Bachelor Thesis

Entity Unification for Semantic Search

Anton Stepan

18.07.2013

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

Bearbeitungszeitraum
18. 04. 2013 – 18. 07. 2013

Gutachter
Prof. Dr. Hannah Bast

Betreuer
M.Sc. Björn Buchhold

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, July 18th, 2013

Anton Stepan

Contents

Acknowledgments

At this point I want to thank my supervisor, Prof. Hannah Bast, who made this thesis possible by being very inspiring in her lectures. I also want to thank Björn Buchhold for his very good explanation of semantic search, his guidance throughout this work and the very helpful discussions.

Furthermore a special place in this list is reserved to my parents, for being a rock solid foundation in my life, supporting me throughout my whole students' career and also to all my friends, especially Steph & Stella for their steady support in time of need.

Last but not least, I would like to thank my girlfriend, Lilli, providing me with her knowledge, her comfort and her unweary support despite my lack of time during the last years. I would not have been able to achive anything of this without her.

- yo -

Anton

Zusammenfassung

In dieser Arbeit untersuchen und präsentieren wir ein Verfahren zum Vereinigen von Ontologiedatensätzen mit dem Schwerpunkt auf ortsbezogenen Entitäten. Das damit angestrebte Ziel ist, eine größere und komplexere Eingabedatenbank für die semantische Suche zu erzeugen. Das in dieser Arbeit vorgestellte Verfahren erklärt eine mögliche Herangehensweise für dieses Problem, erläutert aufkommende Schwierigkeiten und demonstriert das Vorgehen anhand einer Anwendung. Unser Framework zur Vereinigung basiert auf der Idee von Punkten, die für übereinstimmende relationen zwischen zwei Entitäten vergeben und gesammelt werden. Es ist für ein Paar von Entitäten nur möglich sich zu verschmelzen, wenn sie mit ihren Punkten einen spezifizierten Schwellenwert erreichen. Diese Punkte und Schwellenwerte sind sehr flexibel und können durch den Benutzer in einer Konfigurationsdatei selbst festgelegt werden, mit dem Ziel das Ergebnis der Vereinigung zu optimieren oder eine besondere Gewichtung auf eine Kategorie von Vergleichen zu legen.

In einem Experiment wird die Performance und die Qualität überprüft, in dem eine Vereinigung zweier Datensätze, basierend auf Geonames (37 MB) und Freebase (244 MB), durchgeführt wird, und dabei eine Vereinigung von mehr als 50% der kleineren Ontologie, bei einer durchschnittlichen Laufzeit von 20 Sekunden, erreicht wird.

Abstract

In this paper we introduce and investigate a method to unify ontology datasets with location based data. The goal is to generate a bigger and more complex input set for semantic search engines. Our research explains a possible unification framework to solve this problem, outlines arising obstructions and demonstrates our algorithm with an application and an unification framework. The framework is based on the idea of awarding scores in the unification process, gathered from matching relations and only unifying two entities if they reach a specific score threshold. These scores and thresholds are very flexible as the user can define them in a config file, with the objective to tweak the result or set a specific weighting on a particular comparison category.

With an experiment we review the performance and quality of our approach by unifying two datasets, based on Geonames (37 MB) and Freebase (244 MB) and achive more than 50% unification of the smaller ontology with an average execution time of 20 seconds.

1. Introduction

In this thesis, we introduce and discuss the problem of entity unification that emerges while extending datasets behind semantic search. Many of those datasets use different entities for identical objects, which prevents a trivial merge algorithm. To be able to unify those entities we need to consider the further information about the objects, especially their relations. The following chapter introduces the motivation for the work, points out our contribution and gives a brief overview about the structure of this thesis.

1.1. Motivation

With the enormous growth of the internet new data is published continously and it may be hard for users to find the information they are looking for. One of the key factors for the huge success of the word wide web are search engines, that provide a userfriendly method to search the internet for specific topics. With fairly good results on most queries, users are able to find quickly answers to their questions. But there are still complex queries that sometimes do not return satisfying results. If a user e.g. is looking for a list of „all scientists who were born in Berlin (Germany)“, the classical full text search, to put it simple, will only return documents where our query words appear in the text, and not a complete list the user is looking for. If we are lucky, someone created such a list and published it on his website, but this is rarely the case for complexe queries. The major problem of the classical search engines is that they are not able to identify the „semantics“ - the meaning - behind our query. For the above example query it does not know that „Berlin“ is an instance of the class location, that „born in“ is a relation between two entities, or that scientists is a class describing a special kind of persons. A semantic search engine is able to identify entities in a search query, assign classes to the entities and interprete the query, which leads to a result of specific classes instead of documents just containing the words.

