Graphs of Mice and Men

Harmen Prins

July 1, 2016

[1]

[1] Credit: Sanna Dinh

Acknowledgements

Abstract

Summary

Contents

1 Introduction

It is a Friday afternoon and you realise you want to watch a movie with some friends this weekend. So, what do you do? To achieve this goal you must take three steps. These steps are also shown in Figure 1.

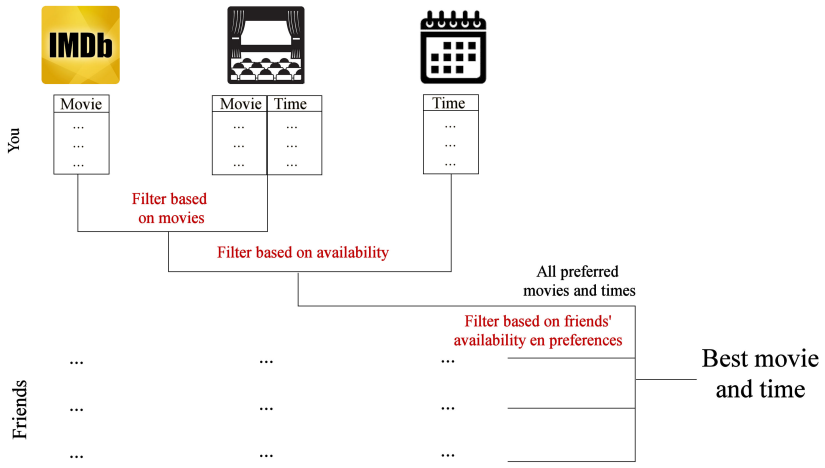

Figure 1: To find the best movie, the shown movies need to be filtered by the ones you like and the showtimes need to be filtered by the times you are available. All remaining showtimes need to be cross-references with the preferences of your friends.

Firstly, search for movies that you like that are also playing this weekend. To do this you must either manually go through the list of movies playing this weekend and check if any movies on that list match your preferences.

Once you have found one or more movies that you like, you have to select a time at which you can see the candidate movies, so you check for every movie every time slot at every cinema near you. The number of time slots is equal to the number of movies you chose in the previous step, times the number of cinemas near you times the number of times every movie is shown at one cinema. You also have to filter the out time slots based on your availability, so you cross-reference the available movie showtimes with your agenda.

Then onto the third and final step, getting friends to go with you. In this step you do not want to burden your friends too much by having them pick both the movie and the time, so you will have to decide which movie to watch or when, even though you do not know if your friends will like this movie or are available at that time. Once you contact your friends, most of them do not respond as they are not online at that time or they do not want to fill out an availability poll or movie choices.

Most of the time people will limit themselves to suboptimal options by avoiding these steps, considering only a few movies, cinemas, time slots and friends. But limitations like this are not necessary. What would happen if the above process was automated?

The first step, finding movies that match two criteria would be considered trivial with modern technology. One finds the two lists of movies that are recommended for you and movies that are playing this weekend and intersect them. A ranking can be added based on how expensive the tickets are, how certain it is that you may like the movie, and other factors.

The second step, cross-referencing the list of movies with cinema play times, would already be harder. Most of the time the cinema play times are written in a human-readable format, something that is often hard to interpret for a computer. Either the play times have to be written in or transformed into a computer-readable format manually or a computer has to interpret the format and convert it into a computer-readable one. However, errors might occur because the local cinema labelled a sequel as "Movie II" whereas the list we have has it labelled as "Movie 2", for example.

Checking an agenda to find an appropriate time would be easy if the times from the cinemas can be transformed to the same format as your agenda. If we are automating the process, we may be able to add more features, like changing standing appointments (of course incurring a penalty to the score of that time spot) and such.

The last step, contacting friends can be done in one click. Let us call the program that automated the first three steps a *softbot* and assume they are universal, i.e. everyone has one. Rather than bothering your friends with scheduling, your softbot can contact the softbots of your friends and find the movie and time slot that optimize a cost function, taking into account which friends you like the most, which movie everybody likes the most and at which time the fewest dinners with wives need to be moved. Once this best time slot is chosen, everyone gets an invite to see the movie and once it is known who will go, a car pooling route is calculated and you can sit back an enjoy your Friday afternoon with no planning or scheduling required.

The most amazing thing about this story, is that it is already possible with the current technology. The algorithms to convert for example cinema website text to a computer-readable format exist. Many open databases already contain a lot of structured data that can be used to reason about data. And algorithms to combine these pieces of information to make decisions already exist and work.

Two things are required before this web of computers that communicate and understand, this Semantic Web, will come into existence. Firstly, the Semantic Web needs to be adopted by humans. They must see the value of these personal assistants that can schedule things for you, that take into account your personal context when searching the web, that understand questions and can ask you for more information that can improve the search. People have to realise that the

current search engines are not good enough, and that having to read through many papers to find the one nugget of information or the one connection that you are looking for is not acceptable or necessary. But also, the different knowledge bases and unstructured texts need to be *aligned*. Aligning means that softbots should be able to combine information that is contained in different domains, documents and databases. This last problem is the one that I will work on in this thesis. Once this problem has been solved, the Semantic Web is technically possible and only needs to be socially accepted before it will become the norm.

1.1 Motivation

In this section I will explain how the study came into existence.

At the 14th International Semantic Web Conference McCallum presented his keynote on *Representation and Reasoning with Universal Schema Embeddings*[**?**].

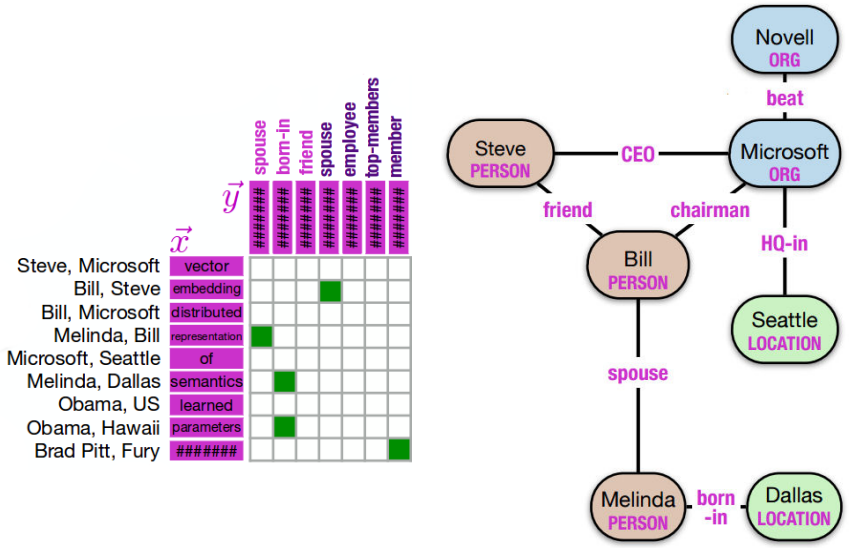

Figure 2: The Universal Schema. Every row and every column is represented by an embedding (in purple). Green cells mean a row and column are in the context of the other. For example, the pair Bill and Microsoft are connected to *chairman*. The embeddings are then trained so that the context can be predicted. That means the embeddings of Bill & Microsoft and *chairman* will become more similar.

The Universal Schema uses two steps. Firstly, the system learns embeddings of entities and relations found in text and Knowledge Bases[**?**], see Figure 2. Secondly, these embeddings are then used to predict new relations for the

knowledge bases. The system finds embeddings such that the embeddings of entities and relations that co-occur are highly similar, see Figure 2.

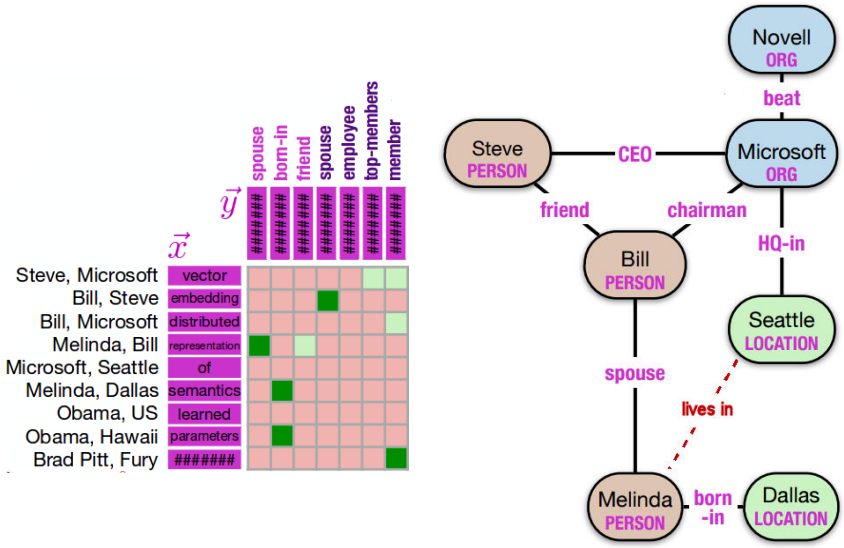

Figure 3: After the embeddings are learned, rows and columns with similar embeddings are connected (in light green). For example, the embeddings of Melinda & Seattle and *lives in* may have become very similar. In that case, the system connects the pair with that relation.

If some entities and relations have similar embeddings but are not known to co-occur, they are flagged as being similar. Similarly, if two entities or two relations have similar embeddings, they are likely referring to the same or similar things, see Figure 3. This can be used to infer even more similarities by collaborative filtering. The technique had very promising results in Knowledge Base completion tasks.

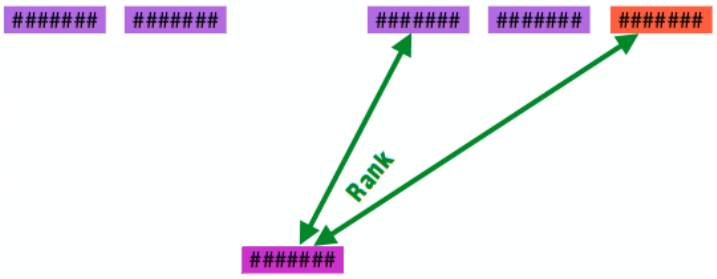

Figure 4: The embedding of a row (in pink) is optimized to be close to its context embeddings (in purple) but far away from other embeddings (in red).

The embeddings are optimized by a model called the skip-gram word embedding model[**?**], a neural network model that learns to predict the context of a word, see Figure 4. McCallum discussed a new extension to the algorithm that allowed for homonyms and as such can deal with more ambiguous data[**?**].

