Automatic Thesaurus Generation from Sub-thesaurus

Yuxiao Wen, Yuanfan Wang, Xintong Zhu

Abstract

Most thesaurus constructions via context distribution, when computing two words' similarity, only consider exactly identical words that show up in neighboring contexts. Is there any information, however, contained in words that are not literally the same but may share similarities in their semantics? This paper proposes a word similarity algorithm that accounts for a pre-established thesaurus based on Lin's [1998] similarity. After parsing and analyzing a given corpus, for given two words, we capture their hidden relatedness–measured by the similarity of their disjoint neighbors–and use their mutual information to compute the similarity between two words. In the case of having no pre-established thesaurus, an iterative formula is proposed to capture such relatedness and is proved to converge as more iterative steps are conducted. We compare our results to WordNet and Lin's results. The results indicate that our algorithm indeed perfect the similarity computation by recapturing the loss of information in the iterative steps until no information is lost in current iteration.

1 Introduction

Synonyms have posed challenges to natural language processing through time, and people have spent decades perfecting thesauruses, hoping to recuperate the lost information during searches and to resolve the potential ambiguity in semantics analysis. Via a comprehensive thesaurus system, people nowadays are able to better implement the information retrieval algorithm. Algorithms of automatically identifying information are improved as well, along with the unsupervised articles clustering based on semantics.

Since synonym study is related to semantics, and therefore essentially depends on human beings' understanding, to enable computers to automatically and unsupervisedly distinguish synonyms is actually a profound and complicated task, a field that always contains broad space to progress and enhance. This paper further proposes an enhancement of word similarity algorithm, mainly based on Lin's [1998] algorithm.

Word similarity has a crucial role in constructing a thesaurus. Identifying the similarity between words is an essential step in classifying synonyms. The traditional way to define and compute the similarity between two words (which would also be introduced and discussed in Section 2) is based on the distributional foundation of the words i.e., to check how many similar words are used in contexts relavent to the given two words, and here we define the "neighbouring context words" to be words that have dependency relationships with the target two words. A relative comprehensive method to compute word similarity is proposed by Lin in 1998. He proposes a formula using mutual information which is computed by generating dependency triple of input documents. This algorithm, however, lacks consideration of certain situations. Assume that we want to compute the similarity between two words "dog" and "human", and we have two sentences in input documents "Dogs nap" and "Humans sleep". Note that based on these two sentences, the similarity between "dog" and "human" should be 0, since there is no identical words in the neighbouring contexts. Based on semantics understanding, however, these two words in effect shares some similarities in the sense that the words "nap" and "sleep" are similar in semantics, although they are not identical. Therefore, using Lin's algorithm, certain information is actually lost while computing the word similarity, and thus the similarity is underestimated.

Aiming to solve this potential problem, this paper proposes an enhanced algorithm. Instead of computing the word similarity only once, we use

an iterative method. In each iteration, we take account of the word similarities we have computed in last iteration while computing the word similarities in the current iteration, through which we are able to recapture the information lost in last computations. Observe that this process would lead to an increasing similarities computation, and the ideal situation would be that this computation process would converge as more and more iterations are conducted. Our project derives a formula to compute word similarity which can be indeed proved to be monotone increasing and convergent as more iterations are conducted. In this way, still consider the example above: at the first iteration, though the similarity "dog" and "human" is still 0 in the above two sentences, the similarity between "nap" and "sleep" would be computed based on their neighbouring context throughout the entire input document; therefore, in the second iteration, in re-computing the similarity between "dog" and "human", the algorithm would plug in the similarity of "nap" and "sleep", and therefore the result for the similarity of "dog" and "human" based on the current two sentences is not 0 in this iteration, which is definitely more reasonable.

Our project evaluates our algorithm by comparing our result to WordNet–a widely acknowledged handcrafted thesaurus built by Stanford–and by comparing our result to Lin's. Since WordNet does not contain a value to represent the exact similarity between arbitrary two words, based on Lin's paper, we first define a word similarity measure based on the structures of WordNet. We sort the similarities of a certain word in descending order, and store the top 100 similar words to a certain word. We conduct this sorting procedure to both WordNet similarity computation result and our algorithm's, and then we use a measurement, mainly based on cosine similarity, to compute the similarity between two thesaurus. We then evaluate our comparison results with the comparison results of Lin's algorithm to WordNet.

The next section introduces the background of this project and explain more in details about some related paper works, especially the ones that our project's inspirations come from. In Section 3, we define some essential concepts and mathematical formulas, and prove the monotone increasing and convergence of our iterative word similarity formula. In Section 4, we introduce our input and output files, propose our algorithm, and display parts of the running results of our algorithm. The evaluations are explicated in details in Section 5, and some possible future works are stated in Section 6.

2 Related Work

Over the past a few decades, topics on thesaurus generation has collected much academic interest and has been extensively studied. A thesaurus is defined by Schutze and Pedersen [1997] to be "simply a mapping from words to other closely related words." More explicitly, Kilgarriff [2003] defines it to be "a resource that groups words according to similarity." Consequently, finding a solid similarity measure between words is crucial to a thesaurus generation, and a given similarity measure spontaneously generates a thesaurus.