„Semantic Search“ is an upcoming hot topic in the field of computer science as many big companies, for example Google[1] are conducting research at it. Also at the University of Freiburg at the Faculty of Engineering, the Chair of Algorithms and Data Structures is researching in this topic and implementing its own semantic search engine presented in the paper „Broccoli: Semantic Full-Text Search at your

[1] http://googleblog.blogspot.de/2012/05/introducing-knowledge-graph-things-not.html

Fingertips“ [2]. In semantic search the engine can identify the entities and relations in the query and then „understand“ its meaning. In the previous example query „all scientists who were born in Berlin“ the engine would understand that „scientists“ is a class, „Berlin“ an instance of a class and „born in“ a relation. It would then return all entities of the class „Scientist“, which have a „born-in“ relation with the object „Berlin“. To be able to idenfity all entities of a query, semantic search engines need as many entities as possible in their database to recognize as much of the query as possible. In this example we need information of entities which are of the class scientist and have a relation which represents their place of birth. We also need the information whether this birthplaces are actually located in Berlin, Germany. Our result can only be satisfying if our entity database has enough data to fulfill our query by identifying the entities and match them with the corresponding database entries. It implies that the search results can only be as good as our entity database. Usually the problem occurs that there is no single entity database, but multiple small ones with each having a specific focus (i.e. locations, scientists, celebrities, brands, ...). To achieve the best possible results you need to combine those smaller databases to a single one and be able to match and merge equal entities from different sources. This unification process can be a very difficult task as many problems occure, like for example the same entity in database 1 could have a completely different name in database 2.

In this thesis we will discuss the results and the emerging problems we solved during the creation of our EntityUnification program, which takes two entity databases as an input and generates an unified output. The programm was developed using C++, has about 2250 lines and test cases for the major parts using Googles test suite „gtest“. We will point out the occuring problems in the relation based unification process and present our suggested solutions. This paper shall describe several arising problems of entity unification, reveal our intention to solve those problems and compare different approaches based on the evaluation of the unified output.

1.2. Contribution

The main contributions of this thesis are:

- Identifying the major problems occuring in the process of entity unfication.
- An approach and implementation to unify two datasets with a strong focus on location based entities with our program EntityUnification.
- An evaluation and explaination of different tuning parameters in our unification algorithm in an experiment by unifying two datasets based on Geonames and Freebase.
- Reveal problems for future research, which were too rich in detail for this work.

[2] http://broccoli.informatik.uni-freiburg.de/

While our framework works very well and achives nice results (see Chap. 4.1), there are still many possible improvements to do. Therefore our thesis contains an outlook on future enhancements, some already with a solution outline.

1.3. Structure of this Thesis

This work is separated into five chapters, which will lead to explain, evaluate and discuss the problem of entity unification for semantic search.

- Chapter 1 **introduces** the backgrounds of this thesis and points out the **contribution** we would like to achieve.
- Chapter 2 provides an overview of other **related work** which has been done in this particular topic.
- Chapter 3 describes the whole **unification process** from parsing the files, checking whether two entities are the same and explaining the problems occuring during the unfication process.
- Chapter 4 **evaluates** the results of our approach and compares different tuning parameters.
- Chapter 5 sums up our contribution and discusses **possible improvements** and **future research** possibilities.

2. Related Work

The unification of different entity databases is an often occuring problem as there are multiple small- and midsized databases which have a strong focus on a certain topic but no allrounder. This problem often needs a strongly customized solution to provide a satisfying result, mostly done by hand, as each database has its own syntax and naming conventions. In the topic of semantic search we want to combine different entity databases to improve the backend of our engines and therefor improve our query results. Other research papers often have the benefit of merging databases which have the same naming convention as a basis.

For example PROMPT [NM00] describes an semi-automated ontology unification process. Their algorithm compares entity classes based on their name and tries to merge them, if possible. On conflicts the algorithm suggests possible solutions so that the user can decide whether a merge is possible or not.

Another work called NERD [RT12] focuses on the unification of named entities provided by ten popular named entity extractors using their APIs. As a user you are able to query the REST API[1] of NERD, as NERD will deligate your query to its extraction driver, which will query the named entity extractor websites. Afterwards NERD will retrieve their output, unify it and map the annotations to the „NERD ontology“, which is manually created schemes mapper for the different entity extractor website outputs.

The unification of ontologies is also a big topic in the field of life sciences, as the unification of gene ontologies is needed to create a large database to standardize the representation of genes and their product attributes. „Gene Ontology: tool for the unification of biology“ [Ash00] provides a implementation to combine different gene databases into one by which biologists do neither need to search in all available informations nor need to get used to the wide variations of different terminology.

