Wordnet Wordsense Disambigioution using an Automatically Generated Ontology

Sven Olsen
Swarthmore College
solsen1@swarthmore.edu

Abstract

In this paper we present a word sense disambiguation method in which ambiguous words are first disambiguated to senses from an automatically generated ontology, and from there mapped to Wordnet senses. We use the "clustering by committee" algorithm to automatically generate sense clusters given untagged text. The content of each cluster is used to map ambiguous words from those clusters to Wordnet senses. The algorithm does not require any training data, but we suspect that performance could be improved by supplementing the text to be disambiguated with untagged text from a similar source. We compare our algorithm to a similar disambiguation scheme that does not make use of automatically generated senses, as well as too an intermediate algorithm that makes use of the automatically generated semantic categories, but does not limit itself to the actual sense clusters. While what results we were able to gather show that the direct disambiguator outperforms our other two algorithms, there are a number of reasons not to give up hope in the approach.

1 Introduction

Word sense disambiguation algorithms are valuable because there are a number of tasks, such as machine translation and information extraction, for which being able to perform effective word sense disambiguation is helpful or even necessary. In order to fully define the task of word sense disambiguation (WSD), we need to know the set of senses associated with a given word. What set of senses ought to be associated with any word almost certainly depends on the context we are working in. In the case of automatic translation from English to another language, the best sense set for each word should be influenced by the set of translations of that word into the target language. Translation between distant languages such as English and Inuit might require much finer sense disambiguation than would be needed when going between related languages such as English and German.

WSD becomes a much more tractable problem when we have some understanding of the semantics of the senses that we are disambiguating. For this reason word sense disambiguation experiments are usually do assuming the sense sets of large ontologies such as Wordnet. Using Wordnet senses gives researchers access to information regarding the semantic relationships of the senses of deferent words, and many WSD algorithms rely on knowledge of these relationships. Using Wordnet senses may also make the act of sense disambiguation more useful. For example, an information extraction algorithm may take advantage of the semantic content implied by Wordnet senses.

However, there are a number of reasons why Wordnet might not be the ideal ontology for any given task. If we try to use Wordnet in an information retrieval task we may find that important technical terms are missing (O'Sullivan, 1995). If we try to use Wordnet for machine translation tasks, we may find that the sense distinctions are too fine. In a perfect world, we would have a separate ontology specifically tailored for each task. However, compiling ontologies tends to be very difficult, and so Wordnet is still the de facto standard for most WSD experiments.

Naturally there is a demand for algorithms that can automatically infer ontologies from text, thus providing researchers with an infinite set of viable alternatives to Wordnet. While no current automatically generated ontology can compete with Wordnet's fine sense distinctions, Pantel and Lin (2002) present an algorithm capable of generative sense groups of a quality similar to those in Roget's thesaurus (2002). Unlike Wordnet, this automati-

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

cally generated ontology has no hierarchical information, instead it simply provides groups of related words senses.

In this paper we present and algorithm which automatically generates an ontology given untagged text, and then disambiguates that text into the senses of the generated ontology. Thus we hope to provide researchers with a context sensitive alternative to Wordnet based disambiguation. We also outline a method for converting our senses to Wordnet senses. This allows us to disambiguate text to Wordnet senses by first disambiguating to the automatically generated senses, and then mapping the results to Wordnet. Because we expect the automatically generated sense clusters to be coarser than those of Wordnet, and because the act of generating the senses leaves our algorithm with access to extra information regarding the ambiguous senses, we expect that disambiguating to the automatically generated senses will be easy.

There are ways in which our method of disambiguating to Wordnet senses might have advantages over more direct approaches. Because the senses used by our system are inferred from the text to be disambiguated, we can expect to avoid confusion caused by senses that never appear in our text. Additionally, our system has the advantage of requiring no tagged training data. Mapping the automatically generated senses to Wordnet senses may be complicated by the fact that the generated senses are coarser than Wordnet's, however, we expect that the type mistakes realized because of this to be similar to those mistakes that a human would make when tagging text with the often frustratingly fine Wordnet senses.

