6-8-1999

An Application of Word Sense Disambiguation to Information Retrieval

Jason M. Whaley
Dartmouth College

Dartmouth College Computer Science Technical Report PCS-TR99-352

An Application of Word Sense Disambiguation to Information Retrieval

Jason M. Whaley

June 8, 1999

Javed A. Aslam, Advisor

Abstract

The problems of word sense disambiguation and document indexing for information retrieval have been extensively studied. It has been observed that indexing using disambiguated meanings, rather than word stems, should improve information retrieval results. We present a new corpus-based algorithm for performing word sense disambiguation. The algorithm does not need to train on many senses of each word; it uses instead the probability that certain concepts will occur together. That algorithm is then used to index several corpa of documents. Our indexing algorithm does not generally outperform the traditional stem-based *tf.idf* model.

1 Introduction

This paper addresses and attempts to synthesize two separate problems: word sense disambiguation and information retrieval. Dekang Lin [7] defines the problem of word sense disambiguation as follows:

> Given a word, its context and its possible meanings, the problem of word sense disambiguation is to determine the meaning of the word in that context.

In other words, consider any word that has several meanings. If that word occurs in some document, performing word sense disambiguation on that word is deciding which of those meanings was intended by the author of the document. The problem of information retrieval is to find the subset of documents within a large and diverse set of documents that are relevant to a certain set of keywords or phrases. That set of keywords is known as a *query*.

These two problems have been studied extensively independently of one another. We will attempt to combine them in such a way that word sense disambiguation improves the performance of an existing information retrieval application. This synthesis naturally arises from an observation made by Ellen Voorhees: if the word sense disambiguation task can be performed sufficiently well, it can help ameliorate two problems inherent in information retrieval, synonymy and polysemy [11]. This paper will propose a method of effectively performing word sense disambiguation on a large and diverse set of documents, and a way to apply that method to the information retrieval task.

1.1 Context and Word Sense Disambiguation

Any algorithm for performing word sense disambiguation uses the context in which a word appears to choose the most appropriate meaning for that word. Most corpus-based approaches to word sense disambiguation consider the words immediately to the right and left of the target word to be the context in which it appears. This narrow window is called the *local context* of the word [7]. One could also look at other words with which the target word co-occurs. This broader definition of context is known as *topical context* [5, 6].

Many approaches to word sense disambiguation use statistics gleaned from large amounts of text that are hand-tagged with the "correct" answers [5, 6, 7] to analyze a new text. The hand-tagged text is known as *training data*. Though local context has been shown to be effective in disambiguating the senses of a particular word form [5], the use of topical context has the advantage that it makes maximal use of small sets of training data. Maximizing the use of training data is desirable because the data is expensive and time-consuming to produce [7]. Topical context effectively increases the size of training data in the following way. Consider a document that contains 100 distinct words. 9900 pairs of words co-occur in that document. Since topical context is interested in which words co-occur, it can use all 9900 of these data points; using local context would get only 400 data points from the same data, if it considers a window at ± 2 around each word. In light of the fact that we wish to disambiguate every word in a given document and have limited training data, our approach to word sense disambiguation will use topical context.

1.2 Bag of Words

The dominant model in information retrieval is the *vector space*, or *bag of words*, model. In this model, documents are simply viewed as collections of words. Each word has an associated weight, which is usually determined by the number of times the word appears in the document. This bag of words is then viewed as a vector in n-space, where n is the number of distinct words that appear in all documents. A query consisting of keywords can similarly be viewed as a vector in n-space. The relevance of a document to a query can then be determining by calculating the angle between the two vectors.

1.3 Methods of Evaluation

The results of a query search are evaluated according to two criteria: precision and recall. Given a set of documents returned by some retrieval algorithm given some query, the *precision* of the algorithm is the percentage of those documents that are actually relevant to the query. Given a document set that has n documents related to a query, the *recall* of a given retrieval algorithm is the percentage of those n documents that are actually returned when that algorithm is run on that query.

The rates of precision and recall in any information retrieval system are greatly affected by two properties of natural language that are generally ignored by search engines: synonymy and polysemy. Two words are *synonymous* if they can have the same meaning in some context. For example, the words "drive" and "crusade" are synonymous. A word is *polysemous* if it can have more than one meaning. "Drive," for example, is a polysemous word.

Synonymy is a problem because words that appear in search queries may be equivalent to different words in the documents being searched. This equivalence will not be recognized by a search algorithm that uses the bag of words representation for documents. For example, it is possible that a document containing the word "drive" is relevant to a query containing "crusade," but will not be returned by the search algorithm. Synonymy therefore reduces the recall of a given system. Polysemy is a problem because words that appear in search queries may be used with different meanings in the documents being searched. The precision of the search is correspondingly reduced.

1.4 Bag of Concepts

If we were able to index documents according to the disambiguated meaning of each word in those documents, rather than according to the words themselves, the problems of synonymy and polysemy would disappear, and precision and recall would be correspondingly improved. Words that mean the same thing would be stored at the same place in the vector, and the different meanings of a given word would be stored in different places. The resulting vector could be called a *bag of concepts*.

Assume, briefly, that there exists an acceptable general solution to the word sense disambiguation problem. Then, turning the *bag of words* model into the *bag of concepts* model is simple. When automatically indexing a collection of documents, we first disambiguate all the words in the document, and consider the document to be a collection of meanings rather than a collection of words. When performing a query, we first disambiguate the words in the query, and then perform a normal vector space query search.

We have attempted to do just that: first, to provide a reasonable general solution to the word sense disambiguation problem, and second, to apply that solution to information retrieval.