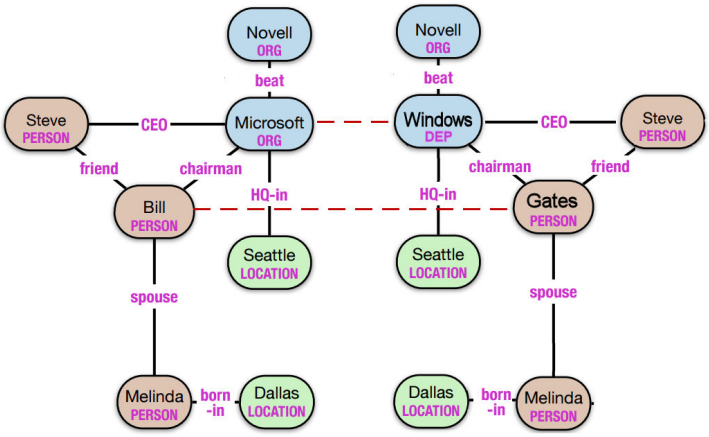

Figure 5: Similar to how the embeddings are used to predict relations within an ontology, ontology matching can be done by predicting relations between two ontologies. For example, since Gates and Bill will have very similar sets of embeddings (as their contexts are very similar) the system will recognise they are the same entity.

McCallum plans on using these embeddings to predict new intra-knowledge base relations. However, from the keynote it is clear that this technique should work with multiple Knowledge Bases as the Universal Schema is applied to a set of Knowledge Bases including Freebase, knowledge extracted from text and the TAC Knowledge Bases Population Track Dataset. This means that inter-knowledge base relations can also be predicted using the Word Embedding Tensor Factorisation technique.

An example of an inter-knowledge base relation is shown in Figure 5. If these relations can be predicted, multiple knowledge bases can be aligned. Doing ontology alignment using these word embeddings is the goal of this work.

Currently, knowledge base and ontology alignment techniques do not make use of embeddings. Since embedding models have good performance on intra-knowledge base relation prediction, they should also work well on ontology alignment.

1.2 Justification

This study is important to the future of the World Wide Web. The next goal of the Web is the Semantic Web, where computers can interact with web pages as humans do. It is necessary to build the Semantic Web as soon as possible,

as the amount of information on the Web is growing so fast it will be infeasible for humans to handle[**?**].

The core of the Semantic Web are Knowledge Bases with the corresponding Ontologies. Ontology Matching is essential for the Semantic Web to work, as the information contained in the Knowledge Bases has to be combined. Since the Semantic Web will have many small knowledge bases, they will be combined at the moment information from them is required. To combine these knowledge bases, their ontologies first need to be aligned. This is an act that will happen very often on the Semantic Web, since every interaction with the Semantic Web will usually require information from multiple domains. Large domains may be stored on multiple devices, which can be viewed as different knowledge bases which also need to be aligned. Therefore, it is save to say that efficient ontology matching will be very valuable.

The most important goal of this study is to find an Ontology Matcher that is able to deal with ambiguous concepts. Since most of the knowledge on the Semantic Web is automatically gathered, it may be unreliable, ambiguous and even wrong. Therefore, ontology matching that is capable of handling this uncertain knowledge is required. Word Embeddings and especially multi-sense Word Embeddings should be able to handle a high amount of ambiguity, since multi-sense Word Embeddings are made to differentiate between different meanings of the same word.

Normally, the raw text representation of words is used for matching, for example to calculate the string distance. However, Word Embeddings Models use a vector representation, which should be more efficient. Calculating the distance between vectors can be done very efficiently, therefore ontology matching with distributed concept representations fits that efficiency criteria.

I believe the algorithms proposed in this study will work on new data without the need for redesigning. This is because distributed representation algorithms can be trained unsupervised on new domains, so they can be applied to those new domains without effort from humans. As opposed to for example the string edit distance. This is also an improvement for the semantic web.

This study should be done now, as ontology matching is already used in areas other than the semantic web. It is, for example, used by companies for merging knowledge from different knowledge bases when merging with other companies. If the proposed matching algorithm is better than the current algorithms, or at least improves the performance when used in ensemble with the current algorithms, it will be useful immediately in these situations, not just in the future, on the Semantic Web.

1.3 Context

This section will enable the reader to put the problem in perspective as well as understand the reasoning behind the solution to the problem. Everything that is mentioned in this section will be explained more in-depth further on. Firstly, I will give a short summary of the context by explaining why the Semantic Web requires Ontology Matching. Then I will illustrate the problem of ambiguity. Lastly, I will describe the algorithm that this study is based on.

On the Semantic Web, Ontologies are used to define knowledge[**?**]. The term ontology will be properly defined later on, but for now it suffices to know that an ontology is a set of rules that determine what relations are allowed and required in a given domain of discourse. As a consequence, if two agents want to communicate in a given domain, they need to agree on the rules of that domain, or in other words, share the ontology of that domain.

However, it is unlikely that a conversation only covers one domain. To solve this, there are two potential solutions. Either you ensure that all possible combinations of domains are covered by an ontology, which means one ontology that covers all domains is required, or, alternatively, ontologies are combined on-the-fly.

The first option, using one single ontology, is infeasible. This is due to the fact that inference and search time scales with Ontology size[**?**], and the number of concepts that are needed on the Semantic Web is enormous. Every single concept that is present in the billions of terabytes of text, images and sounds that are on the web needs to be represented, for every professional domain and science, in every language. And all these concepts are related. Searching through this Ontology just to find one concept would be incredibly costly. And since every interaction on the Semantic Web requires lookups in an Ontology, these lookups need to be fast.

The second option, the one I address in this thesis, is the problem of combining ontologies from multiple domains. Combining in this sense means connecting the related concepts from the different Ontologies. Some research has focused on Ontology Merging, with promising results. However, recently research has been slowing to the point where there is little improvement.

One big problem in Ontology matching is the fact that in different domains, words can have different meanings. Common examples are synonyms like author and writer, homonyms like bank (for money or by a river) and ambiguity. Ambiguity means that similar words mean slightly different things, or a word might have multiple related meanings.

A recently developed model for word embedding may be used to deal with ambiguity. It maps words to a vector space that has semantic properties, and thus can be used to calculate the relatedness of two words. There are a number of extensions that can deal with homonyms as well as data structured other than the continuous bag of words representation that the initial model was developed for. These facts show that it may be possible to make an embedding model for Ontologies and use the word relatedness to combine Ontologies[**?**].

Homonyms are two concepts that share a label, or in other words, a homonym is one word with multiple distinct meanings. As such, homonyms are inherently ambiguous, since the meaning is uncertain without context. This ambiguity is very hard for computers to deal with, as computers are used to dealing with absolute certainties.

1.4 Problem

We have seen that when information from multiple domains needs to be integrated so that it is possible to reason over them together, the ontologies of those domains need to be aligned or merged. If we want to merge two ontologies, we need to find the concepts from the different ontologies that are related by an 'is-same-as' or other rule. However, since the ontologies are made for different domains, the same concept can be represented by different labels in the two ontologies. This problem is called the synonymy problem, as synonyms are two words that refer to the same concepts.

The opposite problem is called the homonymy problem, in which two concepts are represented by the same label. Since they have the same label, naive algorithms might align the concepts as being the same, when they are not.

Then there is the problem of ambiguity, which encompasses many other problems. It encompasses the fact that some concepts are used wrongly by humans, but also the fact that some concepts are extremely similar, and may occur in the same context, but are still slightly different.

The problem is:
"" Matching ontologies and dealing with the ambiguity and homonymy contained in them. ""
I will use distributed word representations to answer the above question, as McCallum suggested in his keynote. Distributed word representations, specifically multi-sense *Word2vec*, should be used for this problem since multi-sense *Word2vec* is specifically designed to combat homonymy while the neural networks in distributed word representations use the context to deal with ambiguity.

1.5 Research question

The main question that I will answer with this work is the following research question:

Can distributed representations of concepts be used to find relationships between concepts in ontologies better than existing matchers?

1.6 Hypothesis

Matchers that use a distributed representation of concepts can find relationships between concepts in ontologies better than the best existing matchers.

To prove this hypothesis, I must proof the following points:

- The algorithm uses distributed representations of concepts.
- The algorithm finds relationships between concepts in ontologies.
- The algorithm has been compared to the currently best matchers.
- The algorithm performs better.

This hypothesis is true for distributed representations matching in general if it is true for one distributed representation matcher. Therefore to test the hypothesis, I must test it for every distinct implementation of the matcher.

1.7 Overview

2 Background

In this section I will discuss all concepts that are required to understand the study. Firstly, the Semantic Web, the key technology that depends very strongly on Ontology matching, will be discussed. Secondly, the concept of the Ontology itself will be addressed. Thirdly, current alignment strategies and their their problems are discussed. Lastly, other related research and technologies will be discussed. Later sections will refer back to the concepts discussed in this section.

2.1 The Semantic Web

The first concept I will discuss is the Semantic Web, also called Web 3.0, a vision of a Web that enhances the User Experience in ways that the current Web can not provide.

2.1.1 Progression of the Web

The original Web connected authors to readers. It allowed authors to create content, mostly static websites. Readers, however, could not change anything, merely read the sites and click on links. In this iteration of the Web there was no interaction. See Figure 6.

Figure 6: The first iteration of the Web consisted of static content, written by authors and read by readers.

Web 2.0, the current Web, models the internet as *hubs* (or sites) with a specific purpose and interaction between users, who can also add content, as in Figure 7. However, there is only limited interconnectivity between those hubs[2]. It is possible to link to content from other hubs using URLs, but this moves the user to another hub, rather than connecting the content present in the hubs.

[2]For example, Facebook is a hub for sharing content with people you consider friends, but if you want to share that content with more people you will have to go to another hub, like Imgur or Pinterest.

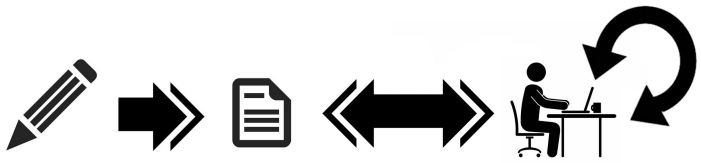

Figure 7: In the second iteration of the Web, readers have become users that can alter and add to the content found on the interactive hubs. This also allows users to interact with each other.

We are currently progressing towards Web 3.0, where not only users interact, but also virtual agents, see Figure 8. The so-called Semantic Web contains sources of *information* which can be combined whenever the user needs it. If someone wants to find information on a certain topic, the virtual agent can collect all documents that are relevant to that topic, summarize each document in a way that is relevant to the search request and provide sources for every fact it finds. It would even be possible to have a question answering session on that specific topic, where the agent finds the answer to every question posed in the material that is linked to the topic.