2 Related Work

Lin (1994) introduced PRINCIPAR, a broad coverage English parser that works using a message passing model. Among other things, PRINCIPAR can be made to output a set of "dependency triples" given any sentence. Recent work done using MiniPar, PRINCIPAR's publicly available successor, has shown that these dependency triples prove quite useful in the context of a number of different tasks.

Lin (1997) introduces an algorithm for word sense disambiguation based on information from MiniPar's dependency triples.

Lin (1998) includes an excellent articulation of the means through which the syntactic information represented by the dependency triples can be used to infer semantic knowledge. Papers such as our own and Pantel and Lin (2002) tend to rush their descriptions of the methods first outlined in this paper, and readers trying to implement our algorithms for themselves will be well served by referring back to it.

Pantel and Lin (2002) presents an algorithm in which the information from the dependency triples is used to automatically generate sense categories. The same paper also proposes a method for evaluating the similarity between sense categories and Wordnet. Using their own similarity measure, Pantel and Lin found that the categories they created automatically were more similar to Wordnet than Roget's thesaurus.

3 Methods

3.1 Automatic sense clustering

In order to generate our ontology we have implemented the method described in Pantel (2002). The starting point of Pantel's algorithm is the publicity available parser MiniPar. For each sentence in our corpus, we use MiniPar to generate a parse tree, and then from that tree infer a set of dependency triples. Each dependency triple specifies two words and the syntactic relationship between them. For example, one of the triples generated by the sentence "Bob drinks the wine" is [drink, verb/direct object, wine]. We call the construct [drink, verb/direct object,*] a "feature". The frequency of a feature for a given word w is defined as the number of times that we see that feature with w filling in the wildcard slot. Thus, if we see "Bob drinks the wine" in our corpus, one of the frequencies that we increment is $F_{[drink,verb/directobject,*]}(wine)$. The frequency of all features that ever appear with a given word define that word's frequency vector [1].

In order to quickly compile frequency information for a large set of sentences, we use a number of sorted associative containers (STL maps)[2]. We use a map from word-feature pairs to integers to represent a sparse matrix that holds the frequencies of any word/feature pair. We also use maps from strings to integers to store the frequencies of each feature and word. We use yet more maps, these from strings to vectors of strings, to store lists of the features associated with every word, and conversely the words associated with every feature. The map based representation of our data allows us to quickly update frequency information given new sets of dependency triples; $O(log(n))$ string comparisons are required to lookup a value given the string key, and the data structures are such that it is easy to insert information corresponding to novel words and features. But once all the triples have been processed, our map based data structure becomes needlessly inefficient. Therefore we convert as much of the data as possible to an indexed representation, assigning each word and feature an integer label, and collapsing many of our maps to vectors (dropping lookup time from O(log(n)) to O(1), and doing away with the

[1] The concept of feature frequency is explained with more detail in Lin (1998), and with less detail in Pantel (2002).

[2] In order to properly analyze the space/time efficiency of our algorithm, it need to be noted that the version of STL that we use implements maps using red-black binary search trees.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

need for expensive string comparisons)[3].

The basic assumption at the heart of Pantel's algorithm is that semantic similarity will be reflected in the syntactic information inherent in the feature frequency vectors. In other words, if we see [drink, verb/direct object, wine] a lot, and [drink, verb/direct object, beer] a lot, then there is a good chance that beer and wine fit into the same semantic category. We now outline a semantic similarity measure which reflects this assumption.

For each word we compute its mutual information with every feature, and store that information in a sparse vector. The equation for mutual information given a word w and a feature f is

$$mi_{w,c} \quad = \quad \frac{\frac{F_c(w)}{N}}{\frac{\sum_i F_i(w)}{N} \times \frac{\sum_i F_i(w)}{N}}$$

As you can see from the equation, the values that were stored when compiling frequency information were picked to make calculating each word's mutual information vector as fast as possible.

Following the suggestion in Pantel (2002), we multiply our mutual information score by the following discounting factor:

$$\frac{F_c(w)}{F_c(w)+1} \times \frac{min(\sum_i F_i(w), \sum_i F_i(w))}{min(\sum_i F_i(w), \sum_i F_i(w))+1}$$

The theory motivating this discounting factor was not well explained in Pantel (2002), but because we admire his results we follow Pantel's lead.