2 Background

Our approach to word sense disambiguation and information retrieval makes use of several existing pieces of technology. A cursory understanding of these tools is necessary to understanding the explanation of our experiments.

2.1 WordNet

WordNet, developed at Princeton by Miller, et al. [9], is a lexical database akin to a dictionary or a thesaurus. The major difference between WordNet and traditional lexical databases, such as dictionaries, is in the way it is organized.

The basic unit in WordNet is a "concept," which is represented by a set of synonyms (a *synset*). These synsets are then organized according to various semantic relationships. Noun concepts, for example, are organized in a tree according to IS-A relationships (hypernymy). For instance, "chair" is a hypernym of "chaise longue," "rocker" and "barber chair." Other parts of speech, such as verbs and adjectives, are organized in other ways. Many other semantic relationships between synsets are defined, and therefore WordNet is better described as a "semantic net" than as a tree or hierarchy of concepts.

WordNet is an extremely large project in scope, and most of the details of its organization and implementation are irrelevant to the experiments described in this paper. A much more detailed description of WordNet is available at:

http://www.cogsci.princeton.edu/∼wn/

In the scope of this paper, WordNet is interesting in that it describes a set of unique concepts (the synsets), and states explicitly what words can be used to represent those concepts in a piece of text.

2.1.1 Terms

Several terms related to WordNet will be used with very specific connotations in this paper, and are therefore explicitly defined here:

Definition 1 *A* lemma *is any word form (i.e., series of typographical symbols) present in WordNet.*

For example, "dog" is a lemma.

Definition 2 *A* synset *represents a unique concept in WordNet, and is defined by a set of synonyms.*

The set of lemmas *(cad, bounder, blackguard, dog, hound, heel)* is an example of a synset, in this case representing the concept "someone who is morally reprehensible." All of the lemmas in this synset are synonymous in the sense that they can all be used with this meaning (though they may also have other meanings and be present in other synsets).

Definition 3 *A* concept*, in this paper, is defined to be a synset.*

In this paper, the terms "synset" and "concept" will be used interchangeably. They denote exactly the same thing: a node in the WordNet semantic net.

2.2 SemCor

A semantic concordance is "a textual corpus and a lexicon so combined that every substantive word in the text is linked to its appropriate sense in the lexicon" [8]. In other words, a semantic concordance is a collection of documents where every word in the document has been tagged with its meaning. The Standard Corpus of Present-Day Edited American English (the *Brown Corpus*) "consists of 500 passages excerpted from edited, contemporary American English documents, each about 2000 words long." The topics of the document in the Brown Corpus varies widely. Landes, et al. [4] have used a subset of the Brown Corpus (186 of the 500 documents) to build a semantic concordance.

In their work with the Brown Corpus, Landes, et al. used WordNet as the lexicon. So, SemCor is a corpus of 186 documents in which every "substansive word" has been linked, by hand, to the WordNet synset that it represents in context.

2.3 Related Work

Significant work has been done in word sense disambiguation, though most of it has focused on disambiguating the senses of a particular word form. One such project is described by Leacock and Chodorow [5]. In a similar project, Dekang Lin attempted to disambiguate every polysemous noun in a modestly sized corpus [7]. Less work has been done in combining word sense disambiguation with information retrieval. One experiment in which this combination was attempted was designed and run by Ellen Voorhees [11].

Leacock and Chodorow designed an experiment to test the efficacy of local context clues in automatically disambiguating four sense of the verb "serve" [5]. In their experiment, they made use of a 350-sentence corpus that had been tagged by hand. They trained on data sets of varying size, keeping track of what words occurred at most n words away from each sense of the target word, "serve." To disambiguate an input sentence, they found what words were present within the n-sized window around "serve" and found which sense was most likely to have those words around it, according to their training data. They found that when using a window of size 6 and training on 200 sentences, they were able to choose the correct sense 83% of the time.

They were able to improve this number by using a similarity measure based on WordNet semantic relation and requiring that words similar to the words in the input sentence be present in the training data, rather than that the words themselves be in the training data. This effectively increases the size of the training data. This experiment realized impressive results, but only disambiguates one word form. In order to disambiguate all of the thousands of lemmas in WordNet, very large amounts of training data would be required, and producing that data is extremely labor intensive.

Dekang Lin proposed a method to disambiguate every noun in a corpus using training data that does not need to be tagged by hand [7]. Dropping the requirement that training data be hand-tagged greatly increases the amount of data available. Lin then follows essentially the same reasoning as Leacock and Chodorow, assuming that similar words will occur in identical contexts.

His algorithm works as follows: it first trains on a large data set, storing the local contexts of many word forms. On an input text, it extracts the local context of every word. It then searches the training data to find all the words that occurred in an identical context (the *selectors* of the word). Finally, it selects the sense of the word that maximizes the similarity between the word and its selectors.

After training on a large corpus of Wall Street Journal articles, and testing against the seven news articles in SemCor, Lin found that his algorithm was able to correctly disambiguate 56.1% of the nouns in the SemCor files. He also observed that often times several of the senses of a given lemma will be very similar, and that often humans reading the article will disagree about which sense is appropriate. Lin found that 50,000 randomly generated pairs of concepts were 27% similar on average. Therefore, if two senses of a lemma are at least 27% similar, they can be sensibly used in the same context. So, he reasoned, if his algorithm picked a concept within 27% of the correct concept, it could be considered "close enough." In his experiment, his algorithm picked concepts at least 27% similar to the correct concept (as tagged in SemCor) 68.5% of the time.