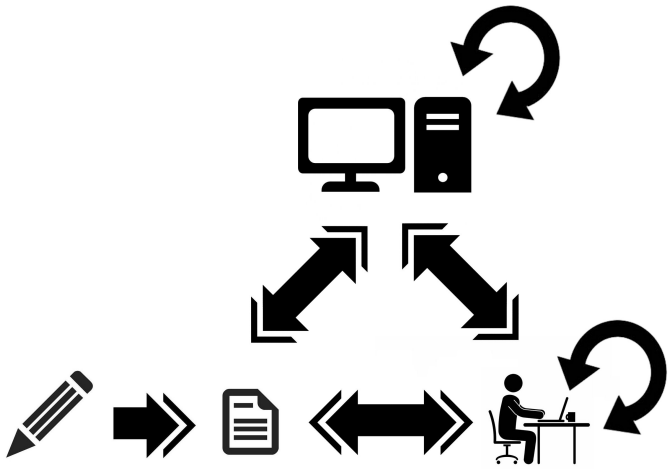

Figure 8: On the third iteration of the Web, virtual agents are also able to read and write content and interact with humans (through question answering as an example). Virtual agents may also work together.

Another possibility on the Semantic Web is the scenario from the introduction (Section 1), where a user wanted to see a movie, and ordered his softbot, to find him a suitable time and movie to see. This was shown to reduce a

lot of planning and allow for much more favourable decisions since many more variables can be taken into account.

All of this is possible if the softbots can access all of the information reliably. In the current Web, this is not possible, as all information is written in an ambiguous, unstructured format called natural language. To allow softbots to access the same information that we can, the Semantic Web proposes to add a layer to the internet where all information is saved in a unified format that is unambiguous, robot-interpretable and can be used to store any type of knowledge.

The Semantic Web also improves user interactivity by connecting sources of information. To stick with the social media example, a person can create a piece of content, say a picture of a tree, and share it with his friends. One of his friends then can share this piece of content with a group of people that likes nature pictures. However, everyone can access all references someone makes. So if a person from the nature group notices the tree is ill and comments on this, the original poster can see this comment and act accordingly. On the current web the friend who shared the picture with the group has to be contacted directly by the commenter and then has to relay the message manually.

2.1.2 Goal

The goal of the Semantic Web is very pragmatic: help people in everyday activities by leveraging all information available on the internet. The fact that it is pragmatic goes a long way of making it a reality: if even one application comes into existence that uses online information in a structured way and thus makes everyday tasks a little easier, the goal is accomplished. Of course, then the goal is stretched and we should make more applications leveraging more information for more tasks.

Ultimately, the goal is to leverage all information that is available to an entity on the internet to improve the lives of humans. This information is not limited to the information that is online *now*, but can include sensor data from the Internet of Things, data from robots, facts deduced or statistically inferred from existing data and so on. All this information can be used to accommodate the wants and needs of humans, when asked for it *and* before the users are aware of their needs. The personal agent should be able to predict the users need and provide the tools that can satisfy the need.

To provide these tools to the user, an agent uses many different sources. For example, if the need of a researcher is knowledge on a certain topic, the agent must chart relevant topics, map relations between different domains and connect these new ideas to topics that the researcher is already familiar with. In fact, the largest part of the job an agent has to carry out on the Semantic Web is

finding relationships between pieces of information on the Web. Based on this observation, we can state that Ontology and Knowledge Base Alignment is the cornerstone of the Semantic Web, since without this vital system the Web is merely a collection of separate pieces of information. Only when those pieces are aligned can we talk of knowledge, of semantics.

To summarize, the goal of the Semantic Web is to connect all online information to help humans. Only when all information can be connected will it be possible to aid humans in their needs. But humans will only accept the Semantic Web if it is useful to them, so the Semantic Web must show its applications and usefulness before it will be used in daily life.

2.1.3 Workings

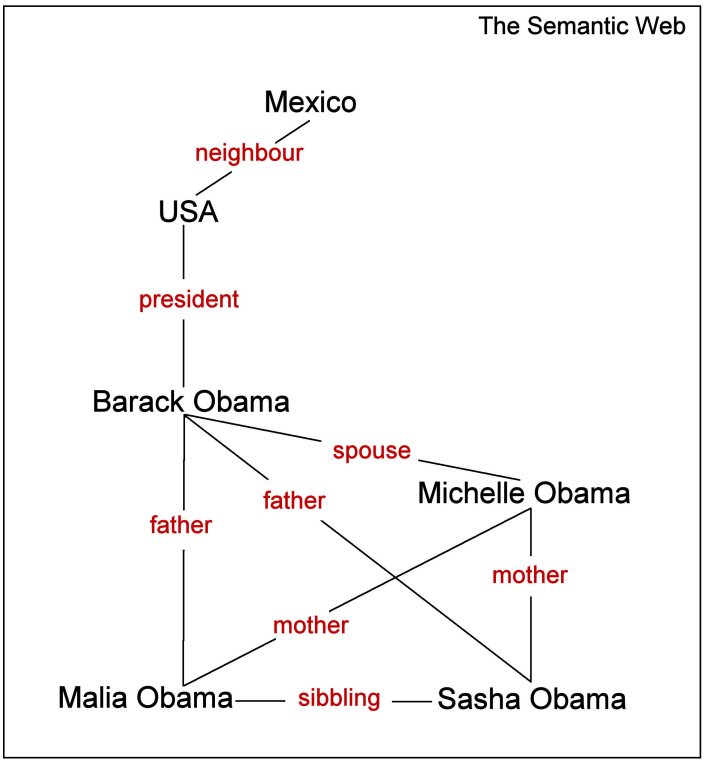

Figure 9: An example of the Semantic Web represented as a graph of entities (in black) and relationships (in red). It contains many different kinds of entities, including countries and people.

The Semantic Web is one big graph of entities and relationships between those entities, containing information about, amongst others, people and countries as in Figure 9.

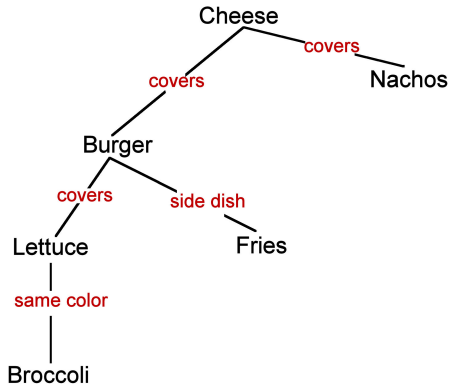

Figure 10: New information to be added to the Semantic Web.

If new information is added to the Semantic Web, a new concept or connection is added to this graph. In Figure 10 a whole set of related entities and their connections are shown. These entities are also connected to the rest of the Semantic Web graph so that it stays one connected web, as in Figure 11.

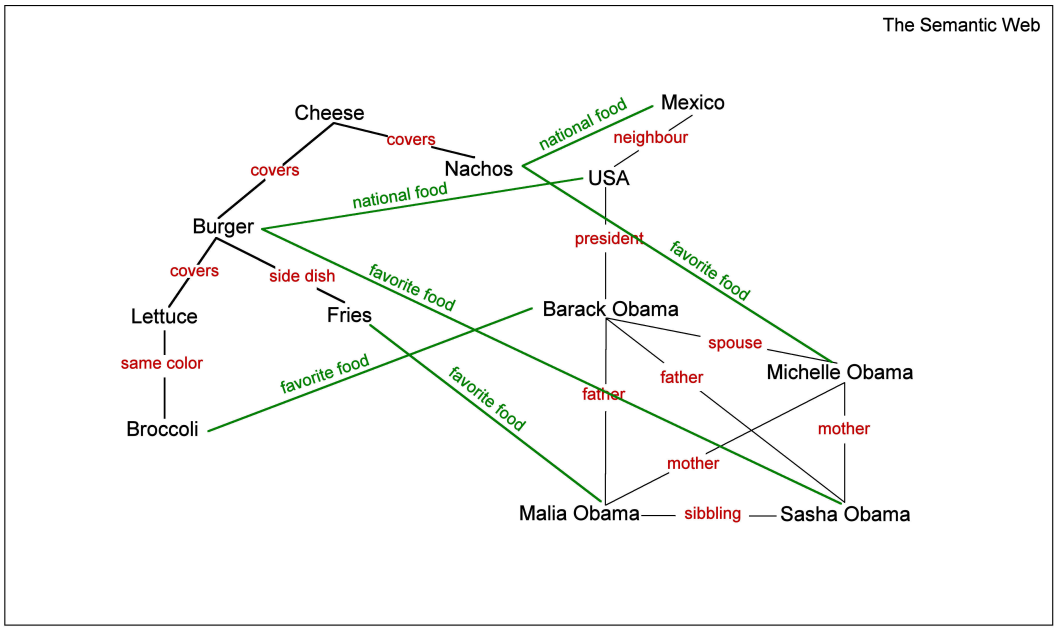

Figure 11: The new information is connected to the rest of the Semantic Web by connecting the new entities with entities in the Semantic Web.

This big graph covers many different domains, and every domain is covered by a part of the graph, a sub-graph. For example, all people are in the same sub-graph, as are countries and the recently added foods. Every such sub-graph is one Knowledge Base that covers that specific domain. See Figure 12 for the Knowledge Bases in the example.