We define the similarity between two words as the cosine similarity of the associated mutual information vectors.

In order to perform the main clustering algorithm, we create a matrix that caches the similarity between any two words. Taking advantage of the sparse nature of the dataset, we only calculate the similarity of words which share a common feature (preliminary tests show that this strategy allows us to compute the similarity matrix roughly an order of magnitude faster than we could using the naive approach). Had our similarity matrix calculations gone too slowly, we could have further speed up the process by applying the "salient feature" heuristic described in Pantel (2002), however never applied the algorithm to a situation in which the extra speed was necessary. Pantel refers to the processes of setting up the similarity matrix as "phase 1" of the clustering algorithm.

In "phase 2", the words' mutual information vectors are clustered using the "clustering by committee" (CBC) algorithm. The goal of CBC is to create large tight clusters with the additional goal that the centroids of each cluster not be too similar to each other.

The CBC algorithm is recursive. Given a set of elements E, each $e \in E$ contributes a potential cluster composed of between 3 and 12 of the elements most similar to e. Potential clusters are assigned a score equal to the product of their size and their average pairwise similarly. Ideally, we would like to find the highest scoring subset of the set of e's twelve most similar elements and use it as e's potential cluster. Unfortunately, performing an exhaustive search of all possible subsets is computationally expensive. Performing hierarchical average-link clustering seems like one reasonable way to attack the problem, but we suspected that a straightforward greedy search might perform better in practice. Thus our own implementation of CBC uses the greedy search[4].

Potential clusters are sorted according to their score. The sorted list is then traversed, and each potential cluster is added to the set of committees C (initially empty) on the condition that its centroid not have a cosine similarity of more than $\sigma = .35$ with any element of C. If C is empty (which happens in the case that we were unable to create any valid potential clusters in the previous step), we return C.

We then define the set of residual elements R as all e such that e does not have a similarity of at least $\theta = .075$ with any of the committee centroids. Finally the algorithm is recursively called replacing E with R, and we return the union of C and the result.

The algorithms now proceeds to phase 3. Once committees have been created, we generate our ontology by assigning each word to some number of committees; each committee is taken to represent a semantic category. The assignment algorithm works as follows: for each word w, we first select the top 200 committees with centroids most similar to w's mutual information vector. Of those, we consider the committee with centroid most similar to w to be w's first sense. Then we remove the common features of w and the centroid from w's mutual information vector. Now we look for another committee to add w to that has a centroid similar to the new w vector. If at any time the similarity between w and the most similar remaining cluster falls bellow some threshold (in our case .05), we stop assigning senses to w. This method allows us to assign words to clusters that represent very rare senses of that word. Unfortunately, the algorithm is very slow, as the similarity of each cluster to the word vector must be recalculated after every step through the loop.

[3]Our source code is available for the benefit of readers interested in finding out more about the details of the data structures used.

[4]It is unclear what method Pantel (2002) uses to create the potential clusters, our initial interpretation of the paper lead us to believe that Pantel had used the hierarchical approach, but we are no longer certain of this.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

There are a couple of things worth noting about the sense generating algorithm. The committees that a word is assigned to in phase 3 have no immediate connection to the committees that the word helped define, and every word will likely be assigned to some committees that it had played no part is creating. Note also that there is no guarantee a word will be assigned even a single sense.

Given the committees representing its senses, disambiguating a given instance of a word is simple. We calculate the implied feature vector of the instance of the word as if the word instance were a novel word appearing only once in the text. We then find the committee with a centroid most similar to that vector, and say that the sense of the word is the one associated with that committee.

3.2 Disambiguation to Wordnet Senses

Wordnet defines a number of relationships between word senses. Our algorithms only make use of the hypernym and hyponym relations. The hypernyms R of word w are those r for which it is the case that "w is a kind of r". Conversely, the hyponyms of w are the words P such that "p is a kind of w" is true. Thus, 'drink' and 'inebriant' are both hypernyms of a sense of 'wine', whereas 'sake' and 'vermouth' are hyponyms of 'wine'. (According to Wordnet wine is polysemous, 'wine' can also mean a shade of dark red).