An attempt to combine word sense disambiguation and information retrieval was made by Ellen Voorhees [11]. She attempted to index a corpus of documents with the bag of concepts model. The word sense disambiguation method she used was one based essentially on similarity. It works on the intuition that if lemma L is in a document and synset S is the intended meaning of L, then words that are closer to synset S than the others synsets of L will also be in the document.

She analyzed her experiments by indexing several collections of documents. For each collection, there existed a set of queries for which there were predefined "correct" subsets of documents to be retrieved. For every one of these collection of documents, the bag of concepts model consistently performed worse than the bag of words model. Voorhees concluded that her word sense disambiguation algorithm did not work sufficiently well to improve her information retrieval algorithm. Unfortunately, she does not evaluate her word sense disambiguation algorithm outside of the context of information retrieval, for instance by comparing it to SemCor. So, her experiment does not establish a baseline above which word sense disambiguation algorithms should perform in order to be considered useful in information retrieval applications.

3 Word Sense Disambiguation Using SemCor

3.1 Domain and Range

The problem of word sense disambiguation is essentially one of mapping items in some domain, words, to some range, a set of meanings that those words can have.

Therefore, the first task in designing an algorithm to perform word sense disambiguation is to define the domain and range. Ideally, the domain and range chosen should have several characteristics. They should:

1. not be limited to the vocabulary of a specific context (e.g., medicine), so that the method be as general as possible, and

2. map synonymous word forms to the same meaning (rather than identical but separate meanings), so that every "concept" is unique.

WordNet fulfills both of these requirements. It was specifically designed to fulfill requirement 1, and fulfills requirement 2 because each node in WordNet is a set of synonyms, meaning that synonymous senses of different lemmas are mapped to the same synset.

In designing their word sense disambiguation algorithms, Leacock and Chodorow, Lin and Voorhees all chose the lemmas in WordNet as the domain, and the synsets in WordNet as the range. We have done the same. Therefore, the goal of our word sense disambiguation algorithm will be to map all of the lemmas in a given document to the proper synsets.

3.2 Mutual Reinforcement

Intuitively, it is obvious that certain concepts tend to appear in the same documents. Using this intuition, assume that some concept A and some concept B tend to co-occur. That is, if we see concept A, we could reasonably expect to see concept B in the same document. Now, let a be some lemma that can represent A (that is, it is in the synset for A), and let b be some lemma that can represent B (though lemmas a and b might also represent any number of other concepts).

Assume that lemma a appears in some document; it is possible that it is representing A. Lemma b also appears in the document, and can represent B. B is known to often occur with A, so the fact that b occurs in the document reinforces the notion that a is representing A in context. Similarly, the fact that a is in the document reinforces the idea that b is representing B.

This intuition can be exploited to help in the word sense disambiguation task.

3.3 Bucket Algorithm

In section 1.1 we presented several ways to determine the context of a given word, and stated that we use *topical context*. Generally, the topical context of a word is determined by looking at some window around the target word (e.g., all words that appear ± 40 words from the target) [5]. In this paper, however, we use the entire document in which a word appears as the context for that word. This choice is justified by the fact that the documents we use to train (the Brown Corpus) are relatively small and monotopical.

In order to differentiate between possible synsets when analyzing a particular lemma in a document, it is desirable to have some numeric value associated with each of the synsets in the context of that document. Once we have that value, we can say that the synset with the largest value is the correct synset for the lemma. In order to compute these values, we can consider the set of synsets in WordNet to be associated with a set of *buckets*. Each bucket holds the weight of the concept that it represents. As we analyze a document, we can alter the weights in the buckets based on the lemmas that have been encountered. Once we have considered every lemma in the document, we can then look at the weight in the buckets to determine the correct synset for each lemma.

Formally, in order to find the weights in each bucket for a given document, we construct the model diagrammed in figure 1. A document is a set of lemmas. With that document, we will associate a set of concept buckets, one for each synset in WordNet, all initially empty. We will draw edges from a lemma in the document to a bucket in the set of concepts if that lemma could represent that concept, that is, if the concept is a possible meaning of the lemma. Then we make one pass through the document, and for each lemma increase the weights in all the buckets that share an edge with that lemma. Then we make another pass through the document, and for each lemma, we consider all of the concepts with which it shares an edge. The concept whose bucket has the greatest weight is chosen as the concept that the lemma represents in context.

The intuition behind this algorithm is the same as the intuition behind mutual reinforcement: synonymous senses of different word forms will tend to occur in the same document. Those lemmas will have edges to the same concept. When each of those lemmas is encountered, the weight in the bucket for the synset they have in common will be increased twice, whereas the weights in the buckets for the synsets they do not have in common will only be increased once. In the second pass, when each of those lemmas is analyzed, the bucket they have in common will have a greater weight, and will therefore be chosen as the proper synset for each of those lemmas.

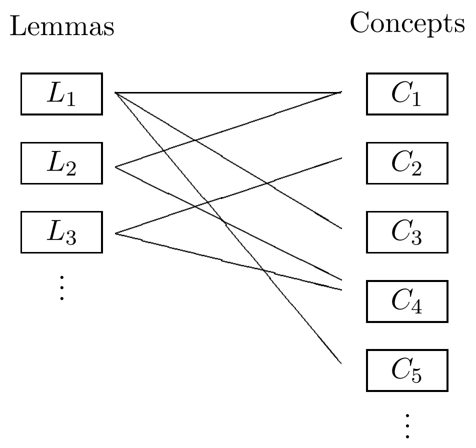

Figure 1: Bucket Algorithm Model

3.4 Edge Weights

The bucket algorithm as described above has a built in bias which can be easily corrected: not all lemmas have the same number of possible meanings. For example, the lemma "draw" has 42 possible meanings, whereas the lemma "appeaser" has only one.

