Identifying Polysemous Words and Inferring Sense Glosses in a Semantic Network

Maxime Chapuis
ENSIMAG
maxime.chapuis@ensimag.fr

Mathieu Lafourcade
LIRMM
mathieu.lafourcade@lirmm.fr

Introduction

The present paper aims at detecting polysemous words from their hypernyms. For instance, a native speaker knowing that the French word *frégate* (*frigate*) is a *ship* and a *bird* can easily guess that *frégate* is polysemous. Indeed, it is difficult to conceive something being both a *ship* and a *bird* at the same time. We can say that those two hypernyms are *"incompatible"*. If one had a list of all incompatible hypernyms (which will be referred as *incompatibility rules* later in this paper), one could easily detect polysemous words. Is it possible to create such a list ? Can it be done automatically ? To answer these questions we experimented on the French lexical-semantic network JeuxDeMots, Lafourcade (2007), which a free and open resource.

Identifying polysemous words is crucial in order to understand a text. It is usually done by detecting high density components in co-occurrence graphs created from large corpora, as in Véronis (2003). Similar methods have been used by Dorow and Widdows (2003) and Ferret (2004) to discover word senses also in corpora. To detect the different dense areas of their graphs, Dorow and Widdows (2003) used the Markov Cluster Algorithm, van Dongen (2000). These methods are very effective, but they highly depend on the corpora used to create the graphs which might induce many biases. To choose the proper glosses for naming the different word senses, Dorow and Widdows (2003) used the hypernyms present in the lexical network WordNet, Fellbaum (1998). WordNet is also used by Ferret (2004) to evaluate his results. We experimented our approach on the French lexical-semantic network JeuxDeMots, and there is no other complete enough french resources equivalent to WordNet to automatically compare our results to. Hence, we had to rely on some manual evaluation.

In this paper, we will first present the JeuxDeMots network and some of its specificities. Then, we will detail the method we used (a) for generating list of incompatible hypernym and then (b) for inferring glosses for naming word senses, followed by some evaluations.

1 Methods for Dealing with Incompatibilities and Glosses

1.1 Few Aspects of the JeuxDeMots Lexical-Semantic Network

JeuxDeMots (JDM), Lafourcade (2007) is a French lexical-semantic network. It is a knowledge base containing lexical and semantic information. The network is composed of terms (nodes) and relations (edges). The relations between nodes are typed, oriented and weighted. Around 100 relation types are defined, such as synonymy, antonymy, generic (hypernymy), specific (hyponymy) and refinements. Refinements are representations of word senses or usages.

The different refinements of a given term T take the form of (T, glosses) pairs, as T>$glose_1$, T>$glose_2$, ..., T>$glose_n$. Glosses are terms that help the reader to identify the proper meaning of T. For instance, the French term *frégate* (*frigate*), which is a *ship* and a *bird*, has two refinements, *frégate>navire* and *frégate>oiseau*. Thus, a term T is linked to its refinements in the network, through a specific relation type (*r_semantic_raff*).

1.2 Generating Incompatibility Rules

The algorithm used to generate the rules relies on the *refinements* present in JDM to partition sets of hypernyms (there are around 26 000 refined terms and more than 69 000 refinements in the network).

Let **T** be a refined term of JDM with two refinements **A** and **B**. Suppose that **T** has only two hypernyms and that one is a hypernym of **A** and the other a hypernym of **B**. Partitioning the hypernyms of **T** is trivial because you only have to put one hypernym in a partition and the other in a different partition.

Let's go further and assume that **A** and **B** have now multiple hypernyms. The algorithm still creates two partitions but this time, it selects among the hypernyms of **T**, every hypernym *h* which is only in **A** or only in **B**, and puts it in the corresponding group. These groups can be expressed as:

$$\begin{aligned} G_A &= \{h \in hypernyms(T) \wedge h \in hypernyms(A) \wedge h \notin hypernyms(B)\} \\ G_B &= \{h \in hypernyms(T) \wedge h \notin hypernyms(A) \wedge h \in hypernyms(B)\} \end{aligned} \tag{1}$$