In order to link sense clusters created by CBC to Wordnet senses we need to decide what Wordnet sense is associated with every sense of every word in the automatically generated ontology. To do this we search Wordnet for semantic relatives of each ambiguous word's senses in order to find semantically similar words that have a good chance of exhibiting the syntactic features that CBC would have picked up on in the course of its sense clustering. We then find the centroid of that group of words, and decide what Wordnet sense to associate each word's CBC sense with by comparing the centroid of the CBC sense's committee with the centroid of the group of similar words gathered from Wordnet.

As an example, lets say that we some generate the following similarity group for the first sense of wine: {sake, vermouth, champagne, burgundy}. The CBC cluster that we will associate with the first sense of wine will be one based on features that tend to arise around nouns that specify alcoholic beverages. The similarity group for the second sense of wine might look something like {yellow, rose, pink, red}, and thus its centroid vector would be filled with features that are associated with colors. Note that if the text that we are using to generate our senses does not have any instances in which 'wine' is used to describe a color, then we could expect that CBC never added wine to a committee associated with color. In this case we wouldn't map any CBC sense to the second sense of wine (and this is a good thing, as it will force all of our disambiguations of the noun wine to be correct).

Clearly, our mapping method depends on having a good way of creating similarity groups for a given sense. In the case of wine's first sense (vino), the set of hyponyms is very large, and contains nothing but kinds of wine. We would expect kinds of wine to turn up in the same syntactic positions as the word 'wine' itself, so in this case using hyponyms as a similarity set is a good idea. However, the second sense of wine (color), has no hyponyms. What we want in this case for a similarity set are the sisters and uncles of the sense, the hyponyms of the hypernyms of wine (in this case, other kinds of red, and other kinds of colors).

Using the case of wine as our only example, we might conclude that the best way to develop a similarity group for a word sense is to start by collecting its hyponyms, and if they prove too small a set, expand to sister, uncle, and cousin words. We want to avoid using words from too high up in the tree, as 'intoxicant' (one of the hypernyms of wine-*vino*) is not likely to be used in the same sorts of syntactic contexts as 'wine'.

But now consider the problem of creating a similarity group for the word 'intoxicant'. Its hyponyms include things like 'vodka', which will likely have a very different feature vector than 'intoxicant'. The words that we want to see in a similarity group of 'intoxicant' are things like 'barbiturate', 'inebriant', and 'sedative'. These are all sisters of 'intoxicant'.

Because the hyponym favoring approach runs into problems in the case of high level words such as intoxicant, we adopt a method for gathering similar words in which we favor sister words above all else[5], and expand the similarity group to include both daughters and cousin words if we can't find enough sisters to make an informative centroid. Here a similarity group is considered to be "informative" if it contains 15 words for which we have gathered frequency information.

One interesting question is whether or not to limit the allowed members of a similarity group to monosense words. In the case of wine-*color*, two of its sister words are 'burgundy' and 'claret', both of which also hyponyms of wine-*vino*. This example demonstrates a potential problem for our algorithm, if we happen to create a similarity group containing many polysemous words with a shared second sense, the similarity group might create a centroid closer to that second sense than to the one that

[5] It should be mentioned that the first words added to a similarity group are the synonyms. This is a byproduct of the fact that a word's first sister is itself. The Wordnet C library returns its search results in the form of word sets; each set contains a word sense and all of that sense's synonyms. Thus the first search result returned when we look for a word's sisters contains all of that word's synonyms. While we do not consider a word to be part of its own similarity group, we do add all of its immediate synonyms.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

we were trying to represent. We have experimented with limiting similarity groups to monosense words, but found that because most of the words in Wordnet seem to be polysemous, the monosense restriction cripples our algorithm's ability to come up with similarity groups of any significant size.

3.3 Direct and Semi-Direct Wordnet Disambiguation

We have created a direct disambiguation algorithm to compare with our algorithm for disambiguation via CBC senses. Our CBC dependent disambiguation algorithm works by creating a feature vector for the instance of the word to be disambiguated, then finding the CBC sense group closest to that vector, and finally finding the Wordnet similarity group closest to the sense group. The direct algorithm just matches the word instance vector with the closest Wordnet similarity group. Thus, comparing it with our CBC algorithm provides a measure of whether or not mapping to the automatically generated sense is helping or hurting us.