Intuitively, is seems clear that the bucket for the one synset that can be represented by "appeaser" should be incremented more than any of the buckets for the synsets that might be represented by "draw." That is, the presence of the lemma "appeaser" provides definitive evidence for the presence of its corresponding synset, while the presence of the lemma "draw" provides only partial evidence for the presence of any one of its possible synsets.

To account for this disparity, we have attached a weight to each edge, as in figure 2. The most naive way the attach a weight to each edge is to determine the number n of edges coming out of a given lemma, and to give each of those edges a weight of $1/n$. Call the bucket algorithm, as described above, which uses this edge weighting scheme the "Balanced Naive Bucket Algorithm"-balanced because it uses the same weight for every edge in a given lemma, and naive because it does not use other techniques which are described in sections 3.6 and 3.7.

We have analyzed the performance of Balanced Naive Bucket Algorithm on the 186 documents in SemCor, with the following results:

- Maximum Percentage Correct: 52.82
- Minimum Percentage Correct: 27.28
- Average Over All Documents: 40.70

As a basis for comparison, consider that over those same documents, chance would yield:

- Maximum Percentage Correct: 44.82
- Minimum Percentage Correct: 25.66
- Average Over All Documents: 33.17

3.5 Sense Rankings

One problem with the edge weighting scheme described in section 3.4 is that some meanings of a given lemma are generally more likely to occur than other meanings. "Dog," for example, is

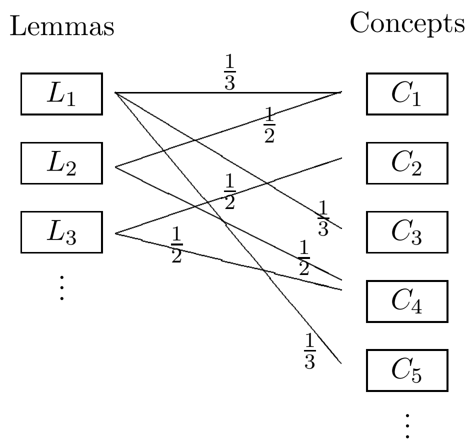

Figure 2: Balanced Naive Bucket Algorithm Model

probably more likely to mean "canine" than "someone who is morally reprehensible." Ideally, a large amount of training data would be available, and the probability of every meaning for every lemma could be calculated. Unfortunately, the large amount of data that would be required to calculate these probabilities is not available.

We can account for these unknown probabilities, however, using a feature of WordNet called the *word sense number*. The word sense number is essentially a ranking of each meaning of a given lemma. We can simulate the priors by giving a higher weight to the synset with the lowest (best rank), and successively lower weights for the rest of the synsets. The way in which we determine how to hand out these weights is the *distribution of weights*.

3.5.1 Linear Weight Distribution

The most naive uneven distribution of weight among the edges is a linear distribution. In this scheme, the edge to the synset with the lowest (best) rank gets a weight of $\frac{1}{n} + \frac{1}{2n}$, and the weights of the remaining edges get progressively lower, such that the synset with the highest (worst) rank is given a weight of $\frac{1}{n} - \frac{1}{2n}$, such that the sum of the weights for all the edges of a given lemma is 1.

The Naive Bucket Algorithm, run with the linear weight distribution scheme, gives the following results:

- Maximum Percentage Correct: 70.52
- Minimum Percentage Correct: 44.65
- Average Over All Documents: 55.76

This is a significant improvement from the Balanced Naive Bucket Algorithm.

3.5.2 Zipfian Weight Distribution

It has been shown that words in natural language texts, when ranked in order of decreasing frequency, tend to be distributed according to Zipf's Law [3]. A Zipfian distribution is one in which the i-th element has a weight $\frac{1}{i}$. The weights in a Zipfian distribution of n elements can be normalized to sum to 1 by dividing each weight by the some of all the weights. The weight given to the i-th element in a normalized Zipfian distribution is therefore:

$$\frac{1}{i\left(\sum_{j=1}^{n}\frac{1}{j}\right)}$$

In light of that fact, we choose to test the bucket algorithm with such a distribution of weights. Running the Naive Bucket Algorithm with a Zipfian weight distribution gives the following results:

- Maximum Percentage Correct: 71.51
- Minimum Percentage Correct: 46.47
- Average Over All Documents: 56.33

So, on average, the Naive Bucket Algorithm using a Zipfian weight distribution chooses the correct concept for each lemma .6% more often than using a linear weight distribution. Moreover, the Zipfian method beat the linear method in 120 out of 186 documents, meaning that it was consistently better. For this reason, the algorithms discussed in sections 3.6 and 3.7 both use a Zipfian weight distribution.

3.5.3 Go With Number One

Given that there is a manually determined "most common meaning" for each lemma, it seems that a good baseline for determining the efficacy of a word sense disambiguation algorithm would be how often simply choosing the highest ranked synset for each lemma is correct. In SemCor, the results of applying this method are as follows:

- Maximum Percentage Correct: 71.51
- Minimum Percentage Correct: 46.47
- Average Over All Documents: 56.34

This is almost identical to the results for Naive Bucket Algorithm with a Zipfian weight distribution. The most likely explanation for this correspondence is that so few synonyms occur in the documents that the algorithm tested in section 3.5.2 almost always chooses the synset with the best rank for each lemma.