At this point we also want to mention GeoReader [SB], a project with the title „Relation Extraction: Extracting relations from the GeoNames database“ to extract entities from the geographical database GeoNames [Geo10] and to transform them into an ontology database. In this thesis we used a modified version of the GeoReader to create a database with GeoNames information exactly to our necessities.

The second dataset for our experiment is based on the ontology of Freebase [BEP+08], a huge community-curated database containing more than one billion facts.

[1] http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

In contrast to the presented work in this chapter, our approach operates without manual interaction during the unification phase and follows the idea of predefined scores and thresholds for containing certain similarities.

3. Unification

This chapter explains the unification algorithm of our program „EntityUnification“, the biggest problems we faced and our corresponding solutions.

3.1. Overview

The input are two datasets consisting of ontologies, a formally representation of knowledge, that get unified. These files provide information about entities and their relations in a structured way, for example that *Freiburg* is a city and that it is located in *Germany*.

The programm we developed consists of the following steps, which are diagrammed in figure 3.1. The main phase of the algorithm is step 3 („Unify Entities“) and is divided into three sub-problems:

1. **Pre Check**: The pre check determines how likely it is for a pair of entities to be identical. Only a successful pre check leads to an expensive full check.
2. **Full Check**: The full check is a complete comparison between a pair of entities including their additional information and relations.
3. **Unify**: A successful full check starts the unification, combining the pair to a single entity.

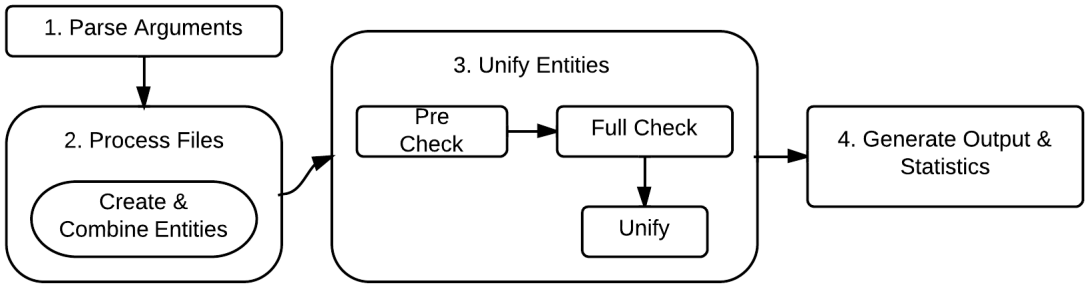

Figure 3.1.: Steps of Entity Unification program

3.1.1. Parsing command line arguments

The following options are available (* are required):

- *filenames for both datasets
- *scores filename: File with different scores and thresholds for the unification algorithm
- output filename
- relationmap filename: defines the map to translate relations to different names or extend them.
  - translate: „has-population“ and „population-count“ are the identical relation with different name, therefore it makes sense to only one name (see Chap. 3.3.6).
- iterations: count of unfiy-phase iterations between the two databases (see Punkt 3.4.1).
- debug: enable or disable the debug functionality, which generates debug files for merged entities and likely equal entities which did not got merged with their scores
- generate example files: creates example files for config, relationmap and scores

Instead of using lots of command line arguemnts, it is also possible to define a standard-path to a config file which defines all arguments.

3.1.2. Processing Files

In this step the two database files are parsed, the entities generated and stored in two maps (one for each database) which are sorted lexicographically. The key for those maps is the entity ID and the value is a pointer datastructure pointing to the real entity which contains all available information, i.e. relations, alternate names or the number of entities pointing to it. The usage of this pointer datastructure makes it easily possible to leverage a merge between two entities to all occurences of them, as each entity is represented by its pointer datastructure. If a pointer datastructure changes its target, i.e. through a merge, every entity which pointed to it will now also point to the new target. A side benefit of this approach is the reduced memory allocation as each entity is exactly once in the main memory while all its occurences are represented by the small pointer datastructure.

3.1.3. Unification of Entities

After the maps got generated and all input data is parsed, the unification phase starts at the top of the two maps and compares the entities, one of each map.

Before the expensive full check of two entities is executed, a pre check compares the entities and only initiates the full check if the two entities are likely equal. The full check compares all names, alternative names and relations of the entities and calculates scores on the basis of hits and contradictions. If two entities exceed a critical treshold, the two entities are getting merged. Different scores and thresholds make sense, because thereby the user is able to tweak the result or set a weight on a specific categories, i.e. geographical information more important than naming equality. Hence it is important that these values can be configured. For the sake of useability, this is available through the scores config file, which can be modified without recompiling the program. Regardless of an unification or a termination by a failed check, the map iterator of the lexicographically smaller ID is getting increased to move through the maps until both iterators reach the end.