Similarly, we can modify the CBC dependent algorithm by substituting the entire set of committees generated in phase 2 for the set of CBC senses associated with the word. This algorithm allows us to avoid the expensive computation costs inherent in phase 3. Because the "semi-direct" approach has the potential to take advantage of some of the advantages of using an automatically generated ontology (because we are moving first to a coarse sense we can hope that our mistakes will be between senses with similar meanings). However, because of the large number of potential committees, it is likely that the vector that we end up matching with the Wordnet simgroups will be reasonably similar to the vector that we started with, and for this reason our results should tend to be more like those from the direct approach than those achieved using the CBC senses.

3.4 Evaluation Method

We have tested our algorithms using the SEMCOR corpus, our version of which was created by transforming the tags of the Wordnet 1.6 version of SEMCOR to Wordnet 1.7 senses. We comparing our algorithm's disambiguation of polysemous nouns and verbs with the SEMCOR's stated correct senses. All of our word/feature frequency statistics are generated from SEMCOR sentences. To evaluate our performance on a given sentence, we need to align MiniPar's parsing of the sentence with the answers from SEMCOR. This alignment process is necessarily imperfect. Sometimes MiniPar incorrectly identifies the part of speech of a word, and when that happens none of our algorithms have a chance of correctly disambiguating it. In the case that MiniPar incorrectly claims that a word which is not a verb or a noun is one, the extra word makes sentence alignment difficult. We have implemented some fairly simple algorithms that attempt to identify and throw out all such cases. MiniPar will also group words that it feels denote a single concept. For example, "Fulton Superior_Court_Judge" is stored as two words in SEMCOR, but MiniPar treats it is as a single word. In order to make sentence alignment as easy as possible, and avoid many of these kinds of cases, we ignore all proper nouns. Once sentence alignment is complete, we are left with a set of nouns and verbs, their correct Wordnet senses as provided by SEMCOR, and their MiniPar parse tree indices. Those indices are used to gather the related dependency triples, which in turn are feed to our various disambiguation algorithms.

4 Results

4.1 Wordnet Similarity Groups

All of our disambiguation algorithms rely on the Wordnet simgroups. However, a brief investigation of the similarity groups returned by our program demonstrate some worrisome trends. For example, we sometimes fail to find large similarity groups for common senses of words. The simgroup for the first sense of the verb "know" (to be cognizant of information) is:
realize,recognise, recognize.
On the other hand, obscure senses of words can turn up much larger similarity groups. The biblical sense of "know", has the similarity group:
bang, bed, breed, do_it, fuck, jazz, love, make_out, mount, nick, ride, screw, serve, service, tread.
Notice that as feared, many of the words in the similarity group are polysemous, representing relatively obscure senses of other words.

Another nasty case comes up when an obscure sense of a word has a meaning very close to that of a more common sense. For example, the 11^{th} sense of "know" (to perceive as familiar), has a similarity group very close to that of the first sense:
recognise recognize refresh review.

4.2 Disambiguation Performance

For each of the disambiguation methods that we tested: direct, CBC sense based, and semi-direct, we gathered a number of statistics. We store the number of polysemous words which were of sense 1. This allows us to compare our results to the baseline method of assigning each word to its most common sense. We also record the performance of a disambiguator that simply selects a random valid Wordnet sense to assign to each word. Finally we store performance for a third baseline disambiguator, one that uses a "qualified random" approach. The idea here is that we select randomly between the valid disambiguators for which we can find non-empty similarity groups. Such a disambiguator is useful for figuring out how much our success is being influenced by the fact that some word senses are ruled out in the similarity group generation phase.

Because the algorithms used were extremely memory greedy, unix tended to kill the processes after they had run for about an hour. However, one hour was enough time for our experiments to collect a reasonable amount of data, though the trials varied slightly in length depending on what other processes were competing for memory space.