This process can be generalised to n sets of hypernyms. Let's assume now that **T** has n refinements R_1, R_2, ..., R_n, then the algorithm selects among the hypernyms of **T**, every hypernym *h* which is only present in one refinement and creates the corresponding groups. The previous expression becomes:

$$\forall j \neq i, \quad G_{R_i} = \{h \in hypernyms(T) \wedge h \in hypernyms(R_i) \wedge h \notin hypernyms(R_j)\} \tag{2}$$

This algorithm gives us a way to group the hypernyms of **T**. Let's run it on an example:

```
hypernyms(T) = {a, b, c, d, e, f}
hypernyms(R1) = {a, b, c, g}
hypernyms(R2) = {a, d, h}
hypernyms(R3) = {e, f, i, j}

GR1 = {b, c}    GR2 = {d}    GR3 = {e, f}
```

The hypernym *a* is present in both R_1 et R_2, therefore it is ignored (it does not meet the condition (2)). The hypernyms *b* and *c* are both hypernyms of **T** and are only in the refinement R_1, thus they end up in the group corresponding to R_1. It goes the same way for *d*, *e* and *f* which are only in R_2 and R_3. The hypernyms *g*, *h*, *i* et *j* are ignored because they are not hypernyms of **T**.

The hypothesis we made is that, if for a term **T** with n senses the algorithm produces the groups G_1, G_2, ..., G_n, the hypernyms of a group are *incompatible* with the hypernyms of all the other groups, meaning that for $i \neq j$:

$$\forall x \in G_i, \forall y \in G_j, \quad x \quad incompatible \quad y \tag{3}$$

The generated rules are represented as:

```
hypernym1|hypernym2|origin|GroupID1|GroupeID2
```

where :

- *hypernym1* and *hypernym2* are two incompatible hypernyms ;
- *origin* is the refined term used to generate the rule ;
- *GroupID1* (resp. *GroupID2*) is a unique integer identifying the group where *hyperonyme1* (resp. *hyperonyme2*) belongs.

Here is an example of a rule:

```
n1="papillon>insecte"|n2="oiseau>animal"|origin="empereur"|gId1=2192|gId2=2191
```

The hypernym *papillon>insecte* (*butterfly>insect*) is *incompatible* with *oiseau>animal* (*bird>animal*). The rule was generated using the term *empereur* which in French is both the name of a *butterfly* and the name of a *bird*. The hypernym *papillon>insecte* belongs to the group 2192 and *oiseau>animal* to the group 2191. The group identifiers will be used later in section 1.4 to choose the right glosses of the refinements.

However, you should proceed with caution when using this method because the JDM network is not complete yet. It contains many silences[1] which could lead to the production of false rules. Let's take the example of the French term *aubergine* (*eggplant*) and its two refinements "*aubergine>plante potagère*" (*eggplant*) and "*aubergine>contractuelle*" (*policewoman*) :

```
hypernyms(aubergine) = {plante, femme, personne, eucatyote, etre vivant}
hypernyms(aubergine>plante potagere) = {plante, eucaryote, etre vivant}
hypernyms(aubergine>contractuelle) = {femme, personne, etre vivant}
```

If you follow the algorithm as it was presented, you will produce the following rules : *plante incompatible femme*, *plante incompatible personne*, *eucaryote incompatible femme*, *eucaryote incompatible personne*. The absence of *eucaryote* (*eukaryote*) in the hypernyms of *aubergine>contractuelle* leads the the production of two false rules (*eucaryote incompatible femme* and *eucaryote incompatible personne*). One solution to the problem would be to add the hypernym *eucaryote* to *aubergine>contractuelle*. However, the fact that a *policewoman* is a *eukaryote* seems to be irrelevant even if ontologically true.

Another solution is to intentionally ignore the hypernyms which are high in the hierarchy. For instance *être vivant*(*living being*) or *métazoaire* (*metzoan*) seem too general to give us useful information. Therefore, the algorithm uses a list of around 50 hypernyms to ignore such as *biconte (bikont), uniconte (unikont), chose (thing) , organisme (organism), etc.* .