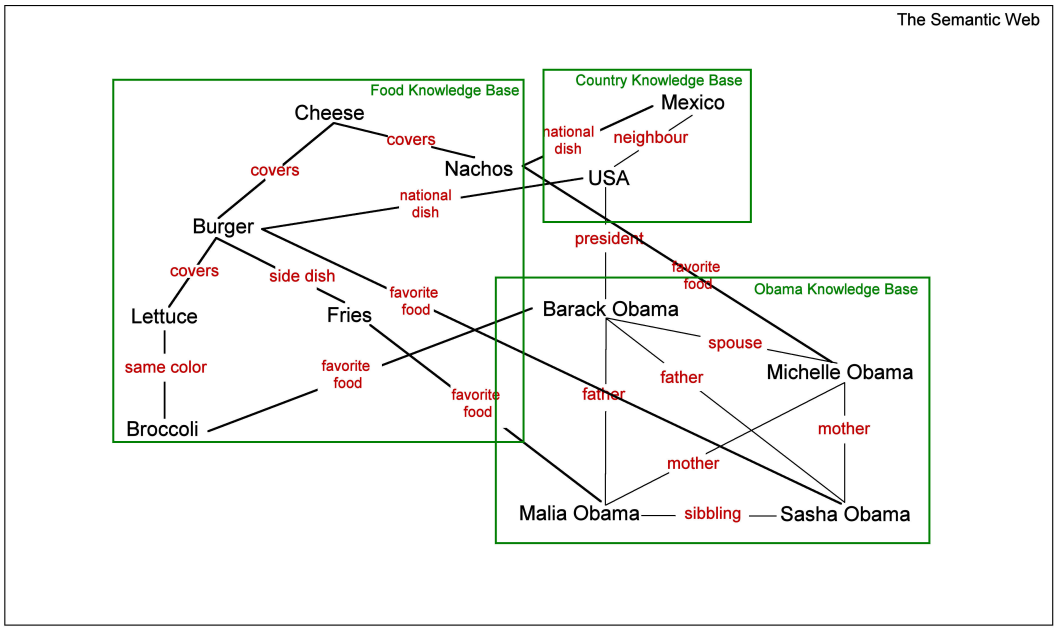

Figure 12: The different sub-graphs in the Semantic Web graph. They are contained in different Knowledge Bases and connected by URIs.

To ensure that the Semantic Web is one big connected Web, the knowledge bases are connected by inter-knowledge base relations. The relations are made by referring to the URI of that entity. The URI consists of a link to the knowledge base and then the identifier of the entity. For example, the people are connected to their country of origin, so in the people Knowledge Base there is a reference from Obama to *countries#USA*.

In this paragraph I will explain the use of ontologies on the Semantic Web. Ontologies describe the rules by which the Knowledge Bases must play. For example, every food must have a country of origin, and those countries of origin need to be of the Country *Class*. This rule enables search algorithms to limit search to the countries Knowledge Base, as it is the only Knowledge Base that contains countries[**?**]. This is just one example of a constraint that improves the work of one system. There are many more reasons, for example adding the rule "all mothers are parents" allows inference systems to reason about richer information without having to explicitly write down every relationship.

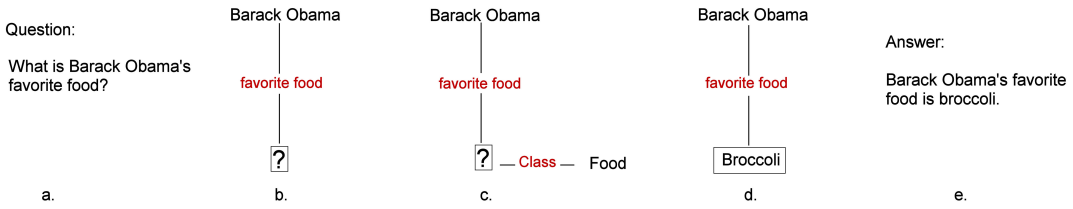

Figure 13: The question posed in a) is converted to a query pattern b). Ontology information is added in c). The result is found after pattern matching in d) and then converted to a natural language answer in e).

There are many ways of interacting with the Semantic Web, but they all interact with the graph in some way. For example, a user can ask a question "What is Obama's favorite food?" Firstly, this question is converted into a format that can be used by a search system. Then, information from the ontologies is added so that the search can be performed more efficiently. Then the pattern is matched in the Semantic Web graph, listing all results. The results are then combined or filtered such that one answer pattern remains, and that pattern is converted to natural text. This process is depicted in Figure 13.

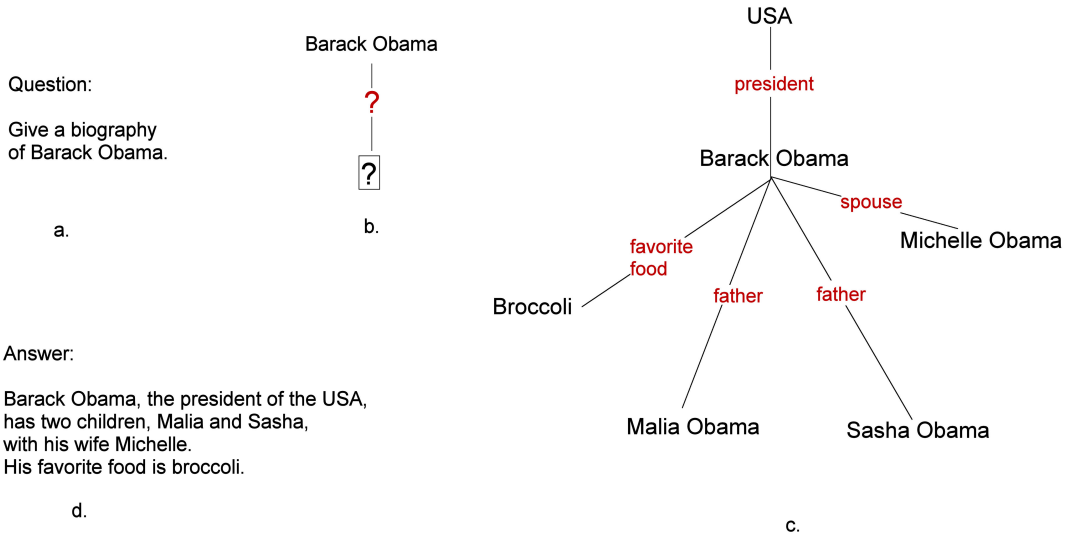

Figure 14: Similarly to a question with a single answer, the question in a) is converted to the pattern in b). The answers that match b) are combined in c) and converted to natural language in d).

More complex question can be posed, although they do not need to be longer. If the question can be mapped to more parts of the Semantic Web, the answer

will be longer. In this case all knowledge of Barack Obama is extracted, combined and converted into an answer.

2.1.4 Current state

Currently, people are not aware of the Semantic Web. Either websites and applications do not use any semantic information, or it is used on the back-end, hidden with other complex programs that users will not understand. So what is the Semantic Web used for nowadays, if at all? In this section, I will list a number of general ways people use Semantic Web technologies as well as specific applications. Note that these are Semantic Web technologies, i.e. technologies that came into existence through Semantic Web research and can eventually be used in the Semantic Web, but currently are not used for the Semantic Web since it does not exist yet.

Semantic Web systems are currently divided into two groups: the systems that need high accuracy and the ones that need high recall. The first group are deployed in domains that require exact knowledge, whereas the second group are systems that must make available a big amount of data. In the past it was not possible to extract information from texts and other sources automatically, so all systems were hand-crafted and thus highly accurate but very small. This also impacted research, which did not focus on scaling ontologies until recently.

The group that requires high recall use hand-crafted knowledge bases, created by experts, containing only facts that are accurate, verified and relevant[**?**]. This method of creating a knowledge base does not scale very well and usually consists of fewer than ten thousand facts. Because of this, the knowledge bases are restricted to small domains, including some parts of medicine and cultural heritage[**?**, **?**].

For the group that requires a high recall, or in other words a big amount of data, automated methods are required. These methods work fast, parsing thousands of websites a minute, but are inclined to make mistakes. Often, these methods use crowd-sourcing to improve accuracy, using experts or the general public to suggest or check facts[**?**].

On way the W3C is fighting ambiguity is to ensure very few different languages are used for ontologies and knowledge bases. The main languages W3C endorses are RDF, OWL and SKOS[**?**, **?**, **?**].

Large ontologies already exist, with the largest, Freebase, boasting almost two billion triples and the three hundred combined Large Open Data databases contain over thirty billion[**?**, **?**]. The LOD is simply a single alignment over many different knowledge bases.

Applications include improved web searching, question answering, product comparison, context merging, data integration, decision support, translation, all the way up to the intelligent softbot mentioned earlier. [**?**, **?**]

The many uses of ontology matching show that it is important to research all avenues that might improve ontology matching, as it will have a direct result in many other fields.

2.1.5 Technologies

2.2 Ontologies

Earlier, ontologies were referred to as 'a set of rules that determine what relations are allowed and required.' Now, the term will be defined more extensively, the standard format will be described and the uses of ontologies will be discussed.

An ontology describes concepts as their relationships to other concepts. The concepts that are described determine the domain the ontology covers, and the relationships determine the rules the ontology imposes on the domain. For example, a hierarchy is an ontology that describes concepts that are subclasses of other concepts. Therefore the relationships are 'is-a' relationships: a bird *is an* animal. Concepts can be instantiated, which is done in a Knowledge Base (KB). KBs and Ontologies are often confused, and are indeed very similar in structure. The Knowledge Base in our example can contain actually existing birds or fictional birds, and refers to the Ontology concept of bird to embed the instantiations with meaning[**?**].

2.2.1 Example

In this section I will give an example of a pair of anatomical ontologies with their alignment. This example will be used throughout this work.

A mouse consists of the following body parts:

- Head
- Torso
- Whiskers
- Tail
- Front paws
- Hind paws

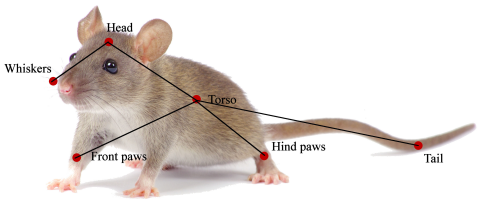

Figure 15: The different body parts of the mouse are connected as in this image.

Humans consist of:

- Head
- Torso
- Moustache
- Arms
- Legs

Figure 16: The different body parts of the man are connected as in this image.

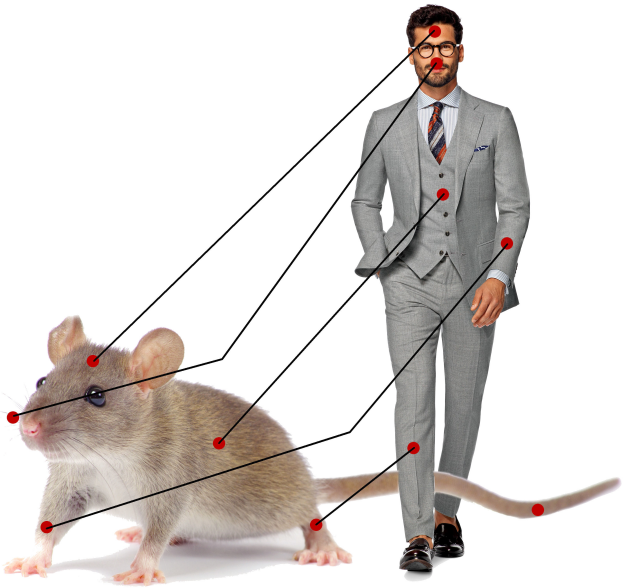

Figure 17: Humans and mice are connected as shown in this image.