Properly summarizing our results is made more complicated by the problems inherent in word alignment. For example, during our evaluation of the direct disambiguator we successfully aligned 54941 nouns and verbs. 2715 words were discarded because SEMCOR and MiniPar disagreed about whether the word was a noun or a verb, and 8486 more of them were discarded because they were monosense. This information, along with performance statistics for the direct disambiguator and the 3 baseline disambiguators is given in tabular form below (percentages for monosense words and POS error are calculated relative to the total number of aligned words, while percentages for disambiguator performance are calculated relative to the number of attempted words):

	Number of Words	Percent
Monosense	8486	15.5
POS error	2715	4.9
Attempted	43740	79.6
Direct Disambiguator	14591	33.3
Random Choice	10082	23.0
Qualified Random	10244	23.4
First Sense	28392	64.9

Here are results for the semi-direct disambiguator.

	Number of Words	Percent
Monosense	9210	15.3
POS error	3039	5.0
Attempted	47758	76.6
Semi-Direct Dis.	13434	28.1
Random Choice	10941	22.9
Qualified Random	11118	23.3
First Sense	31029	64.9

The CBC based disambiguator often found itself trying to disambiguate words that had no associated CBC senses. Thus we recorded compiled two scores for this disambiguator, one in which we only recorded success when a CBC senses existed and we used it to successfully disambiguate the word, and ”augmented” score in which we had the disambiguator return sense 1 in all cases where no sense cluster was associated with the word. We also have data for the precision and recall values of the CBC disambiguator data, though they don't fit nicely into our chart. Recall was 27.2%, and precision was 35%.

	Number of Words	Percent
Monosense	3570	15.4
POS error	1015	4.4
Attempted	18639	80.3
CBC Dis. (augmented)	10149	54.5
CBC Dis. (pure)	1778	9.5
Random Choice	4447	23.9
Qualified Random	4515	24.2
First Sense	11973	64.2

5 Conclusions

5.1 Wordnet Similarity Groups

All of our algorithms depend heavily on the similarity groups for the sense of each word. Given the problems we saw in simgroup generation, it is surprising that any of our algorithms performed better than chance. In our future work section we speculate on some ways that the similarity groups could be improved, and we imagine that all our algorithms would perform significantly better given better similarity groups.

5.2 CBC

The constant terms that we used in our implementation of CBC were taken from one of Pantel's later implementations of the algorithm. There is always a chance that the algorithm might have performed better given different parameters, but in this case it seems more likely that the problem lies in the size of our corpora. Pantel (2002) uses a 144 million word corpora to generate the frequency information provided to CBC, the SEMCOR data that we used contains slightly under a million words. It is also worth noticing that the corpora used in Pantel (2002) was a composition newspaper texts, while the Brown corpus data that makes up SEMCOR comes from a wide range of sources, including press releases and a number of different fictional genres. The heterogenous character of SEMCOR probably increased the number of different word senses used in the text, therefore making the sense clustering task more difficult.

5.3 Comparison of the Algorithms

The most comforting number in the performance statistics is the very low percent of part of speech errors. This indicates that MiniPar is doing its job reasonably well, providing a solid foundation for our algorithms to work off of. The best performance that we ever see comes from the ”always pick the most common sense” baseline. This is disheartening, but given the poor quality of the similarity groups and the problems we encountered applying CBC to a dataset as small as SEMCOR, it is impressive that any of our algorithms we do better than random

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

chance. The fact that the qualified random disambiguator performs about as well as the random disambiguator is also heartening, as it implies that the gaps in our similarity sets are not making the disambiguation problem significantly harder or easier. Thus, what success the algorithms do achieve beyond the random baseline is solely the function of their ability to use the syntactic information inherent in the dependency triples to infer semantic relationships between words.

The direct and semi-direct algorithms both solidly outperform random choice, and this gives us cause to hope that if the issues with similarity group creation could be worked out, we would be left with a complete system capable of outperforming the "most common sense" baseline.

The results for the CBC based disambiguator look rather miserable at first glance, as the pure version performs worse than random chance. However, it is worse noticing that most of the errors are due to the failure of our application of CBC to create sense cluster, and that problem is a result of the small dataset size. So we can hold out hope that given a large corpus to supply supplementary frequency information, the CBC algorithm might achieve much higher performance. It is worth noticing that in those cases where it has sense clusters available to it, the CBC based algorithm has a precision higher than either of the previous two algorithms. We had hoped that the low precision CBC algorithm could be combined with the highly successful "most common" baseline. While this project didn't pan out (the "augmented" version of CBC is less effective than the "most common" baseline), we can always hope that given larger corpora and better similarity groups, we could have achieved better results.