This step will be repeated as many times as defined by the iteration count to make use of newly created relations which were generated by the mergers.

If the debugging flag is set, two debug files are created during this unification process. On the one side it generates a file with all entities, which got merged and on the other the entities which passed the precheck but did not get merged. These debugging files also include the final score of the comparions and the relations of the entities in a readable format.

3.1.4. Output and statistics

The final step generates the output of the two unified databases as an ontology file. It is also possible to highlight merged entities in the output file by a config parameter to indicate by which entities they were unified. Also statistical information is getting calculated and reported, including the line counts of the two input files, the entity count of the two databases and the amount of successfull unified entities.

3.2. Preprocessing

Before the unification process can be started the format of the input databases has to be adjusted. For the EntityUnification program the input files need to be in turtle syntax, a notation to represent information by triples, each with a subject, a predicate and an object delimited by a tabulator. In our example the lines of each ontology file has the following appearance:

<Entity-ID><TAB><Relation><TAB><Entity-ID or Attribute-Name>

The entity IDs have to be unique within their source database which occasionally leads to an unreadable or very complex format, for example the following could all be the ID of the same entity: „*Berlin,_(Berlin),_(Germany)*“, „*Berlin,_(0156q)*“,

or „*187w184*“. Through their relations the IDs get their semantic meanings, i.e. full name, location, and geographical coordinates.

3.3. Recognizing similar entities

The main purpose of the unification process is to merge equal entities of different database to generate a single database without losing information. If the unification process operates correctly, the newly created database will have more detailed information about the source entities, i.e. we do not only know that „*Wolfgang Amadeus Mozart*“ is born in „*Salzburg*“, but also know that „*Salzburg*“ has a latitude of 47.48, a longitude of 13.20, a population number of 148,521 and is located in Austria. But how can we recognize if two entities are equal? This chapter points out the most important difficulties in detecting equal entities, presents a solution outline and explains the method of our approach.

3.3.1. Entities with slightly different name

In one database it could be a convention that entities are seperated by their ID which allows the multiple occurence of the same name. It could also be possible that the name of the entity is simultaneously its ID, by using unique names. Or it could also be possible that a database has the naming convention to use just the plain name of an entity while any other might use a combination of its name and a suitable suffix. The problem occures to identify two such entities from different naming conventions, i.e. the equal entities „*Freiburg*“ and „*Freiburg_im_Breisgau,_(Germany)*“ or the unequal entities „*Jimmy_Walker,_(Actor)*“ and „*Jimmy_Walker*“ the former mayor of New York City. For example Freebase [BEP+08] disjoins IDs and names, as it uses unreadable IDs throughout its complete dataset and maps the name of an entity with a naming relation, while the knowledge databases DBPedia [1] or yago [SKW07] use unique names to distinguish between the entities.

But how do we determine whether those entities might be the same regardless of their different format?

A possible solution would be to compare the edit distance or the length of the longest substring of the two entity names to determine whether it is likely that the entities are similar. Another solution would be to compare the entity name prefix until a specified delimiter (i.e. first space or last comma) or a defined length. As equal entities from different datasets most likely do not exactly have the same name but often various similarities in it, it is a good idea to do a substring test (edit distance, longest substring, prefix, ...) with their names.

[1] http://de.dbpedia.org/

EntityUnification: Our approach extracts the name of every entity in the file processing phase and then identifies entities based on the prefix (the first word) of their names to determine in the pre check part whether a pair of entities are likely identical.

3.3.2. Multiple entities with identical name

Many entities in our source databases have the same name and are just seperated by an unique ID. But every source has its own unique IDs, and its own namespace and naming conventions. In this way for example, we might have eight entities with the name „*Germany*“ in database A, and thirteen in database B, and we need to find out which „*Germany*“ from A is equal to a „*Germany*“ of B.

As there are lots of entities with the same name in every database, we have to concentrate on the additional information of the entities: their relations. With the comparison of relations between two possibly equal entities we can observe whether they have multiple similarities within their relations or even contradictions. Are they located in the same country or federal state and do they contain the same locations? Do they have a similar population count? Are the geographical information (latitude, longitude) equal or totally different? By comparing those attributes and using a suitable weighting it could be determined how likely it is for two entities to be equal.

EntityUnification: In our algorithm the procedure to unfiy two entities compares all entities with identical name, as the entities are stored in a sorted array and iterated regarding their names.