3.6 Augmenting the Bucket Algorithm with Co-Occurrence Probabilities

The Naive Bucket Algorithm works on the assumption that synonymous words will tend to occur in the same document, and will therefore mutually reinforce each other, insuring that the correct concept is chosen for each. Because so few synonyms actually occur in the same document, this method is of limited use. Observe the lemmas "bank" and "teller." If those two lemmas occur in the same context, the "cashier" meaning of the word teller reinforces the "financial institution" meaning of the word bank, a relationship not captured in the Naive Bucket Algorithm.

So, if we have some evidence that the "financial institution" concept is in a document (for instance, because we have seen the lemma *bank*), then we can reasonably conclude that there is some probability that the "cashier" concept also appears in the document.

```
foreach lemma in document {
  foreach synset S with edge to lemma {
    increment S's bucket by edge-weight;
    foreach synset T that co-occurs with S {
      increment T's bucket by probability
                that S and T co-occur;
    }
  }
}
```

Figure 3: Pseudo-Code for Co-Occur Bucket Algorithm

Definition 4 *The* co-occurrence probability *of two concepts a and b is the probability that they will occur in the same document.*

The most obvious way to calculate this probability is to find the number of documents in some corpus that contain both a and b, find the number of documents that contain either a or b, and divide the two. Formally, let A be the set of documents that contain the concept a, and let B be the set of documents containing b. The co-occurrence probability of a and b is given by:

$$\frac{\mid A \cap B \mid}{\mid A \cup B \mid} \tag{1}$$

This co-occurrence measure can be applied to the Naive Bucket Algorithm in the following way. For each lemma, the Naive Bucket Algorithm increments each bucket associated with the lemma by the weight on the edge between them. In the new algorithm, for each of the synsets that have an edge with the lemma being examined, increment that synset's buckets by the weight on the edge. Then, increment the bucket of the synsets likely to co-occur with the synset being increment by the weight on the edge times their co-occurrence probability. The pseudo-code for this algorithm is given in figure 3.

Since all of the words in SemCor are tagged with the correct concept, we can simply view each document as a bag of concepts, and use that data to calculate co-occurrence probabilities.

This Co-Occur Bucket Algorithm, trained with the data in SemCor, performed significantly better than the Naive Bucket Algorithm. When tested on the 186 documents in SemCor, it yielded the following results:

- Maximum Percentage Correct: 77.45
- Minimum Percentage Correct: 54.35
- Average Over All Documents: 63.96

These results are significantly better than the algorithm described in section 3.5.3, which simply choose the synset with the best rank for every lemma.

3.7 An Information Theoretic Co-Occurrence Probability Measure

Observe that equation 1 will produce the same number regardless of the total number of times concepts a and b occur in the corpus, or in any particular document. We can turn to information theory to derive a measure of co-occurrence probability that takes the number of times a and b actually occur into account.

First, let us define $I(A \cap B)$ to be the *information content* of the co-occurrences of concepts a and b. According to information theory [2], if a_n is the number of times that concept a occurs in document n, and s_n is the size of document n, then the information contained in each occurrence of a in document n is given by:

$$-\ln \frac{a_n}{s_n} \tag{2}$$

Therefore, the total information contained by all the occurrences of a in document n is $-a_n \ln \frac{a_n}{s_n}$, and the information contained by concepts a and b in document n is $-a_n \ln \frac{a_n}{s_n} + -b_n \ln \frac{b_n}{s_n}$. So, the total amount of information contained in the intersection of concepts a and b within an entire corpus is:

$$\sum_{i=1}^{n} \left(-a_i \ln \frac{a_i}{s_i} + -b_i \ln \frac{b_i}{s_i} \right) \tag{3}$$

where documents 1 through n are all of the documents in which both a and b occur.

Similarly, if A is the total number of times that concept a appears in the corpus, and B is the number of times b appears, and S is the total number of words in the corpus, then the total information contained in all of the occurrences of a and b in the entire corpus is given by:

$$-A \ln \frac{A}{S} + -B \ln \frac{B}{S} \tag{4}$$

Note that this is the same as the amount of information contained by the occurrences of a and b in the union of the set of documents containing a and the set of documents containing b, or more simply $I(A \cup B)$.

We can now combine equations 3 and 4 to find the information-theoretic measure of the Co-Occurrence Probability of two concepts a and b:

$$\frac{\sum_{i=1}^{n} \left(-a_i \ln \frac{a_i}{s_i} + -b_i \ln \frac{b_i}{s_i} \right)}{-A \ln \frac{A}{S} + -B \ln \frac{B}{S}} \tag{5}$$

where documents 1 through n are all of the documents in which both a and b occur. This equation can be simplified, but we have found that the simplification does not make computation any easier in practice, and therefore the simplification is not done.

We can now use this co-occurrence measure in the Co-Occur Bucket Algorithm instead of the measure given in equation 1. When tested on the 186 documents in SemCor, it yielded the following results:

- Maximum Percentage Correct: 78.06
- Minimum Percentage Correct: 55.99
- Average Over All Documents: 65.50

Furthermore, the algorithm using the information theoretic measure was better than the original Co-Occur Bucket Algorithm in 171 out of 186 documents.

3.8 Discussion

3.8.1 Advantages to the Co-Occurrence Approach

Using co-occurrence probabilities available in SemCor provides several advantages over previous word sense disambiguation methods. First of all, it is not reliant on training data for each sense of a given word form. Each synset may have several lemmas in it. If only one of those lemmas is present, it provides data for all of the other lemmas in the synset. So, a given lemma does not necessarily need to be present in the training data to be effectively disambiguated.