2.2.2 OWL

OWL is a family of languages and syntaxes that can be used to create an ontology. The different languages are designed around requirements of possible relationships and concept definitions, and thus may differ a lot between them. The W3C has defined three variants which are the bases for all other adaptations. The variants trade off levels of expressiveness versus computability.

A relationship in OWL is written as follows:

```
<owl:Class rdf:about="http://mouse.owl#Whiskers">
  <rdfs:connectedTo rdf:resource="http://mouse.owl#Head"/>
</owl:Class>
```

2.2.3 Uses of Ontologies

There are three main reasons Ontologies are used. Firstly, it allows querying languages to optimize the search process. Secondly, it allows expansion of information on a node. Thirdly, it allows for consistency checking a Knowledge Base. Lastly, it allows merging of knowledge bases.

Search can be optimized by taking into account constraints the ontology provides. For example, no professor is a student, therefore no professor can be connected to a course with 'follows' predicate. This allows a search algorithm to skip all professors when looking for people that might follow a certain course.

Expanding the available information of a certain node can be done by taking into account *positive* constraints of a concept. For example, since all birds have beaks, it would be inefficient to store this fact for every instance of bird in the knowledge base. However, if this is a constraint given in the ontology, it can be accessed for all instantiations of the bird concept with less space required. Superclassing enables ontologies to store these types of information even more efficiently, allowing for enormous amounts of information to be extracted for every node without the need to store it all explicitly for that node.

One important aspect of data bases is consistency. Manipulations on a data base such as a knowledge base must not result in a knowledge base that does not adhere to the data base constraints. In a knowledge base, these constraints are defined in the ontology.

When two knowledge bases need to be merged, aligned or matched, instantiations that refer to the same thing need to be merged. For example, if two person databases both contain references to the same person, all information on that person needs to be linked to the same object, such that information of that person from both knowledge bases can be combined. Merging the ontologies of the two knowledge bases vastly improves the alignment between the

two knowledge bases. However, ontology and knowledge base merging is not a trivial problem and actually the problem that this thesis addresses. Therefore it will be explained more in-depth later on[**?**].

2.2.4 Aristotle and Plato

All ontologies lie on a spectrum of formality. On the one end of the spectrum are the Platonic ontologies, which should only contain concepts that are perfectly defined and constraints which are completely binding. The name refers to the Platonic philosophy that all concepts are existing entities and can thus be perfectly captured in a definition.

On the other end of the spectrum are the Aristotelian ontologies, which do not necessarily contain perfectly defined concepts, but rather concepts as we observe them. Aristotle disagreed with Plato's theory that concepts are existing things, and thus these ontologies are named after him[**?**].

Aristotelian ontologies have the advantage of being easier to create. One could use statistical methods given a sample of all possible observations or crowd sourcing to create concepts and constraints. This would enable enormous ontologies and knowledge bases with the amount of data currently available and cheap mental labour with services like Amazon's Mechanical Turk.

The advantage of a Platonic ontology would be the fact that every query would result in a fact, since the ontology itself is perfect. The disadvantage is that creating such an ontology is much more costly than an Aristotelian ontology, since every concept and constraint needs to be correct in all cases.

Another way to represent this spectrum is the precision-recall trade-off. A Platonic ontology has maximum precision but its recall is limited since it can only represent very little if creation budget is limited, or paradoxes exist. The Aristotelian ontology on the other hand, can theoretically reach perfect recall given enough storage size and observations for its statistical methods. Again, in reality this is limited by a budget. All ontologies fall between these two extremes. Often, a combination of statistical methods with expert confirmation is used.

Platonic ontologies are often used in areas where precision is important, like medicine, where lives depend on the information contained in the ontology and its knowledge base. Aristotelian ontologies are more common on the Semantic Web, where massive amounts of data need to be represented and queried.

2.3 Current alignment strategies

Many different alignment strategies have already been developed. All strategies follow the same two-step approach. The two steps are independent and as such

different methods can be used interchangeably. The first step is to generate an initial correspondence set, where correspondences between nodes are found based on just their labels and meta-data. The second step is to use this initial set and the structure of the ontology to find more correspondences. These steps are called the terminological and the structural steps. Some strategies also use extra steps like the extensional and semantic steps, which use vector space models and inference, respectively[**?**].

Popular terminological strategies are WordNet comparison and edit distance. The former uses the popular WordNet hierarchy as a distance measure between two concepts, e.g. the number of edges between the concepts and their least general common superconcept[**?**]. For example, the distance between moustache and whiskers is 3, since their least general common superconcept is hair the distance between moustache and hair is 2 (with facial hair between them) and the distance between whisker and hair is 1 (direct). Edit distance is a purely string-based similarity measure. The similarity is a (weighted) count of edits required to transform one word into another, where common edits are insertion, deletion and replacement. Similar methods include substring matching and n-gram matching, which compare parts of the strings to find similarities[**?**, **?**]. For example, arm and front paw have an edit distance of 8, since they have 1 character in sequence in common (either the a or the r) and thus 8 characters that are different.

2.3.1 Problems in ontology alignment

How should the wildly differing demands the applications place on the ontology matching systems be satisfied by a single or just a few systems? When merging ontologies on one end of the Aristotle-Plato spectrum, often a completely different approach is required than an ontology on the other side. Sometimes the matching needs to be very fast, for example when the ontologies are used in a search query which the user expects to be done in milliseconds.

Online matching would allow for updates in a knowledge base to be represented in the alignments of that knowledge base. This would be required on any ontology that tracks changing knowledge, like stores, movie theatres and social media, for example.

Based on the previous two points, we can state that it would also be useful to have different benchmarks to test those different types of challenges and matching systems. This would allow for better comparison of methods and better tracking of the improvements in the field.

Since the Linked Open Data keeps growing, it, and other resources, should be used in matching. This is already done to some degree, as discussed in

the WordNet matching strategy section 2.3. The challenge is to all available ontologies and other sources rather than just WordNet.

Matchers should be able to explain the results it produces, i.e. making those results more interpretable. This would allow a number of other improvements to occur. Firstly, it would allow users of a system to have more confidence in the results and make better decisions based on the source of the result. Secondly, it would allow people manually aiding the alignment to check if the matcher did a good job, and thus improve the interaction between the system and the domain expert. Thirdly, it would allow researchers to improve the matching system because they are able to detect the source of errors.

When these problems are solved, ontology matching has matured enough to be used in Semantic Web applications. Then it is merely the challenge to have the public adopt these technologies.

2.4 Word representation

An important area of research in Natural Language Process is the representation of words. The most obvious representation is the collection of characters humans are used to. However, this is not useful for computers as these characters do not say anything about the meaning of the words, nor is it a representation that computers can quickly do calculations on.

Many different representations have been proposed. The main purpose of word representations in NLP is to represent the meaning of the word. One way to represent a word is the term frequency. This representation counts the number of occurrences of a word per document in a set of documents. The resulting vector of occurrence counts is then used to represent the words. Since the documents have a subject, there is some meaning embedded in the vector representation. However, the representation is very sparse and the semantics or meaning of the word are not captured very well. Also, the method requires many discrete documents to train on, which are not available in Ontology matching.

The rest of this subsection will describe the latest word representation model which has very interesting results in NLP. It will cover the history and the most recent advances that are relevant to this study.

2.4.1 Skip-gram models

Skip-gram models have been in use for some time. Like the document frequency representation, it represents words as vectors. However, the vectors are much more dense and can be trained on sentences rather than documents. The vector distance of skip-gram models has been shown to be related to the semantic similarity between words. This semantic similarity property is very useful for

ontology alignment, as parts of one ontology have to be aligned with parts of another ontology depending on their semantic relation.

Skip-gram models work by teaching a 3-layer neural network to predict the context of a word. In NLP the context is a number of words before and after the word. Firstly, all words a represented by 1-hot encoding, which is a vector with all zeroes and a single one on the index of the word. This vector has length V equal to the number of words in the corpus. This vector represents the first layer, the input layer of the neural network. The second layer, the hidden layer is obtained by multiplying the first weight matrix with the first layer. Since the first layer is one-hot encoded, this is equal to the row of the matrix corresponding to the index of the word. This representation will later be used as the representation of the word.

Then the representation is multiplied with another matrix to create the third layer, the output layer. The word representations and second matrix are optimized in such a way that the output layer is as close as possible to the one-hot encodings of the context.

As a consequence words that occur in similar contexts will be represented similarly. This again causes contexts that contain similarly represented words to also become more similar. This positive feedback loop causes co-occurring words to be represented more and more similarly. On convergence, words with similar meanings are represented similarly.

2.4.2 The state of the art

Recently, an effective and efficient skip-gram model has been developed, called *Word2vec*, which uses a number of extensions over previous methods that enable it to efficiently learn an effective representation[**?**]. One example of such an extension is negative sampling, which is a step that teaches the network that randomly selected words should have non-similar representations.

These extensions enable *Word2vec* to be trained on large corpora and represent words in a way that is very useful in NLP. It is used as a language model in part-of-speech tagging and machine translation.

Interestingly, the representations have been shown to contain semantic information too. A common example is the following formula: $vec(\text{"queen"}) = vec(\text{"}king\text{"}) - vec(\text{"}man\text{"}) + vec(\text{"}woman\text{"})$. This means that gender is encoded in the vectors and other types of meaning may also be encoded.

After the authors showed the usefulness of *Word2vec*, many researchers developed new uses for the algorithm. Some extensions that are relevant to this study will be discussed. Firstly, I will describe a model that represents syntax

as opposed to semantics. Secondly, I will describe a model that is capable of representing multiple meanings per words.

2.4.3 Word order

Word2vec has been expanded in a number of different ways. For example, word order can be preserved, leading to a similarity measure that is closer to syntax, as syntax defines positional properties of words within a sentence[**?**]. To enable this alternative representation, the model had to be changed structurally. Rather than a single output layer generated by a single matrix, one layer per word in the context was used. As such, the model learns to predict the context words based on their position relative to the source word. More important than the fact that these models can be used for syntax is the fact that the research shows that the input does not have to be a continuous bag-of-words representation. This sparked the idea that it may also be possible to have a graph as the input to the network. This assumption is the basis of this study and will be tested thoroughly.

2.4.4 Multi-sense

Another interesting development is the multi-sense *Word2vec*, which allows for multiple vector representations per word, depending on the number of different definitions a word has. For example, the word bank represents both the monetary institute and the riverside, which would both have a different representation in this extension. It can differentiate between two different meanings based on the context. The importance of allowing for multiple meanings is that between multiple ontologies, the same word can have different meanings and as such should not be matched. On the other hand, a concept in a single ontology can have multiple meanings and should therefore be matched with more than one concept from another ontology[**?**]. This idea sparked the second reason why I started investigating semantic word representations for Ontology matching, since current methods do not allow for disambiguation.