3.3.3. Same entities with different name

As the databases are created by humans from different countries, it can be the case that the databases are using different character encoding formats (UTF8, Unicode, ASCII). A frequently occuring problem are the mutation characters of German, Austrian and Swiss entity names, for example „*Zuerich*“ and „*Zürich*“. Another problem in this scenario is the fact that many entities have their original name while others have a translated version, for example „*Rheinland-Pfalz*“ and „*Rhineland-Palatinate*“ or „Hongkong“ and its equivalent chinese name written in Mandarin. A feasible method to avoid this kind of problem would be to focus on the creation of the ontologies from the databases, as most databases have multiple character encoding support, i.e. a UTF8 and a ASCII name, or an English version for all entity names. If this extended information is not available, it could be a possible solution to generate a translation map, with the purpose to translate entity names into a specific language (i.e. English) to have a constistent naming convention throughout the ontologies.

EntityUnification: In our experiment we relied on the fact that our datasets had an UTF8 version of every name, however it would be also possible to extend the relationmap file (see Chap. 3.1.1) to also translate entity names.

3.3.4. Entities with sparse relations

Sometimes entities have only a few or no relations at all, which makes it difficult to match them to entities with equal name. Another example in this problem category is the case that entities have the same name but completely different relations, i.e. one location has only geographical information while the comparator has only information about people who were born there. This scenario makes it a problem very difficult to merge, as we can not be sure whether the entities are actually the same or completely different.

EntityUnification: Our approach only merges entities if they exceed a critical threshold. Entities with sparse relations still can be merged if the unification phase iterates multiple times, as it can happen that the relations of those entities were merged in previous iterations expanding the sparse information.

3.3.5. Different entitites with similar name and similar relations

As the databases could contain special places, i.e. „Berlin_Central_Station“ or „New_York_City_Madison_Square_Garden“, it is also feasable that those places share a great amount of information as their location. If „Berlin_Central_Station“ has the equal information as „Berlin“, it is possible that both get unified.

A solution which can identify „special“ places could avoid such problems. It would also be a solution to compare the number of words of two entities, as „Berlin_Central_Station“ has more words as the city.

EntityUnification: We handle this problem by comparing the number of words in the names of the entities in our pre check. If the number reaches a specified threshold, which also can be defined in the config file, the pre check fails.

3.3.6. Relations with different name

It is an often occuring problem that a relation in one database has a different name as in another, i.e. „located-in“ and „located“. Also it appears that a specific relation in two different databases is reversed, i.e. „Freiburg located-in Germany“ and „Germany contains Freiburg“. These two relations carry the same meaning but are expressed in a reversed way. A selfmade map could solve this kind of problem as a user could rename relations with different names.

EntityUnification: In our algorithm those relations can be mapped to the same name using the relationmap config file. This config file allows the following three actions:

1. Renaming a relation to have identical relation names:
   a) $f(relationname_{old}) = relationname_{new}$, i.e. „located“ to „located-in“
2. Renaming a relation and reverse its subject and object to have an identical format:
   a) $f(subject, relationname_{old}, object) = object\, relationname_{new}\, subject$, i.e. „Freiburg located-in Germany“ to „Germany contains Freiburg“.
3. Add another relation to generate more information:
   a) $f(subject, relationname_{old}, object) = [subject\, relationname_{old}\, object,\ object\, relationname_{n}$ i.e. „Freiburg located-in Germany“ adds also „Germany contains Freiburg“ to our knowledge base.

3.3.7. Mistakes in the databases

As many entries of the popular ontology databases are mostly made by hand, it is possible that those entries contain mistakes or duplicates with various names. A person who posted an entry into the database might accidently interchange two entities or make a minor spelling mistake at an important attribute. In the unification process it could be a solution to bypass such problems by comparing the ratio between positive and negative comparisons. If an entity has a huge amount of positive comparisons between another and only a single contradiction, it still might be the same entity.

EntityUnification: Through a suitable choice of scores and thresholds, this problem can be reduced as minor mistakes do not directly lead into a false check.

3.4. Unification

After the command line arguments are parsed and the two input databases processed, the unification phase starts to compare the two maps of entities. These maps got created during the file processing phase and consist of the entity ID as the key and an entity pointer as the value. These entity pointer datastructer contains a pointer to the real entitiy and ensures that each entity is stored only once in the main memory as multiple instances of an entity just refer to the entity pointer. The entity datastructure has a relation map with the relation name as the key, i.e. „located-in“ or „has-name“, and an array of entity pointers as the value. This approach makes it very easy to merge two entities, as only the target of the datastructure has to be

redirected to a new real entity with the effect, that also all relations which pointed to that entity pointer will now point to the new target. These entity datastructures are also able to store further useful information, i.e. the number of entities pointing to it or the number of rearrangements.