Second, because the co-occurrence method uses topical context (rather than local context), it greatly increases the size of the training data. For instance, for a document that has n different concepts, there are $n(n-1)$ pairs of concepts that co-occur. This means that for a corpus of size n, the amount of co-occurrence data acquired from that corpus is $O(n^2)$.

3.8.2 Methodology

Our experiments were performed using PERL scripts that interfaced with MySQL, a freeware SQL database. In order to be able to both train and test on the documents in SemCor without producing artificially inflated results, the tests were performed using 10-way hold-out cross validation. That is, when testing on the first tenth of the documents (documents 1-19), the algorithm trained using the last 90%. When working with the second tenth of the documents, the algorithm trained using the first 10% and last 80% of the corpus, and so on.

The documents that were used for testing consisted of the lemmas as tagged in SemCor. This provides a slightly unnatural bias, since the documents were essentially stemmed and stop-worded by hand. However, it would be possible to simulate these results almost identically using an automatic part-of-speech tagger, such as the tagger implemented and made publicly available by Eric Brill [1].

The way in which we use the sense ranking to determine edge weights is not entirely obvious, for the simple reason that there are multiple sense rankings for each lemma- one each for noun senses, verb senses, adjective senses and adverb senses. We needed a ranking across all parts of speech for each lemma. We constructed such a ranking by interleaving the part-of-speech rankings, i.e.:

$$n_1\, v_1\, adj_1\, adv_1\, n_2\, v_2\, adj_2\, adv_2 \ldots$$

Again, if Brill's tagger were used to determine the part of speech for each lemma, this interleaving would not be necessary.

3.8.3 Random One Half

While performing the experiments described above, we discovered that if the co-occurrence probabilities are divided by two (so that the range of possible probabilities is between zero and one half) before they are added to the buckets, performance increases slightly (less than two percent). All the results reported in this paper were calculated with scripts that multiplied the probabilities by 1/2.

One possible theoretical explanation for this behavior is that it reduces "noise" in the buckets. That is, some buckets that are irrelevant to the document will be incremented as the algorithm is run. If the probability that is added to them is reduced, their effect will be less.

It might seem that the probabilities could be multiplied by any constant without changing the results. However, it should be remembered that the Co-Occur Bucket Algorithm is a specialization of the Naive Bucket Algorithm. So, the weight added to the buckets that have edges to each lemma does not change if the co-occurrence probabilities are multiplied by a constant (only the weight of the synsets that those concepts co-occur with will be changed).

3.8.4 Comparison to Lin

The work most similar to our own is Dekang Lin's attempt to disambiguate every polysemous noun in a corpus. Lin trained his local context disambiguation algorithm using a relatively large (25-million-word) corpus of Wall Street Journal articles, and tested against the 7 news articles in SemCor. He found that his algorithm picked the correct concept 56.1% of the time. Our algorithm was trained using 90% of the 192-thousand-word SemCor, in which documents are not limited to a certain context or style. It picked the correct concept 65.5% of the time.

3.8.5 Possible Improvements

The Co-Occur Bucket Algorithm does not take into account how each word is used grammatically in the document in the disambiguation process. For example, the word run can be used both as a noun and a verb. If that information were available, we could more accurately determine what buckets should be incremented during the first pass, and which buckets should be considered during the second pass. It might be possible to use an automatic part-of-speech tagger to determine how each word is being used in a document as a pre-processing step to the Co-Occur Bucket Algorithm.

Every lemma in SemCor is tagged not only with the concept it represents in context, but also with its part-of-speech in context. We used the part-of-speech data available in SemCor to construct an experiment to determine if this data is a useful way to improve our algorithm (and by extension, to determine if using a part-of-speech tagger as a preprocessor is a good idea). This experiment yielded the following results:

- Maximum Percentage Correct: 84.60
- Minimum Percentage Correct: 64.65
- Average Over All Documents: 74.70

Of course, this is not a fair test of how the algorithm would perform if the each lemma were automatically tagged in a pre-processing step, since the part-of-speech tags were partly generated by hand [4]. However, it does suggest that part-of-speech data would be very useful.

4 Applying Word Sense Disambiguation to Information Retrieval

4.1 Indexing with Word Sense Disambiguation

In section 1.4 we presented the *bag of concepts* method of indexing documents for information retrieval. In order to empirically test the effect of the bag of concepts model of query searches, we modified the Information Theoretic Co-Occur Bucket Algorithm to index a collection of documents.

The algorithm described in section 3.7 was shown to choose the correct synset for 65.5% of the lemmas in a given document on average. Conversely, this means that about 35% of the synsets chosen are wrong. Ellen Voorhees observed that choosing the incorrect synset for a lemma in some document will have a very negative effect on performance in information retrieval. To see why this is so, consider a query that is looking for the correct concept for that lemma. It is impossible that the query return that document [11], ensuring a drop in recall. A similar drop in precision will also occur.

In order to overcome this limitation in the indexing process, we decided to store not one, but three synsets for each lemma, each with a probability weight such that their weights sum to 1. Obviously, this is not done for lemmas that have only one or two possible meanings. Using this method of indexing ensures that more lemmas will have the correct synset stored in the index, even if two incorrect ones are also stored. This method will help to cushion the drops in precision and recall that are inherent in an imperfect word sense disambiguation method. Moreover, the fact that weights are assigned to each synset minimizes the risk that incorrect synsets will worsen search results.