For example, string matchers only look at the phrases representing the concepts, and as such do not care about semantics. Structural matchers use the context, but do not use the meaning of the concept since they only look at the location of the concept within the graph. Lastly, lexical matchers do use synonyms and antonyms to see which different words could refer to the same concept, but if two concepts can be represented by one word. As such, initially the multi-sense representation seemed like a very strong candidate for improving matching results.

2.5 Used technology

2.5.1 Factorie

Factorie is a toolkit that contains a number of tools which includes word-embedding models [**?**]. [**?**].

Factorie provides a number of utilities for writing an algorithm in the shape of abstract classes with existing functionality, namely IO, parallelisation, quick parsing and command line argument parsing. It also provides example implementations and programs. Factorie is written in scala, a language that has been gaining popularity recently and interfaces with java. See [**?**].

The automated IO allows for quick reading in of the data set and secure saving of the trained model. The file reading is quicker than a naive CSV parser would allow for, and thus enabled quicker training of the models. This meant less time is used and more time can be used for experimenting and improving the algorithm. Saving the different models allows for model comparison after all models have been trained, which means new investigations can be performed after conclusions have been drawn from earlier investigations, without having to retrain all models. This also allows multiple models to be trained during a period where the experimenter is absent, after which he can still manually investigate the models.

Parallelisation can massively improve training speed. It allows an algorithm to update its model for multiple learning instances at the same time, thus reducing the time needed by the number of instances it can process simultaneously. The number of parallel processes differs per device but modern personal computers already boast 8 parallel processors.

Since I investigate many different implementations of the same algorithm, it should be easy to switch between these implementations at run-time without much effort. Command line parsing allows for this to happen, and thus improves experimentation ease.

2.5.2 Jena

Jena is the most used library for storing, manipulating, querying and reasoning on ontologies[**?**]. It has an interface that allows for easy access to an ontology. For example, if one would like to obtain a list of all neighbours of a certain node, one can simply use the following Jena command:

```
ontology.listStatements(givenNode, null, null)
```

It automatically gives all statements that have the given node as subject. In terms of graphs this is equal to obtaining all edges that go out of the given node.

In Jena it is possible to obtain the label of any node, if it has it, by calling:

```
label = node.isBlank() ?
                node.getBlankNodeLabel() :
                node.getLiteral()
```

The ternary is to distinguish between blank and non-blank nodes. Blank nodes are a feature of ontologies and thus need to be taken into account. Blank nodes can be used to represent concepts with multiple representations, for example a label, a description and a known synonym. More advanced algorithms can take into account all these representations. Another use of a blank node is representing a complex relationship that cannot be represented by an object-predicate-subject triple. For example, if the relationship has a certain confidence or more than two concepts are involved.

3 Method

This section contains the description of the matching system and experimental setup. It consists of four parts. Firstly, it describes the idea and challenges of implementing the system. Secondly, it describes the data used to test the system. The third part describes all different implementations of the system. The fourth and last part describes the evaluation criteria.

3.1 Idea

The goal of any ontology matcher is to find an alignment between two ontologies that is as similar as possible to the true alignment. Such an alignment consists of node pairs where the two nodes in each pair come from the two different ontologies. Sometimes a label is added to each pair, indicating the relationship the two nodes have. For example, 'part-of', 'similar-to', and such. However, we only consider equality relationships and thus the label can be omitted.

Then the question becomes: *how do we measure similarity of concepts between ontologies?* As shown in 2.3, attempts have been made to model similarity of nodes as the string similarity of their labels. Also, WordNet is used to find similar nodes. The string similarity assumes that similar labels refer to similar concepts. While this is true to a certain degree, homonyms, synonyms, and other phenomena make this assumption very weak. WordNet-based similarity measures are better since WordNet links concepts semantically, and thus similar concepts will be close in WordNet. The major problem with WordNet is that it is constructed by hand, and thus does not scale very well to new domains.

The idea behind the algorithm introduced in this study is to find an alternative to WordNet that is at least as good in modelling semantic distance but scales to new domains. It should be trainable on this new domain through existing ontologies and texts in an unsupervised way so that it does not require the interference of experts. This will allow it to overcome the weakness of WordNet. This is only relevant, though, if the performance is equal to WordNet-based matchers.

Word2vec is an unsupervised method that implicitly maps words to a vector space that has been shown to contain semantic properties. Explicitly, it predicts words from the context of that word. It can be trained on large corpora quickly and the resulting vectors can easily be used to measure the distance between words. All these properties make it an excellent candidate for a concept distance measure to use for ontology matching. As an added benefit, it can also take into account the context of a concept, which may be able to improve its performance over a matcher that only uses the label of a concept.

3.2 Challenges

The semantic representation will be used for finding nodes in the two ontology graphs that are similar, i.e. close in the vector space. This step is relatively straightforward if proper vector representations are found. However, this may be hard due to a number of problems which I have listed below.

- The problem I am trying to solve is the problem of differently labelled nodes referring to the same concept. For example *writer* and *author* will not be matched by a string matcher, but should be matched. The two concepts may also have different labels in their context even though their contexts refer to the same concepts.
- The size of the training corpus should be large enough for proper representation to be learned for every concept. It should also be relevant to the labels that are in the ontologies. This is a problem since the ontologies may be relatively small, so they need to be extended while keeping relevance.
- Some labels may be ambiguous, for example homonyms, which are concepts that have the same label but represent a different context.

To solve these problems I will adapt *Word2vec* to graphs to create an algorithm that converts nodes to vectors, in other words *Node2vec*. It can be extended with multi-sense embeddings, different ways of considering the neighbourhood and hot-starting.

3.3 Data

The data used comes from a list of data sets that the OAEI provides[**?**]. Every data set contains two ontologies that need to be aligned and a ground truth alignment. The goal is straightforward: find the alignment given only the ontologies and whichever outside resources are needed.

3.3.1 Preprocessing

3.3.2 Mice and men

The first data set considered consists of two anatomical ontologies. The first ontology describes the anatomy of mice and the second ontology the anatomy of men. Since they are similar creatures, there are many correspondences between the anatomy. The ontologies contain X and Y nodes, and X and Y edges, respectively. There are Z correspondences between the two ontologies.

The ontology is structured as a hierarchical tree with the only label being *subClassOf*. This poses a few problems for some of the implementations as they take into account the edge labels. Since there are no edge labels, these implementations will not be different from their edge-less counterparts.

3.3.3 Medical

The second data set is larger, boasting X and Y medical concepts with X and Y relations. There are Z correspondences between the two ontologies. This data set has N different edge labels.

3.4 Considered implementations

In this section, I will discuss the different implementations that were made of the algorithm. All implementations use the same basic algorithm in different ways by taking into account the context in different ways. All of these implementations were tested and the results can be found in Section 4. Some implementations extend others, but all implementations are covered to ensure completeness. This way, all improvements are recorded precisely. The explanations refer back to the example in Section 2.2.1.

3.4.1 Node vectorisation

The most basic *Node2vec* model converts a node to a vector purely based on its own label. This method has one advantage over simple string matching: if no node is found in the other ontology that matches exactly, we can still find a node that is similar since it is close in the vector space. Therefore, this method should already be an improvement over the most basic string matching algorithms. It may even be competitive with more advanced string matchers (that look at substrings of labels). For example, *whisker* and *moustache* will be matched by a substring matcher, but are also semantically similar since they will often be mentioned in the same context.

3.4.2 Combine with neighbours

To improve the model, we can add context information from the neighbours of a node. The most basic context adaptation would be to create a vector as the average of the context vectors. The resulting vector should be combined with the node vector of the node that is being investigated. This can be done by averaging or weighted averaging where the context is weighed more if there are more neighbours, though not necessarily linearly. These possibilities were also investigated. For example, since the *front paws* and *arms* are both attached to the *torso*, their context vectors will be similar as well.

3.4.3 Select from senses

Another way to take into account the influence of neighbouring nodes is to train a multi-sense node embedding model as described in Section 2.4.4, and selecting from the different node senses based on which is closest to the context average.

3.4.4 Adding edge labels

To add more information to the model, edge labels can also be added to the context of a node. This method effectively doubles the training data and context size when creating the context average. This should make the model more robust, although edge labels may be duplicate (one has many unique family members) and less informative in general, so a lower weight may be appropriate. A different possible problem with this method is that the relation between a neighbour node and its corresponding edge may be lost, since they are just treated as independent contexts. In the case of the mouse Ontology, the *head* is above the *torso* so *isAbove* will be added to the context of *head*. The same goes for the *human head*, which will make them more similar.

3.4.5 Select from senses

This method is exactly the same as the selecting from node senses with just the neighbour nodes except that you now also take into account the edge labels.

3.4.6 Combining edges with nodes

As can be read in Section 2.4.3, it is possible to drop the bag-of-words assumption that the context is sequence invariant. This means we can for example have the context be (previous word, next word) and those words will be treated differently. Similarly we can separate the edge and node labels and treat them differently. This ensures that the model will find any relationship between the edge and the node if it exists and will take this into account. For example, *isBelow* and *torso* will combine into a vector that is similar to *tail* whereas *isAbove* and *torso* combine into a vector that is similar to *head* since the edge and node labels are combined in the model, rather than considered separately.

3.4.7 Hot start

It is hard to classify the proposed algorithm in terms of the conventional ontology alignment method classes. It is not purely string matching, since it takes into account information from the context. However, it is also not structural, as it can work without a seed alignment and uses information other than the structure as well. It is not semantic (or logical) since it does not use inference. Nor is it terminological, which looks at dictionaries or other ontologies to find matches. However, it can use dictionaries and other ontologies to improve performance. The algorithm can find matches based purely on the given node and edge labels. However the algorithm might benefit from a hot start. Such a seed alignment might allow the algorithm to train with certain words that are known synonyms as if they were the same word, thus increasing the number of training samples per word (on known relevant words) and decreasing sparsity. This may help the model, improving performance. However, since some information from other algorithms is now used, it would not be fair to compare the results to the cold start algorithm results. For example, it may just add the results from

the generic string matcher to its own results, improving performance, without learning any new relationships. A new testing method had to be designed for this algorithm. This method would have to compare it to the string matchers, to see if it improved over their results, rather than compare it to the cold start algorithm. However, if we compare the cold and hot algorithms with the generic string matchers, we may be able to compare them indirectly. An example of a hot start is that *mouse torso* is already matched with *human torso*. Then it is likely that their neighbours are also related. Since the *front paws* and *arms* are next to the *torso*, they are more likely to be matched now. In the case of *Node2vec* they are more likely to be matched, since their neighbourhoods are the same (the only neighbour of both *arms* and *paws* is *connectedTo torso*).