During the unification phase the algorithm has to accomplish many comparisons and each can have its individual threshold and weight for the decision wether two entities are equal. As for some comparisons it is sufficient that they do not fail, others need to reach a specific threshold value to return a positive result. To handle those complexe decision the unification algorithm is based on scores, which are aggregated and subtracted in the different comparisons. These score values and thresholds are stored in an external config file and can be manipulated and tweaked by the user to enhance the unification result or to set a focus on a specific section of the entities. This config file also makes it possible to tweak the scores without recompiling the program. The possible scores and thresholds which can be redefined by the user including a short description and their standard values can be found in the appendix.

3.4.1. Unification Phase

The unification process starts by having two iterators, one for each map. Those are getting increased depending on the lexicographical comparison between the entity IDs they are pointing to, with the purpose to only compare two entities which have a similar name and not to inspect every possible pair. Instead of doing a complete examination on every pair of entities, as this is a very expensive operation, a pre check first determines whether two entities are likely equal. This pre check is a cheap method and consists of a prefix comparison (first word of the entity name) and a comparison of the number of words of each entity name. The „number-of-words“ comparison tries to eliminate the problems of special places described in chapter 3.3.5. A successfull pre check induces the full entity check which makes a deep comparison between the two entities, considering their full names and all their relations.

The complete check of two entities is started by a successful pre check and makes a deep comparison. It examines the information of both entities and computes a score based on different values and thresholds from the scores config file. If a certain overall threshold is reached, the two entities are considered equal and are getting merged.

The merging procedure unifies the two entities E_1 and E_2 into a single one. Each entity has a relation set R_E, that maps relations that have E as an subject to the set of the corresponding objects:

$R_E = \{(r_i.name,\, f(r_i)) \,:\, r_i \,\epsilon\, relations_{out}(E)\}$,

where r_i is the set of relation targets, i.e. $f(r_i) = \{\, y \,:\, (E,\, y) \,\epsilon\, R_i\}$.

For example:

$R_{E_1} = \{(\text{"}located-in\text{"}, \{E_2, E_3\}), (\text{"}has-latitude\text{"}, \{E_5\}), \ldots\}$

with $E_1.name = Berlin$, $E_2.name = Germany$, $E_3.name = Baden{-}W\ddot{u}rttemberg$, $E_5.name =$ 13.24.

The merging procedure consists of the following steps:

1. Setting a merge flag to indicate that this entity was merged:
   a) $E_1.merged = true$
2. Unifying the relations, such that the *relationmap* of E_1 is the unified set:
   a) $R_{E_1,new} = R_{E_1,old} \cup R_{E_2}$
3. Adding the ID and name of the second entity to the „other-IDs“ array of the first:
   a) $E_1.others\_IDs_{new} = E_1.others\_IDs_{old} \cup E_2.ID.$
4. Reallocating the *entity pointer datastructure* of E_2 to point to E_1:
   a) $Pointer(E_2).target = E_1.$
5. Deleting E_2.

By rearranging the pointer it is guaranteed that all entities that previously pointed to the second entity, now are going to point to the first. After the complete check or a failed pre check one iterator gets incremented until the end of both maps is reached, which ends the unification phase.

To exploit the knowledge of newly connected entities, which are created by the unions of the unification process, the whole unification phase can be executed repeatedly as defined in the config file. Through the further acquired information about the entities, the unification process can do more precise and accurate comparisons in the following rounds as more relations are given. For example the following datasets:

Dataset 1		Dataset 2	
Berlin_123		**Berlin_456**	
relation	value	relation	value
longitude	52.31	located-in	Germany
latitude	13.24	located-in	Berlin
Berlin_789		**Berlin_000**	
relation	value	relation	value
located-in	New Hampshire	longitude	52
located-in	USA	latitude	13
		located-in	Germany

In the first iteration, only Berlin_123 and Berlin_000 are merged, because the other pairs have no suitable relations. Through the merge it would be possible in the second iteration to unify Berlin_123 with Berlin_456, as Berlin_123 now also contains the „located-in“ information.

Another example would be the case that set 1 contains a „Berlin“ and a „Germany“ as well as set 2, with each „Berlin“ located in a different „Germany“. In the first iteration the Germany's are merged, but the Berlin's do not. Through the unification of „Germany“, both „Berlins“ are now located in the same „Germany“ and therefore would get merged in the second iteration.

With the help of the optional debug files, the user is able to review the steps of the unification process as a debug file is created for all entities which passed the pre check and did get merged and one for the entities which passed the pre check but did not get merged. In those files the entities are reported with all their relations and with all the IDs of entities they were merged with.