The fact that the direct disambiguator outperforms that semi-direct disambiguator does not necessarily mean that the semi-direct disambiguator is in all ways worse than the direct disambiguator. Remember that one of the advantages that we hoped to see in the semi-direct disambiguator was errors which had a higher tendency to be mistakes confusing semantically similar senses of a word. However, we had no way of adjusting our results to take into account semantic similarity. While unencouraging, the performance scores alone are insufficient to disprove our hypotheses.

6 Future Work

There are a couple of ways in which the generation of simgroups for Wordnet senses could be improved. At the moment, we have only experimented with methods for generating senses which have a fixed "profile". That is to say, all word senses prefer to have their similarity groups filled with some predefined set of relatives. As we have implemented our algorithm, sisters are preferred over everything else, first order children over cousins, and the most distant allowed relatives are third order cousins. One could imagine changing the set of preferred relations in hopes of getting better results. However, it seems to us that the *right* thing to do would be to build an adaptive algorithm that first inferred a word's position in the synsnet, and then used that information to define the appropriate profile. Designing such an algorithm would be a reasonably large project, we would attack the issue by coming up with a loose parametrization for a family of such adaptive algorithms, and searching that parameter space for the values that maximized the performance of our direct disambiguator.

One of the problems that we observed in our similarity groups was the tendency for rare senses of a word to have simgroups very similar to those of much more common senses. Wordnet contains sense frequency information for each of a word's senses, and we would imagine that our disambiguation methods could be improved by taking advantage of that information when mapping a word instance vector to a Wordnet simgroup; the algorithm could be designed to only return a rare sense in the case that the match was very good, other wise a more common sense of the word would be preferred.

In the course of implementing the CBC algorithm, we saw a couple of ways in which it might be interesting to modify the algorithm. For example, in phase 3, CBC completely removes feature elements included in the centroid of a matched committee. However, it might be more reasonable to subtract the centroid from the word vector, then set all the negative terms to 0 (we thought this seemed like a better way of defining the "residue" of a vector). We also suspected that it might be interesting to enforce the "distance from all other committees" restriction in phase 2 of the algorithm relative to all previously generated committees, instead of just those committees generated in that iteratation of the algorithm. Both of these modifications to CBC would be easy to implement, and we would like to see how these changes to the algorithm would effect both the generated senses clusters and the performance of our disambiguation algorithms.

If the problems hampering our system could be overcome, it would be interesting to compare our results to those achieved by the disambiguator presented in Lin (1997). It represents another "direct approach" that works on principles rather different that those that we used, though like our own algorithms functions based only the dependency triple information.

It is interesting to note that unlike Wordnet, the ontology generated in Pantel (2002) had high coverage of proper nouns, which could make it more suitable for MT tasks. Al-Onaizan (2000) describes a case in which being able to guess whether an unknown proper noun is more likely naming a town or a militia group can improve MT

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

performance. We did not test the performance of our disambiguators on proper nouns, though the only thing that prevented us from doing so was a set of relatively minor technical concerns involving word alignment. If those concerns were overcome, it would be very interesting to see how our algorithms performed in the limited case of proper noun disambiguation.

If we could prove that the CBC based disambiguation system was making different sorts of mistakes than the direct disambiguation system, it would almost certainly be worth trying to create a hybrid model in hopes of combining the advantages of both approaches. Such a system could be implemented by a voting algorithm as in Florian and Wicentowski (2002). It also worth considering ways in which CBC could be modified to produce clusters that are more appropriate for mapping to Wordnet senses. One obvious modification on the current system would be to repeatedly run CBC on SEMCOR in order to find the similarity thresholds for sense clusters that imply sense distinctions most like Wordnet's. If we found that the CBC algorithm was lumping two common Wordnet senses together, we would try increasing the degree to which CBC demanded tight clusters.

Another idea would be to generate our ontology using a clustering algorithm in which sense clusters are initially seeded in a way that reflects Wordnet senses. The simplest way to do this would be to run a k-means clustering algorithm with initial clusters created from Wordnet simgroups. One might try to incorporate some of the ideas of CBC into such an algorithm by forcing clusters to have a certain degree of difference from each other.

