the different relations and adapted the global threshold to gain the most promising results. Table 4.3 displays the number of merges upon execution of our

program with a global threshold of $t_g = 10$ and $t_g = 15$ and over the course of three iterations.

t_g	**Iterations**	**Number of Unifications**
10	1	1,184,206
10	2	1,239,656
10	3	1,239,808
15	1	912,614
15	2	977,380
15	3	977,401

Tabelle 4.3: Results of the Unification Process

Evidently with a lower global threshold the number of unifications is higher, yet allows for more false positives. The additional iterations show that comparisons definitely benefit from previous merges, with an addition of 55,602 and 64,787 for $t_g = 10$ and $t_g = 15$ respectively.

4.2.1 Results of Random and Specific Sampling

To test the precision of the algorithm with a threshold of $t_g = 10$, we sampled at random 100 entries of the log file for merged and and not merged comparisons respectively. Additionally, we took a closer look at comparisons of persons, as some, due to scarcity in information, can easily be mistaken for another person, i.e., are false positives. Lastly, we analyzed particularly close calls of both, merged and not merged comparisons.

The set of 100 randomly selected comparisons leading to a merge showed that, based on the information provided by the ontology, 94% of the merges were correct, 2% false and 4% undecided. The undecided ones are entities where it depends on the definition of duplicity. The two false positives had scores of precisely 10, thus they would have not been merged with a slightly higher threshold.

The other set containing comparisons that failed to reach the global threshold, made only correct decisions, among these 100. Though one has to regard that the number of comparisons is a multitude of the number of entities. For instance, for the name '*[silence]*' there are are 950 million comparisons alone. Overall, approximately 1 in 2,500 comparisons lead to a merge.

Looking at 20 random comparisons of persons and their outcome, we saw that of specific concern are those entities, that have scarce relations. Their only informative relations either describe their profession or the name of their more famous spouse or other family member. As such there are no discrepancies but there is also no common ground, except for the name. By adjusting the scoring schema to accommodate this

behavior, we achieved a 95% precision rate, with one false positive being a film crew member and a film director & film crew member being merged.

To further analyze the quality, we have looked at merging decisions that are barely above or below the merging threshold. These cases predominantly consist on the one hand of comparisons with little data available, and on the other of decision which very much depend on the definition of duplicity. Examples of these entity comparisons, where one has to keep in mind that the name of the entities is identical, are:

- books from one author but with varying degree of correlation for publication date, OPAC number, ISBN number and publisher
- music albums for different countries, but containing the same content
- video games for different platforms

For these cases one can either modify the related relation scores, to make one relation weight more than another, or increase the global threshold, to allow attain a high precision.

If one were to only allow the merging of entities with overlapping relation sets and overlapping facts to common relations, i.e. each relation needs to score full points and no partially disjoint relations sets are tolerated, we discovered that still 810,230 entities would be merged.

Overall, we are able to form three major categories of comparison results, them being:

- close calls for and against duplicity among entities with few and noncommittal relations,
- easily distinguishable separate entities, due to mismatching facts, or strongly diverging relations,
- confident matches, because of similarity among all to almost all common relations,
- and disputable decision for and against merges depending on the definition of duplicity.

4.3 Limitations

An issue that may arise is that one uber-entity is being merged together. Since, if two entities are being merged, their relations are merged as well, this can in worse case scenario lead to one entity soaking up all other entities. This happens because entities with overlapping facts are merged. If one entity mostly overlaps with another entity they are, under the assertion of a moderate threshold, being merged, which

increases the resulting entity's number of relations. If one entity as such gradually merge with all other entities, one uber-entity for the given name may be the result.

Although the likelihood of this is very small, once an entity has amassed a large amount of diverse facts, its getting continuously easier for it to merge with other entities, as it is likely that enough relations overlap, especially of the other entity has only few relations.

We have also seen issues occurring if two entities describe the same real world object but all of the two entities' relations point to different entities with a similar or the same name. E.g. two music albums both contain a set of songs and these sets are disjoint, as they both contain different entities. Yet, actually the songs all have the same name and are unmerged duplicates themselves. If fortunate, the songs are getting merged in the first iteration step and the albums in the second. I.e. if there is a bubble of entities with interwoven relations, but without any other outside relations, all comparisons with these entities would result in disjoint sets.

This may be averted by, instead of comparing if the two entities are the same, comparing if the names of the two entities are the same. Yet this can yield other problems for e.g. *'Topic*' entities.

5 Conclusion

In this paper we have presented an algorithm for finding and merging duplicate entities of modern day ontologies. We displayed the outline of the algorithm's structure and gave insights into the scoring schema based decision making process. The algorithm has been applied on a modified Freebase ontology, allowing us to investigate quality and precision.

5.1 Future Work

There are various possible ways to expand on the contributions of the introduced algorithm.

For once, the generation of the scoring configuration may be improved by computing relation specific default scores based on a relation's usage. By analyzing the average number of facts for a specific relation and the domain of possible facts for a relation, a deduction can be made about how defining a match or mismatch for this relation is. The *'gender'* relation, for example, has only two possible facts - *'male'* or *'female'* - and occurs relatively often. A mismatch is a strong indicator for them being different real world objects. Yet a match is no indicator for them being the same object, it only implies that both are humans.

As previously mentioned, the merging process of our algorithm also allows room for further work. Particularly for FloatRelations, more precise distinctions between the types of data can be made. An evaluation of the domain of an relation's facts can also give hints about any erroneous data, since it allows the spotting of outliers; thereby reducing corruption.

A current limitation to our approach is the focus on entities only being compared if they have the same name. When it comes to investigating entities with variations in the naming, the algorithm does not compare the two entities. This may be circumvented by, for example using the edit distance of two entities' names for deciding if they should be compared.

Beyond this, energy put into further refinement of the scores and the evaluation of the outcome will further improve precision and recall.

Bibliography

[BBBH12] Bast, Hannah ; Bäurle, Florian ; Buchhold, Björn ; Haussmann, Elmar: Broccoli: Semantic Full-Text Search at your Fingertips. In: *CoRR* abs/1207.2615 (2012)

[BG09] Brüggemann, Stefan ; Grüning, Fabian: Using Ontologies Providing Domain Knowledge for Data Quality Management. In: Pellegrini, Tassilo (Hrsg.) ; Auer, Soren (Hrsg.) ; Tochtermann, Klaus (Hrsg.) ; Schaffert, Sebastian (Hrsg.): *Networked Knowledge - Networked Media* Bd. 221. Springer Berlin Heidelberg, 2009. – ISBN 978–3–642–02183–1, S. 187–203

[DSD98] Dey, D. ; Sarkar, S. ; De, P.: Entity matching in heterogeneous databases: a distance-based decision model. In: *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on* Bd. 7, 1998, S. 305–313 vol.7

[KI13] Klimushkin, M. A. ; Ilvovsky, D. A.: Detecting Duplicate Objects in Ontologies Using FCA. In: *Autom. Doc. Math. Linguist.* 47 (2013), Februar, Nr. 1, S. 10–18. – ISSN 0005–1055

[Ste13] Stepan, Anton. *Entity Unification for Semantic Search.* 2013