1.3 Checking Produced Rules

Despite the previous filtering, the list of rules still contains false or non-productive rules. A rule is considered *valid* if there are at least two examples to back it up and *productive* if it produces at least one result. This is a way to remove rules that are too specific from the list. For each rule (*A incompatible B*), the algorithm searches in the network the terms which have both A and B as hypernyms. Let x be a term having A and B as hypernym. If x is already refined in JDM, x is considered as an *example* of the rule and will be used to validate it (there is at least one example to each rule: the term used to generate it). If x is not refined, it is considered as a *result* of the rule.

We have noticed that rules which have more *results* than they have *examples* tend to be false, therefore they are not validated by the algorithm. Being restrictive when validating the rules is not really a problem. Since they are created in groups (cf section 1.2), there is some redundancy in the list, the results of the rules created from the same groups usually overlap.

Another criteria we used to validate a rule, is that A should not be a hypernym of B and B should not be a hypernym of A, otherwise the rule is most likely false. For instance, the rule "*félin (feline) incompatible mammifère (mammalian)*" is false because a *feline* is a *mammalian*.

At the end of this process, we end up with a list of validated rules. The results of these rules are annotated as "to refine or to correct". Indeed, a term can be detected as polysemous because of an incorrect relation of hypernymy. Therefore the results should be double-checked by an expert. The results are stored as: the term detected polysemous, followed by the rules violated by the term. Here is an example of result for the term *danois*:

```
danois
n1="mammifere"|n2="langue"|origin="mangue"|gId1=1342|gId2=1340
n1="mammifere carnivore"|n2="langue>75266"|origin="persan"|gId1=10767|gId2=10765
n1="langue>75266"|n2="animal>117095"|origin="mara"|gId1=919|gId2=918
n1="langue>75266"|n2="mammifere"|origin="mara"|gId1=919|gId2=918
```

In this example, *danois* has been detected as polysemous because in French this term refers to both the Danish language and a dog breed.

1.4 Choosing Glosses

To further automate the process, we created an algorithm capable of finding the glosses of a refinement in most cases. The idea is to use the rules violated by a word to find the different glosses. Let

[1] A silence is the absence of a relation which should be present between two terms

$R_1, R_2, ...R_n$ be the rules violated by the term T. It is possible, thanks to the group identifiers previously created (see section 1.2), to reconstruct groups of hypernyms. Thus, the hypernyms of the R_i rules are grouped by their group identifiers. These *"local"* groups (*"local"* because they are created using the rules of a specific result) are called the L_i. If we apply this to the example *danois*, we find the following L_i groups:

$$\begin{aligned} L_{1342} = \{\textit{mammifère}\}, L_{919} = \{\textit{langue>langage}\}, L_{10767} = \{\textit{mammifère carnivore}\} \\ L_{10765} = \{\textit{langue>langage}\}, L_{1340} = \{\textit{langue}\}, L_{918} = \{\textit{mammifère}, \textit{animal>zoologie}\} \end{aligned} \tag{4}$$

Applying the same process to the entire list of rules gives you back the groups initially created in section 1.2. These *"general"* groups (*"general"* because they are created using every rule of the list) are called the the G_i. The G_i groups give information about the L_i groups, especially which of the L_i groups can be merged together. When creating the G_i groups, if a group contains a refinement, we decided to add the general term of said refinement to the group. We obtain the following G_i groups for the example *danois*:

$$\begin{aligned} G_{1342} = \{\textit{mammifère}\}, G_{919} = \{\textit{langue}, \textit{langue>langage}\} \\ G_{10767} = \{\textit{mammifère carnivore}, \textit{carnivore}, \textit{félin}, \textit{mammifère}\} \\ G_{10765} = \{\textit{langue}, \textit{langue>langage}\}, G_{1340} = \{\textit{langue}\} \\ G_{918} = \{\textit{mammifère}, \textit{animal}, \textit{animal>zoologie}, \textit{rongeur}\} \end{aligned} \tag{5}$$

Because of the way they are created, we have the following relation between the L_i and the G_i:

$$\forall i, \quad \textit{GroupID}(L_i) = \textit{GroupID}(G_i) \quad and \quad L_i \subseteq G_i \tag{6}$$