We ran a modified version of the Information Theoretic Co-Occur Bucket Algorithm to determine how often one of the top three concepts for each lemma was correct. Our hope was that these results would be significantly better than the results for the test when only the top concept was chosen for each lemma. This new test yielded these results:

- Maximum Percentage Correct: 94.55
- Minimum Percentage Correct: 80.06
- Average Over All Documents: 86.68

This means that storing the top three concepts for each lemma ensures that the correct concept is indexed 21.1% more often than simply storing the concept with the highest weight. Therefore, this is what we have chosen to do.

Another decision that must be made during the indexing process is how to handle words that do not appear in the lexicon (i.e., WordNet), and therefore do not have corresponding "concepts" that can be indexed. The simplest solution is to keep track of these terms as they are encountered in the test corpus. Each term is assigned an id, just a concepts have id's. In our experiments, there are 99642 concepts which have id's 1 through 99642. Terms encountered in the test corpa which do not have corresponding concepts in WordNet are assigned id's above 99642.

The *bag of concepts* vector that represents a document then consists of two parts. The first section represents the disambiguated concepts found in the document. Term id's in this section of the vector will have values between 1 and 99642, because they represent concepts in WordNet. At the end of the vector, there is a section that represents the "other" terms encountered in the document; they have id's above 99642. In this way, documents can be searched for terms that do not appear in WordNet.

4.2 Stemming and Stopwording Documents

Two common practices in information retrieval are to remove from documents common, and therefore meaningless, words, such as "the," and to reduce the remaining words to their roots. The word "running," for example, will be changed to "run." These two processes are called stopwording and stemming, respectively.

All of the experiments performed in section 3 assumed that a document was a set of lemmas taken from SemCor. Those sets of lemmas were essentially stemmed and stopworded by hand. Stopwording a corpus by hand does not change the validity of any experiments run on that corpus, because the process can be fully automated without changing the results. Stemming manually, on the other hand, gives experiments run on that corpus an unfair advantage. The source of this advantage is that word forms can have more than one stem, depending on how they are being used. Consider the word "thought." In the sentence *I thought you said something*, the stem of "thought" is clearly "think." However, in the sentence *That's an interesting thought*, the stem is "thought."

When a corpus is hand-stemmed, the person doing the stemming can tell how the word is being used, and therefore what the stem should be. When we automatically index documents in an information retrieval test corpus, however, we do not know how each word is being used. Our only choice is to try stemming each word to each part of speech, and picking the first one that gives a valid stem. That is, we first assume that the word is a noun and try to stem it as if it were a noun. If that fails, assume it is a verb, etc. After performing this process, we will be left with a set of lemmas that we can then input to the Bucket Algorithm. This is what we do in the following experiments.

4.3 Performing Query Searches

One major obstacle to performing query searches with the bag of concepts model is determining what concepts should be searched for. Because queries are generally very small, it is difficult to automatically determine what concepts are present [11]. However, it seems clear that the most reasonable thing to do is to perform exactly the same transformation on the queries that was performed on the documents. In practice, this seems to yield reasonable results.

The results of disambiguating queries using the algorithm described in section 4.1 is a concept vector containing several concepts with an associated weight. The weights in that vector are very important, since they can vary widely. Therefore, a modified version of a standard query search model, one that takes the weight of query terms into account, must be created.

As our starting point, we take the *tf.idf* model, which correlates a relevance score based on how often a term occurs in a document versus how often it occurs in the entire corpus. Calculating an summing these scores over all query terms for each document will yield a ranking of the documents. We start with Robertson's *tf* score and a standard *idf* score defined as follows [10]:

$$\begin{aligned} tfbel_{t,d} &= \frac{tf_{t,d}}{tf_{t,d} + 0.5 + 1.5\frac{len_d}{avgdoclen}} && (6) \\ idf_t &= \frac{\log\left(\frac{N+0.5}{docf_t}\right)}{\log(N+1)} && (7) \end{aligned}$$

where $tf_{t,d}$ is the weight of term t in document d, len_d is the total weight of document d, $avgdoclen$ is the average weight of all documents in the corpus, $docf_t$ is the total weight of term t in the corpus, and N is the total weight of the corpus.

To calculate the relevance value of some document d to some query in this standard model, one simply computes the following sum:

$$\sum_{i=1}^{n} (tfbel_{i,d} \times idf_i) \qquad (8)$$

Corpus	Bucket Idx	Standard	% Diff	Sense Vector	% Diff
TIME	.6601	.6891	-4.2	.6044	+8.4
MED	.5344	.5527	-3.3	.4405	+17.6
CRAN	.4272	.4246	+0.6	.2729	+36.1
CISI	.1546	.2426	-36.3	.1401	+9.4

Table 1: Results of Retrieval Experiments

where terms 1 through n are the terms in the query.

We can modify equation 8 to use weighted query terms by simply adding the weight of the term in the query to the product within the sum. So, define $wgt_{t,q}$ to be the weight of some term t in a query q, as determined by our word sense disambiguation algorithm. Our new equation for calculating the relevance of some document d to query q is defined as follows:

$$\sum_{i=1}^{n} (tfbel_{i,d} \times idf_i \times wgt_{i,q}) \tag{9}$$

where terms 1 through n are the terms in the query.

A search that uses equation 9 is performed as follows. First, the query is disambiguated to determine what concepts should be searched for. During disambiguation, each of those concepts is assigned a weight. Then, every document in the corpus that contains at least one of those concepts is found. The relevance value for each of those documents is calculated, and the documents are ranked accordingly.