After the iterations of the unification phase is completed, the algorithm iterates through both maps and generates the turtle syntaxed output file, which has its duplicates eliminated and is lexicographical sorted. Through a config parameter it is possible to highlight unified entities in the output file with a flag.

4. Evaluation

This chapter will present how efficient our algorithm works using some experimental results. All algorithms were implemented in C++ and were performed on a machine with a Intel i5-3570k @ 3.40 GHz processor and 2 GB RAM running Ubuntu 13.04. The program was compiled using gcc version 4.7.3 with -O3 flag and C++0x support. For the evaluation we created two datasets which satisfy the naming conditions. The first is based on the location based data from Geonames [Geo10] and was created with the GeoReader program[1], and the second set is based on the location relevant data from Freebase [BEP+08].

4.1. Results

The following tables shows information regarding the two datasets used in this chapter:

Dataset	Number of Lines	Number of Entities	Number of Triple	Filesize
Geonames	813,489	383,421	813,489	37 MB
Freebase	4,710,584	3,006,213	4,710,584	244 MB

4.1.1. Performance

The following table present the timings and unification number of our algorithm executed with different config parameters. The elapsed average time refers to the unification phase and was calculated using the average of five repetitions. The unification percentage relates to the positive hits of the smaller dataset.

id	Debug	Iterations	Avg. elapsed time	Unification count	Unification percent
1	off	1	15.21 s	161,746	42.18 %
2	off	2	22.68 s	197,500	51.50 %
3	off	3	27.98 s	203,694	53.12 %
4	off	20	64.44 s	205,897	53.69 %
5	on	1	2.22 min	161,746	42.18 %
6	on	2	5.13 min	197,500	51.50 %

[1] http://stromboli.informatik.uni-freiburg.de/student-projects/anton+marius

Timings for the other phases can be found in the following table:

Phase	Average elapsed time
Processing files	32.59 s
Generating output	7.15 s

4.1.2. Quality

For quality evaluation we compared the debug files of our program with the parameters from the performance evaluation with ID 6 to check which entities were and were not merged. As both debug files are tremendous, it was not possible for us to check every comparison. The file containing the entities which were merged has 91,489,868 lines, while the file with the entities which were not merged has 306,056,416 lines.

We compared:

- **popular entities** (i.e. „Freiburg im Breisgau“, „Germany“, „New York City“)
- **popular regions** (i.e. „Freiburg Region“, „Baden-Württemberg“, „New York“)
- **selected entities** (i.e. „Riga“, „Daugavpils“, „Breisach“, „Neustadt“)
- **30 randomly** selected entities which were merged
- **30 randomly** selected entities which were not merged despite a positive pre check

Within this comparisons we reviewed the scores of the entities, their relations and the different names based on the debugging files and identified no direct error. With an unification percentage of over 53.69% of the smaller ontology we achieved a very good result, though it is not perfect yet. Only two not satisfying cases were found, which makes it possible to deeply investigate them:

1. The unification of the city „Freiburg“ and the region „Freiburg“. The region and the city got merged, as they share a lot of equal information in the two datasets: same longitude, latitude and same „located-in“ relations and even have confusing names among the datasets, as in GeoNames „Freiburg“ is the city and „Freiburg Region“ is the region, while in Freebase „Freiburg im Breisgau“ is the city and „Freiburg“ the region.
   This case can be seen as a minor mistake or as an intentionaly unification because a major city and its urban area often share the same name and are often mentioned exchangeable.
2. The lack of the unification of „Rheinland-Pfalz“ and „rhineland-palatinate“. As the one dataset uses native names, the other uses the English alternatives, it is not possible for the pre check to identify equality between those different namings. For future work it would be easily possible to extend the parsing phase of the algorithm with a function to translate entity names.

All other entities which got retrieved and reviewed were correct, contained no misinformation and had no loss of information.

4.2. Potential problems

As the occured problem with „Freiburg“, the city and the region was discovered during the evaluation, more minor mistakes of this kind could possibly appear. If two entities share a lot of equal information and have no contradiction, it is likely that they are getting unified. The major problem of this case is that the Region often has the same geographical information as the city, which leads to a excessive score and an unification.

The other occuring problem is based on the translation of entity names: as a dataset could have translated names, any other could have natives names. This makes it a very difficult problem to find equal entities, as it is not feasible to compare every tuple of entities.

The major problem of those cases is wrong information stored in the datasets. As a completely wrong geographical information of an entity could lead to a failure during the comparisons and therefore the score might not reach the threshold for an unficiation, it could also lead to a not intended unification. Missing information can be identified as the major problem to influence the algorithm, as it can lead to inaccurate comparisons and false unification.