3.4.8 Training corpus

Word embedding models need to be trained on a large corpus. These corpora need to cover the concepts that are contained in the ontology, but also need to be large enough to build good embeddings. Since the ontologies themselves do not necessarily contain enough examples to embed the concepts properly, I need to use ontologies that are likely to contain the concepts I want to align. In the case of the mice and men ontologies, anatomical or medical ontologies can be useful, since they may list more anatomical relationships or common relationships (for example *whisker* and *moustache* are related to *hair*). As a benchmark, I will use the WordNet and NELL ontologies. To check if adding the ontologies that are to be aligned helps, I will also train a model on an ontology consisting of WordNet, NELL and the two alignment candidates.

3.4.9 DeepWalk

An alternative way of increasing the number of examples that the algorithm considers is using DeepWalk. DeepWalk generates sentences by randomly walking through the ontology and processes those sentences using a normal embedding model. This has the advantage of having a wider neighbourhood, as the neighbours of neighbours are also used as context, or even further neighbours depending on the scope of the algorithm. It also allows for weighting of edges to change the sampling rate of the random walk, which can be used to prioritize important nodes or highlight informative neighbours.

3.5 Implementation

The foundation the algorithm has been build and tested on is Factorie, described in 2.5.1. Earlier work on a forked branch of factorie was done by [**?**] to implement multi-sense *Word2vec*.

3.6 System check

To ensure the results are valid, I ran a system check. The check is designed to detect the following conditions.

1. The data breaks the considered algorithms.
2. There are bugs in the developed system unrelated to the algorithms.
3. The considered context extraction does not work.

I will investigate all three to ensure they cannot be the cause of bad results. If potential cause one is the case, the study is invalid and should be repeated on one or more different data sets. If either potential cause two or three is the actual cause, the study is invalid and the results should not be published until the system is fixed and the results do not change. If it is the case that none of these three potential causes is the possible, the results are valid, the results reflect the performance of the algorithm and valid conclusions can be drawn from this study.

3.6.1 The data

If the data set is broken, switching to another data set will substantially change performance. Of course, different data sets are inherently unequal in their difficulty. Therefore I will check the performance relative to the baseline and the best known algorithm. If it performs similarly relative to those references, it can be concluded that the data works as intended.

Possible reasons for the data causing bad results include the data set being too small, labels that are not useful or the data set having a structure that does not work with the algorithm. For example, the pre-trained *Word2vec* model is expected to work better on a data set that has commonly used words as labels.

The algorithm has been run on multiple data sets which are different. Despite that there is no significant difference in performance. For example, compare the Average precision for the medical dataset with the anatomy dataset. The So the data does not cause the underperformance of the system.

3.6.2 The system

Since the system parts are connected in sequence, if any of the steps in the software system contains an error, the whole system fails. This could cause all algorithms to learn badly or the results to be misinterpreted. The system consists of a data loader, preprocessor, context extractor, the algorithm, the matcher and multiple visualizers. If all of these systems work properly, the system is not the reason for the underperformance. I will analyse every part except for the context extractor, which will be analysed in the next section.

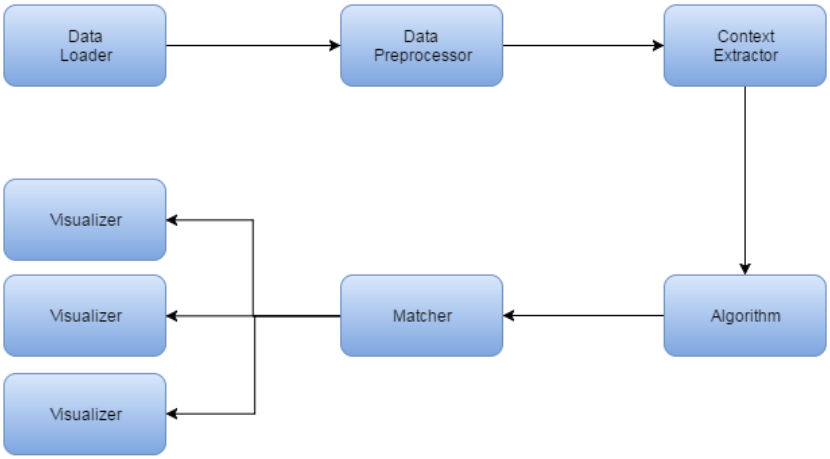

Figure 18: The program is structured in sequence, except the visualisers which are structured in parallel.

Preprocessing

Matcher The matcher is a simple KD-tree that finds the nearest neighbours of every node in the opposing Ontology. This is unlikely to contain errors.

Data loading Since the preprocessing and context extraction works, we can conclude that the data is also loaded properly.

Statistics and visualisation Since the visualizers work in parallel, errors in them would not cascade to the others, so they are all faulty or none of them are.

Since every part works, the system is not faulty.

Secondly, as an extra measure, the following steps have been replicated by another piece of software: the data loader, preprocessor, context extractor and the algorithm. The results of that system are titled "alternative DeepWalk" in the results Section. As this system has been used in other projects and is written by someone else, if the results match the ones from this study, the system from this study works.

3.6.3 Context extraction

There are different context extractors for some of the different algorithms. I will research every one: the sentence generator of DeepWalk, the context extractor for the graph-based models and the context extractor for the pre-trained model. If all three work well, the context extractors are not the cause of the underperformance.

Figure 19

The context extractor for DeepWalk works as generates sentences as described in Section 3.6.5. Since the sentences are generated properly and DeepWalk only requires sentences, the context extraction for DeepWalk works.

For the Node2vec algorithms, the context extraction is more advanced. All edges are extracted from the Ontology using Jena. Then, for every node the context is decided as every node it has an edge with. In the case of the bidirectional algorithm, both outgoing and ingoing edges count as being connected, whereas in the monodirectional algorithm only outgoing edges count as the connectedness. For the algorithms that also use the edge labels for context, double the context size is expected.

The context extraction of the pre-trained Word2vec consists of splitting up every label into multiple words which can be looked up. For example "Adrenal vein" can be split up into "Adrenal" and "vein". The pre-trained model does not use any other form of context integration.

First, I obtained the number of edges from VOWLview[**?**], an Ontology graphing program, which looks like Figure **??**. This was necessary since the creators of the data did not share how many edges were in the database. With the actual number of edges, obtained, it is possible to compare them with the number of edges found. If these match, the context extraction works.

Table 1: The number of edges compared to show that edge parsing was successful.

	#edges mice		#edges humans	
Algorithm	expected	actual	expected	actual
Truth*	**1810**		**3780**	
Monodirectional	1810	1807	3780	3761
Bidirectional	3620	3614	7560	7522

Since the expected number of edges and actual number of edges are very similar, we can conclude that the context extraction works. The difference can be explained by oddities in the VOWLview representation, as it contains edges that are not relevant to the ontology such as in Figure 19.

3.6.4 The algorithm

At the beginning of this section, I reasoned that there are four possible causes for the underperformance of the system. The first three being bad data, bad code and bad context extraction. I have shown that all three are very unlikely

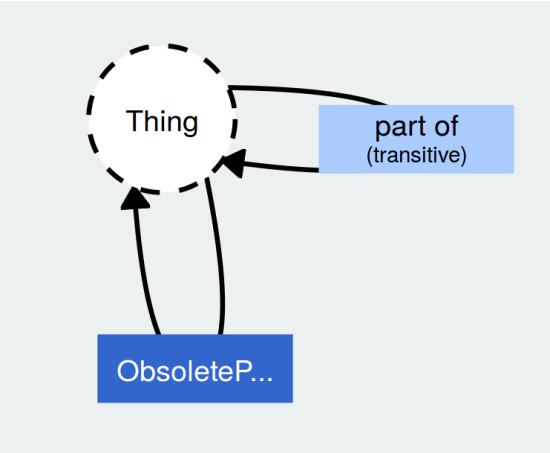

Figure 20: An example of two edges that VOWLview renders that are not relevant to ontology matching.

to be the cause of the underperformance. Since those three possibilities have been excluded, the most likely conclusion is last remaining possibility: the set of distributed embedding algorithms do not work well on Ontology data.

3.6.5 Example: DeepWalk sentences

3.7 Evaluation

The goal of the research is to find an ontology alignment algorithm A and prove that it performs better than current techniques. However, as current techniques may combine many different algorithms and sources of information, maybe such a comparison is not fair, nor is a comparison with individual algorithms as they may be optimised for being combined. So, instead we may want to see if performance is improved when adding A to an ensemble of algorithms $E = (A_1, .., A_n)$.

3.7.1 Evaluation methods

The evaluation should be done using a performance measure $P(Al, T)$ that takes the alignment an algorithm produces given an ontology and the ground truth. If both the alignment and truth are considered sets of alignments, i.e. $Al = \{a_1, .., a_n\}$ and $T = \{a'_1, .., a'_m\}$, then the True Positives $TP = \{a | a \in Al, a \in T\}$, False Positives $FP = \{a | a \in Al, a \notin T\}$ and False Negatives $FN = \{a | a \in T, a \notin Al\}$ determine the performance of the alignment. The True Negatives do not matter, as there are extremely many, and the goal of ontology matching is to find True Positives. If we only care about True Positives, the Jaccard similarity is a prime candidate for the performance measure.

The Jaccard similarity is calculated as follows

$$J = \frac{TP}{TP + FP + FN}$$

i.e. the fraction of the true positives over the sum of the true positives and mistakes.

The Jaccard similarity is a number between 0 and 1, with a higher number being better. Therefore if we compare two alignments, the one which is most similar to the truth, i.e. has a higher Jaccard similarity, is the better one.

However, if we want to find out if an algorithm improves an ensemble, we calculate the improvement as follows:

$$I = J_{E \cup \{A\}} - J_E$$

where J_E is the Jaccard coefficient of an ensemble E. To see if an algorithm is useful to an ensemble, the improvement score I has to be *significantly* higher than zero. To check if a score is significant, a statistical significance test will have to be carried out, which works as follows: generate N solutions randomly and score them. Then score the method you want to investigate. If the score is in the top p%, it is significant, otherwise it is not. In this case the solutions are $E \cup \{R\}$, where R is an algorithm that generates random alignments[**?**].