After that, the algorithm merges the "*local*" groups which have an intersection with the "*general*" groups different from null. For instance, if $(L_{1342} \cap G_{918}) \neq \emptyset$, it merges L_{1342} and L_{918}. The merge of the groups can be written as:

$$(L_i \cap G_j \neq \emptyset) \Rightarrow \textit{merge}(L_i, L_j) \tag{7}$$

When applying this process to the example *danois*, the algorithm merges its L_i into two groups:

$$\begin{aligned} L_{10767} = \{\textit{mammifère carnivore}, \textit{mammifère}, \textit{animal>zoologie}\} \\ L_{10765} = \{\textit{langue>langage}, \textit{langue}\} \end{aligned} \tag{8}$$

Finally, the algorithm selects in each group the hypernym which has the biggest weight in the network. These hypernyms are used as glosses of the refinements of **T**. For the term *danois*, the algorithm suggests the refinement "*danois>mammifére"* (*mammalian*) for the dog breed, and the refinement "*danois>langue*" (*language*) for the Danish language. The glosses found by the algorithm are not always as accurate as the ones that a human would give, but they are usually true.

2 Results and Discussion

With this method, we created 25 119 rules, 2 785 of which have been validated. With these rules, our system identified 3 171 words as polysemous. To assess the precision of these results, we conducted two experiments. The first one aims to evaluate the performances of the detection of polysemous words. To do that, we selected a sample of 320 terms identified as polysemous, and we checked every term manually (it represents 10% of all the words identified) (see table 1).

Correctly identified	**False positive**	**Precision**	**Error**
285	35	89%	11%

Table 1: Precision of the identification of polysemous words on a 320 terms sample

False positives are either due to incorrect rules or to errors in the network. Indeed, if a term has incorrect hypernyms, the term might be identified as polysemous, even if it is not. However, false

positives are interesting because finding and correcting them can help to increase the overall network's accuracy.

The false negatives are all the unrefined polysemous words of JDM that were not identified as such. False negatives can happen when the words do not have enough hypernyms or when the system does not have the rules needed to identify them. Given the size of the network (more than 2 000 000 terms and 100 000 000 relations) it is difficult to explore the network manually in order to find the number of false negatives. It is important to note that this method is best used on nouns and named entities because adjectives and verbs tend to have fewer hypernyms than nouns and therefore are less susceptible to produce good results.

The goal of the second experiment was to test the accuracy of the inferred glosses. To do so, a sample of 300 polysemous words was selected. The glosses were then sorted in two categories. They were either considered "Correct", meaning that the system found one appropriate gloss for each discovered senses, or considered "Ambiguous or Inaccurate", meaning that the glosses found were too ambiguous to make the difference between the different senses, or that the system found to many glosses[2].

Correct	**Ambiguous or Inaccurate**
232	68
77%	23%

Table 2: Accuracy of inferred glosses on a 300 polysemous words sample

As you can see in table 2, the results are encouraging but the process of finding the glosses automatically still needs some improvements. It is not accurate enough yet to be used without a human verification. It is a quite difficult topic, and even when the glosses are "correct", they are less accurate than the glosses given by humans.

Conclusion

In this paper we have presented two approaches (a) to identify polysemous words in a lexical-semantic network, and (b) naming the discovered word senses by inferring adequate glosses. The results obtained on the JeuxDeMots network are promising as they both contributed to the network refinement and to the increase of its accuracy by detecting potential errors.

A possible improvement, if computation time is not critical, could be to enhance the precision of the glosses by selecting terms that are the most connected in the neighbourhood in the network instead of just choosing the term which weight is the highest.

References

Dorow, B. and D. Widdows (2003). Discovering Corpus-Specific Word Senses. *EACL 2003, pp. 79-82*.

Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. Bradford Books.

Ferret, O. (2004). Découvrir des sens de mots à partir d'un réseau de cooccurrences lexicales. *TALN 2004*.

Lafourcade, M. (2007). Making people play for Lexical Acquisition with the JeuxDeMots prototype. In *7th International Symposium on Natural Language Processing (SNLP'07)*.

van Dongen, S. (2000). A cluster algorithm for graphs. *Technical Report INS-ROOl 0, National Research Institute for Mathematics and Computer Science, Amsterdam, The Netherlands, May.*.

Véronis, J. (2003). Cartographie lexicale pour la recherche d'information. *TALN 2003, pp. 265-274*.

[2]It is the case when the system fails to properly merge the groups. As a result, it proposes more glosses than the word have senses.