4.4 Evaluation

To judge the effectiveness of our disambiguation algorithm in information retrieval, we performed precision and recall experiments on the following corpa:

- TIME - consisting of 425 articles from Time magazine and 83 queries,
- MED - consisting of 1033 articles on medicine and 30 queries,
- CRAN - consisting of 1400 articles on engineering and 225 queries, and
- CISI - consisting of 1460 articles on information science and 35 queries.

The results of the experiments are summarized in table 1, and are reported as three-point averages across all the queries in the respective corpa. The 3-point average is determined by calculating the precision for each query at recall values of .2, .5 and .8. Each column in the table represents the results obtained by one of three different indexing schemes. The first column, "Bucket Idx," lists the results achieved by our indexing scheme, which uses the bucket algorithm to perform word sense disambiguation. The "Standard" value for each of these corpa was calculated by Ellen Voorhees using the SMART system's stem-based indexing method, which is widely used and considered a good benchmark. The "Sense Vector" column contains the results of Ellen Voorhees' experiment in sense-based indexing. The value in the table is taken from the results of her experiment which most resembled ours, in which the disambiguated senses and unresolved words were given the same weight. In a separate experiment, Voorhees put twice as much weight on the unresolved words; that experiment yielded slightly better results, though they were still worse than both the standard SMART system and our own.

Several things should be kept in mind when looking at this data, particular regarding the TIME corpus. That particular document collection is very small, and many of the queries have only one or two relevant documents. That means, for example, that if the relevant document is the second one returned, the precision values at recalls of .2, .5 and .8 will all be .5, yielding a 3-point average of .5. This small amount of data means that the results tend to very widely from query to query. Across all 83 queries in the TIME corpus, the average 3-point average is .66, however, the standard deviation is .33. The difference between .68, .66 and .60 is therefore not necessarily a significant one.

The MED, CRAN and CISI corpa are larger than the TIME corpus, and are therefore more reliable indicators of performance. The standard deviations in the 3-point average for those three corpa were .2158, .2751 and .1239 respectively. The results for both MED and CRAN mirrored the results for TIME, and are probably more reliable due to their smaller standard deviations. Our algorithm performed surprisingly poorly on CISI. This could because of the nature of the queries, which were generally smaller and therefore harder to disambiguate than the queries in the other three corpa. Voorhees found that her sense-based vector indexing model also performed very poorly on the CISI corpus. However, the fact that the bucket algorithm performed consistently better than the sense-based vector model and performed nearly as well as the traditional stem-based vector model implies that, with some simple improvements, the bucket indexing algorithm could become very useful.

4.5 Possible Improvements

One aspect of WordNet which our indexing algorithm does not take advantage of is collocations, or word phrases, such as "look up." An algorithm can easily be devised to automatically join the words in a document into collocations [4]. Because there are many collocations present in the data used to train the bucket algorithm, finding collocations in documents to be indexed would probably improve the concepts chosen for all words in the document, in addition to choosing the correct concept for the collocation.

In section 3.8.5 we presented the idea of using part-of-speech tagging to improve the accuracy of the bucket algorithm. This idea translates directly into an information retrieval context, with one important caveat. While it may improve the set of concepts chosen to represent a document, part-of-speech tagging is not easily applied to queries. This difficulty arises from the fact that queries are not necessarily grammatical sentences, and even if they are, they are not likely to be large enough for part-of-speech tagging to be used effectively.

One advantage of the system as we implemented and tested it is that it performs exactly the same analysis of documents and queries. So, if a word occurs in similar or identical contexts in a query and a document, the algorithm will probably assign the same concepts to that word, and query search performance will correspondingly benefit. If part-of-speech tagging were used only on documents and not on queries, this similarity would be lost, which might have an adverse affect of the accuracy on query searches.

References

[1] Eric Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In *Natural Language Processing Using Very Large Corpora*. Kluwer Academic Press, 1997.

[2] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications. Wiley, New York, 1991.

[3] Robert R. Korfhage. *Information Storage and Retrieval*. Wiley, New York, 1997.

[4] Shari Landes, Claudia Leacock, and Randee I. Tengi. Building semantic concordances. In *WordNet, an Electronic Lexical Database*, pages 199–216. MIT Press, Cambridge MA, 1998.

[5] Claudia Leacock and Martin Chodorow. Combining local context and wordnet similarity for word sense identification. In *WordNet, an Electronic Lexical Database*, pages 285–303. MIT Press, Cambridge MA, 1998.

[6] Claudia Leacock, Geoffrey Towell, and Ellen Voorhees. Towards building contextual representations of word senses using statistical models. In *Corpus Processing for Lexical Aquisition*, pages 97–113. MIT Press, Cambridge MA, 1996.

[7] Dekang Lin. Using syntactic dependency as local context to resolve sense ambiguity. In *Proceedings of the ACL-97*, Madrid, Spain, 1997.

[8] G. A. Miller, C. Leacock, R. Tengi, and R. T. Bunker. A semantic concordance. In *Proceedings of the ARPA Human Language Technology Workshop*, pages 303–308. Morgan Kaufman, San Francisco, 1993.

[9] George Miller. Nouns in wordnet. In *WordNet, an Electronic Lexical Database*, pages 23–46. MIT Press, Cambridge MA, 1998.

[10] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of ACM SIGIR 1998*, Melbourne, Australia, 1998.

[11] Ellen M. Voorhees. Using wordnet for text retrieval. In *WordNet, an Electronic Lexical Database*, pages 285–303. MIT Press, Cambridge MA, 1998.