What we have done in our experiments is to measure the extent to which disambiguations using CBC senses clusters can be effectively mapped to disambiguations for Wordnet senses. However, it might be interesting to design an experiment that tries to go the other way: we could run an off the shelf Wordnet disambiguation algorithm and then map the results to sense tags in the automatically generated ontology. If Wordnet-to-CBC worked well, but CBC-to-Wordnet worked less well, then we would be able to speculate that CBC was creating senses using a notion of semantics similar to Wordnet's, but with uniformly coarser sense distinctions. However, if Wordnet-to-CBC worked less well than CBC-to-Wordnet we might start to wonder if Wordnet was missing some interesting relationships between words that CBC was somehow picking up on (given the results in Pantel (2002) it seems likely that this wold be the case for proper nouns).

One of our hypotheses was that the sorts of mistakes that might be made by our system would be the result of confusion between two similar senses of a word. We further hypothesized that a human attempting to disambiguate words to Wordnet senses would be likely to make mistakes on the same cases that give our system the most trouble. It would be very interesting if these hypotheses were true, however we lack the funding to properly test them. If we had a couple of graduate students at our disposal, we could set them on the task of hand tagging chunks of SEMCOR. Then we could directly compare the humans' errors with our system's, as well as with other more direct WSD systems. Instead of merely paying attention to overall correctness, we would attempt to determine, as in Pedersen (2002), which cases were found most difficult by which systems, and whether or not our system made mistakes that were more "human-like" than those of the other statistical systems. If gradstudents proved to be unavailable, a large amount of word aligned text from different languages could be used as in Chugur and Gonzalo (2002) to develop a notion of sense similarity. Our hypothesis would be supported if most of our system's errors came from labelings with high similarity to the proper label, while a more conventional system exhibited errors with varied similarity.

References

Al-Onaizan, Y., Germann, U., Hermjakob, U., Knight, K., Koehn, P., Marcu, D. and Yamada, K. 2000. Translating with Scarce Resources. *The 17th National Conference of the American Association for Artificial Intelligence (AAAI-2000),* Austin, Texas.

Irina Chugur and Julio Gonzalo. 2002. A Study of Polysemy and Sense Proximity in the Senseval-2 Test Suite. In *Word Sense Disambiguation: Recent Successess and Future Directions, Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 32-39.

R. Florian and R. Wicentowski. 2002. Unsupervised Italian Word Sense Disambiguation using Wordnets and Unlabeled Corpora. In *Word Sense Disambiguation: Recent Successess and Future Directions, Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 67-73.

Denkang Lin. 1998. Automatic Retrieval and Clustering of Similar Words. *COLING-ACL98*, Montreal, Canada. August, 1998.

Denkang Lin. 1997. Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. In *Proceedings of ACL-97*. Madrid, Spain. July, 1997.

Denkang Lin. 1994. PRINCIPAR—An Efficient, broad-coverage, principle-based parser. In *Proceedings of COLING-94*. pp. 42-488. Kyoto, Japan.

D. O'Sullivan, A. McElligott, R. Sutcliffe. 1995 Augmenting the Princeton WordNet with a Domain Specific Ontology. In *Proc. Workshop on Basic Ontological Issues in Knowledge Sharing, International Joint*

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College

Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada.

Patrick Pantel and Dekang Lin. 2002. Discovering Word Senses from Text. In *Proceedings of ACM SIGKDD Conference on Knowledge Discovery and Data Mining 2002*. pp. 613-619. Edmonton, Canada.

Ted Pedersen. 2002. Assessing System Agreement and Instance Difficulty in the Lexical Sample Tasks of Senseval-2 Appears in the Proceedings of the Workshop on Word Sense Disambiguation: Recent Successes and Future Directions. In *Word Sense Disambiguation: Recent Successess and Future Directions, Proceedings of the 40th Meeting of the Association for Computational Linguistics*, pp. 40-46.

Appeared in: *Proceedings of the Class of 2003 Senior Conference*, pages 69–77
Computer Science Department, Swarthmore College