It is in principle possible to weigh False Positives and False Negatives differently, but this is not done in practice. To make the results of this study easily comparable to others, equal weights will be used.

4 Results

In this section I will show the objective results of the study. Firstly, I will show the precision-recall trade-off curve of all algorithms. Then I will go into detail of the results of the most promising algorithm.

4.1 Precision versus recall

4.1.1 Mice and men data

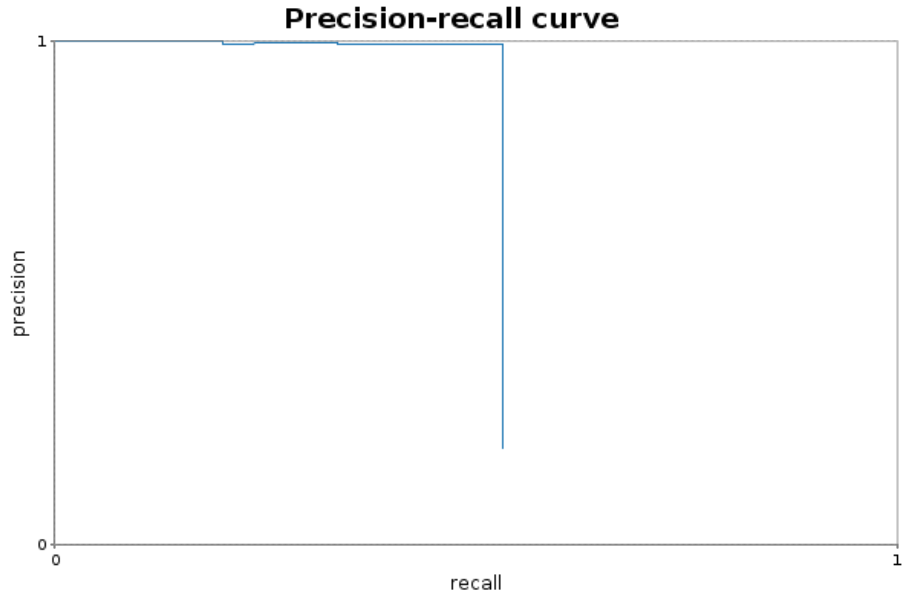

Figure 21: The precision and recall trade-off for the results of the untrained Node2vec. $AP = 0.04194$

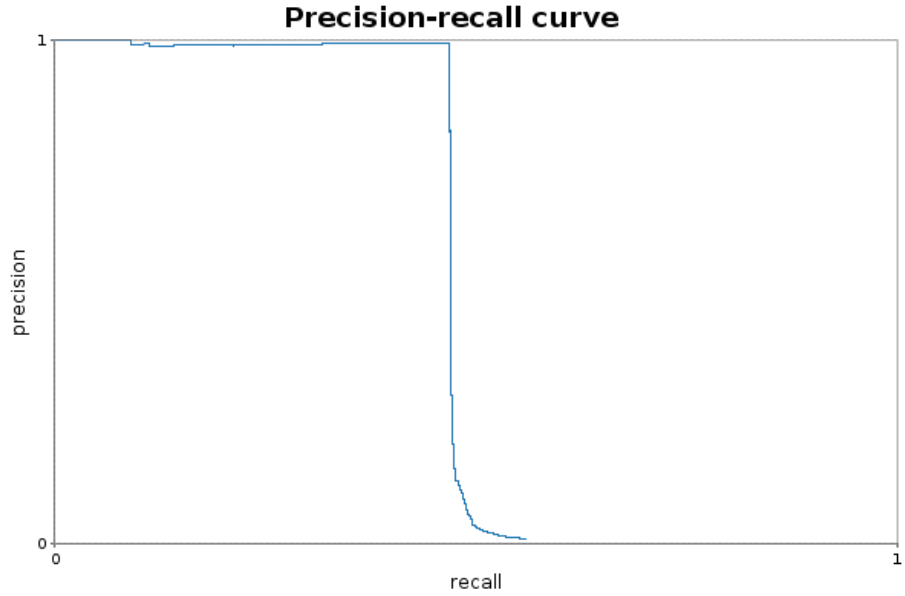

Figure 22: The precision and recall trade-off for the results of the Word2vec algorithm. AP : 0.04636

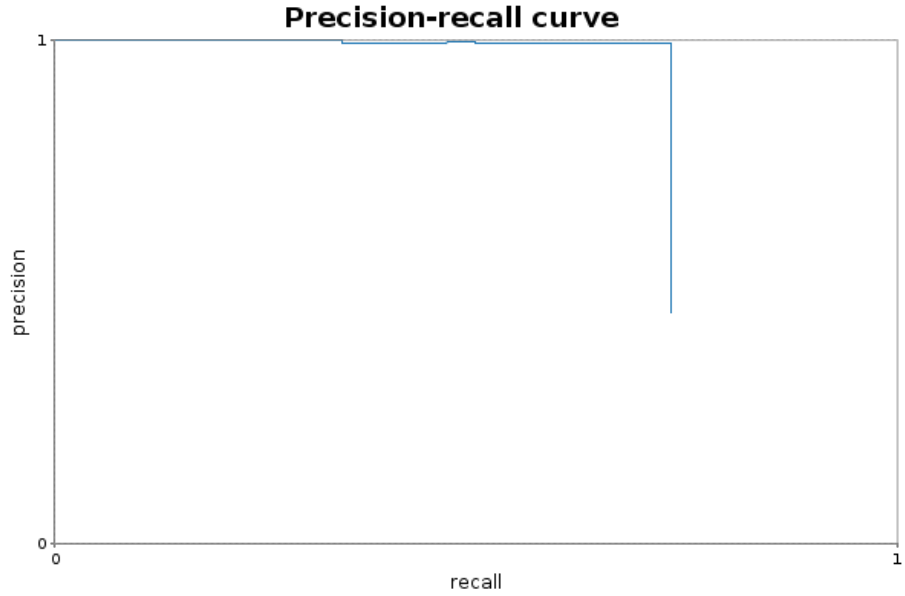

Figure 23: The precision and recall trade-off for the results of the bidirectional Node2vec. $AP = 0.04198$

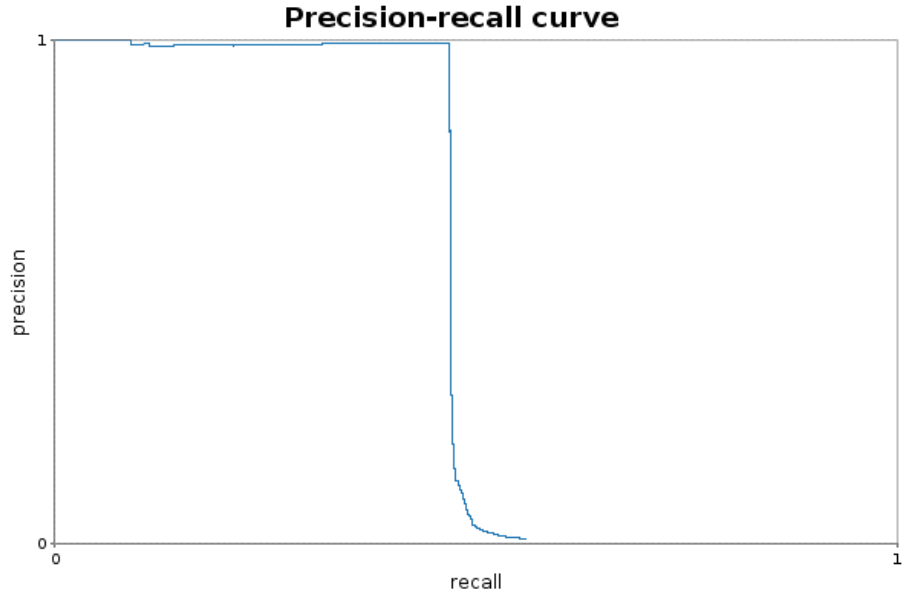

Figure 24: The precision and recall trade-off for the results of the alternative DeepWalk algorithm. $AP = 0.04193$

4.1.2 Medical data

TBD

4.2 F-score

Table 2: The F_1-scores for the different matchers for the mice dataset compared to the result of AML at OAEI 2015 [**?**]

Matcher	F_1**-score**	
	Anatomy	Medical
Alternative	0.6372	0.2494
Bidirectional	0.8432	0.4946
Label match + Random	0.6926	0.3875
Generic	0.6372	0.2419
AML	**0.94**	**0.81**

4.3 Examples to illustrate the differences

TBD

4.4 Interpretation

4.4.1 Unexpected results

4.4.2 Expected results

5 Conclusion and discussion

In this section I will draw conclusions from the results, answer the research question and give a verdict on the hypothesis. In the second part of this section I will discuss these conclusions. In the third and last part of the section, I will describe a few avenues that might be interesting to explore further, including a number of implementations I did not consider and why.

5.1 Conclusions

The goal of this study was to answer the research question from Section 1.5:

Can distributed representations of concepts be used to find relationships between concepts in ontologies better than existing matchers?

To answer this question I have examined the following points from Section 1.6:

The algorithm uses distributed representations of concepts

The algorithm finds relationships between concepts in ontologies

The algorithm has been compared to the currently best matchers

The algorithm performs better I performed empirical research where I made the algorithm, applied it to different datasets and compared the results with existing methods.

Because the last point is not true, the answer is not in agreement with my hypothesis from Section 1.6:

Matchers that use a distributed representation of concepts can find relationships between concepts in ontologies better than the best existing matchers.

The results from all the different evaluation measures are in agreement. From those results we can conclude the following:

*The distributed word representations system is currently **not** able to find relationships between concepts in ontologies better than existing matchers.*

5.2 Discussion

This study shows that the algorithm of McCallum et al as shown in [**?**] does not help improve the results of ontology alignment over the straightforward baseline of simply pairing labels based on their syntactic similarity in the area of Ontology Matching.

The results show that one assumption that was made by the developers of the Universal Schema was incorrect, namely the assumption that when combining knowledge bases matching labels refer to the same entity and vice versa. For example, Bill Gates is called Bill Gates in FreeBase as well as retrieved from unstructured text. In reality, there may be multiple entities named Bill Gates and the same entity may be labelled differently.

This assumption will weaken further as an increasingly growing part of the Semantic Web is automatically generated and as such different labels can be assigned to the same entity. One of the goals of Ontology Matching is to link those differently labelled entities.

5.3 Future research

5.4 Discarded implementations

5.4.1 Just context nodes

5.4.2 Just edges

5.4.3 Context matrix representation

6 Lists of figures and tables

List of Figures

List of Tables

7 Bibliography

8 Appendices

Appendix 1

Appendix 2