5. Conclusion

In this work we have presented an approach for entity unification with the goal to improve semantic search. That approach's application was demonstrated using the programm EntityUnification. Our work revealed several major problems of the unification process and suggested suitable solutions. We compared different approaches and developed a modular and customizable programm to solve the problem of combining different datasets into a single one. In our experiment with Geonames and Freebase based datasets, we examined the performance and quality of our ideas and pointed out possible problems.

5.1. Future Work

In the following we want to summarize a list of possible improvements:

1. To do a deeper evaluation it is highly recommended to generate a sufficient ground truth and evaluate the precision and recall of the algorithm.
2. With knowledge in the field of automated parameter tuning it would be a great idea to further improve the scores of the config file to achive better results in the unification phase.
3. As the algorithm is fast in this version, it would be applicable to create a more complexe pre and full check.
4. Instead comparing two entities per chunk, it would be possible to gather possibly equal entities into sets and than generate scores inside those sets. This approach would allow to compare the scores within the possible candidates of equal entities.
5. It would be feasable to develop an algorithm in the merging step of the algorithm to not unify the right entity to the left, but rather make it depended on an attribute of the entities, i.e. their size.
6. To avoid the translation errors mentioned in chapter 4.1.2 it would be a possible solution to create a translation map for the file processing phase to transform all entity names to the same convention.
7. To make the algorithm even faster it would be feasible to do the relation comparisons in a multi-threaded enviroment, as those computations can be calculated disjoint as they do not depend on each other.

A. Appendix

A.1. Score description

Name	Explaination	Standard Value
t_number_of_words	Threshold for the maximal word count difference between two entity names which are compared in the pre check	3
t_score_threshold	The score which is needed between two entities for a unification	6
s_entity_name	The score granted for the exactly same entity name	4
s_loc_in_name	The score for two entities which are located in the same location determined by an equal name	2
s_loc_in_prefix	The score for two entities which are located in the same location determined by an equal prefix	1
t_loc_in_ratio_bonus	Threshold for the ratio of the shared locations of two entities to gain a bonus	0.6
s_loc_in_ratio_bonus	Bonus granted if the shared ratio threshold is reached	1
t_geo_max/med/min	Difference allowed between two geographical coordinates to gain maximal/medium/minimal score	0.01/0.2/2.1
s_geo_max/med/min	Maximal/Medium/Minimal score possible through the comparison of geographical coordinates	3/1/0
t_geo_false	Difference between two geographical coordinates which causes a comparison failure	2.1
s_geo_false	Score for getting a geographical coordinates failure	-1
t_geo_both_bonus	Score needed between the sum of the longitude and latitude to gain a bonus	6

Name	Explaination	Standard Value
s_geo_both_bonus	Bonus-score for sharing the equal longitude and latitude	2
t_populati-on_max/med	Threshold for the difference between the population number to gain a max/med score	3500/10000
s_populati-on_max/med	The maximal/medium score gained by being within the population threshold	2/3
t_population_false	Difference between the population of two entities to get a failure comparison	25000
s_population_false	Score for getting a population comparison failure	-1
s_misc_relation_name	Score granted for sharing another relation with the same target name	3

Bibliography

[Ash00] Ashburner, M.: Gene Ontology: Tool for the unification of biology. In: *Nature Genetics* 25 (2000), S. 25–29

[BEP+08] Bollacker, Kurt ; Evans, Colin ; Paritosh, Praveen ; Sturge, Tim ; Taylor, Jamie: Freebase: a collaboratively created graph database for structuring human knowledge. In: *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. New York, NY, USA : ACM, 2008 (SIGMOD '08). – ISBN 978–1–60558–102–6, S. 1247–1250

[Geo10] GeoNames: *GeoNames Geographical Database*. 2010. – Last access on Dez 2008 at: http://www.geonames.org/export

[NM00] Noy, Natalya F. ; Musen, Mark A.: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment, 2000, S. 450–455

[RT12] Rizzo, Giuseppe ; Troncy, Raphaël: NERD: A Framework for Unifying Named Entity Recognition and Disambiguation Extraction Tools. In: *EACL*, 2012, S. 73–76

[SB] Stepan, Anton ; Bethge, Marius. *Relation Extraction: Extracting relations from the GeoNames database*. `http://stromboli.informatik.uni-freiburg.de/student-projects/anton+marius`

[SKW07] Suchanek, Fabian M. ; Kasneci, Gjergji ; Weikum, Gerhard: Yago: A Core of Semantic Knowledge. In: *16th international World Wide Web conference (WWW 2007)*. New York, NY, USA : ACM Press, 2007