One approach to construct a thesaurus is to handcraft one, and yet handcrafted thesauri suffer from many problems, such as inconsistency. As one of the most seminal and successful handcrafted thesaurus, WordNet [Fellbaum 1998] is a very comprehensive, general-purpose one with many of those disadvantages reduced. Nonetheless, its comprehensibility makes itself less perfect for domain-specific requirements. For example, the word "map" is almost identical to "function" in mathematical literature, while its general synonym may be much more deviating and hence misrepresenting. Moreover, relatedness of words is more than synonym along; "dog" and "cat" are surely not synonyms, but they are obviously related in the sense that they are both domestic animals.

To make up for such inadequacies, numerous valuable works have been done on Automatic Thesaurus generation that measures word relatedness, namely a semantic similarity, based on some given corpus. The choice of the input corpus allows any user to generate domain-specific thesaurus in one's favor. Many of these works focus on defining a valid similarity measure from where a thesaurus naturally germinate. Based on the well-known WordNet, Leacock and Chodorow [1998] proposes the concept of Least Common Subsumer (*lcs*) that looks for the depth from a shared ancestor to the two words of interest in the *is-a* hierarchy in WordNet (if both "cat" and "dog" are instances of "animal", then they both have "animal" as an ancestor). A Leacock-Chodorow similarity is later defined to reflect the information of this depth.

On top of the *lcs* concept, Pedersen, Patwardhan,

and Michelizzi [2004] proposes a path-similarity that measures the semantic relatedness between words also under the taxonomy of WordNet. They determine the extent of similarity of two words by their paths in WordNet. This measure is well-studied, general, and hence a usual baseline for comparisons.

From a statistical perspective, Resnik [1995] writes on the idea of gleaning information on a word via its frequency in a corpus and then computing the similarity between words based on the Information Content *ic*. Note that Resnik's similarity can range beyond 1. Further, Jiang and Conrath [1997] merge approaches of Resnik [1995] and Rada et al. [1989] and several others in devising a new similarity. Their similarity is based on both *ic* and *lcs*, and it is normalized to 1.

Later, Lin [1998] has done an exceptional work on Automatic Information Retrieval based on word distributional dependencies. He defines the measure, given two words and a fixed corpus, by the fraction of the sum of all mutual information the two words share and the sum of all mutual information of each of them. His definition is underlain by the concepts of *ic* and *lcs*. The resulting thesaurus (generated by a WSJ corpus) is compared with WordNet and Roget Thesaurus and several other semantic similarities, and the comparison gives a convincing reason for adopting Lin's similarity for thesaurus generation tasks.

Overall, though, most of the explorations on semantic similarity assume no knowledge on the words and focus on the contextual coincidence of the exactly same words in comparing two words. Yet as mentioned in Section 1, information may be retrieved from words that are not literally identical, if certain knowledge is pre-established. Therefore, it is worth studying the automatic thesaurus generation based on certain extent of knowledge, i.e. a thesaurus on some words. This generation from a sub-thesaurus collects more information for each word and thereby internalizes some previously hidden relatedness.

3 Semantic Similarity

3.1 Definition

3.1.1 Dependency Triples

In a distributional analysis on a given corpus, we can parse the corpus into triples (w_1, r, w_2) where w_1 and w_2 are words and r is the relation between them. (cell, subj-of, absorb) for example.

3.1.2 Mutual Information

After parsing the corpus into dependency triples, we denote the number of occurrence of triple (w_1, r, w_2) by $\|w_1, r, w_2\|$. We refer to Lin [1998] for a clearer deduction of the definition of the mutual information $I(w, r, w') = \log \frac{\|w,r,w'\| \|*,r,*\|}{\|w,r,*\| \|*,r,w'\|}$ between two words w and w' with r a relation in-between. Similar in Lin [1998], we define $T(w) = \{(r, w') \mid I(w, r, w') > 0\}$ to be the set of informative (r, w') pairs. They are informative in the sense that each (r, w') has a positive mutual information with w.

3.1.3 Hidden Relatedness and Similarity

We clarify our notation: denote any thesaurus $\mathcal{T}$ by a pair (S, sim) where S is a set of words and $sim : S \times S \to [0, 1]$ is a similarity mapping in range from 0 to 1.

Notice that the mutual information only captures the exactly identical words as discussed in Section 1. Now to incorporate the information provided by a sub-thesaurus, we define the **hidden relatedness** of two words as follows:

Let $\mathcal{T}_0 = (S_0, sim_0)$ be the given thesaurus, C our input corpus, S the set of words in C, and $T(\cdot) : S \to R \times S$ the informative sets for a given word, computed from C as in Section 3.1.2 and in Lin [1998]. Further suppose that $S_0 \subseteq S$.

We first extend sim_0 from S_0 to S by setting: for $w_1 \notin S_0$ or $w_2 \notin S_0$, $sim_0(w_1, w_2) = 1$ if $w_1 = w_2$ and 0 otherwise. This extension preserves the provided information in the sub-thesaurus without adding any more to it.

Then we define the hidden relatedness $R(w_1, w_2)$ by:

$$\sum_{\substack{(r_1,w_1')\in T(w_1)\\(r_2,w_2')\in T(w_2)\\(r_1,w_1')\neq(r_2,w_2')}} sim_0(w_1', w_2')(I(w_1, r_1, w_1')+I(w_2, r_2, w_2'))$$

Now to combine the (direct) mutual information and (indirect) hidden relatedness, we define the semantic similarity $sim(w_1, w_2)$ by:

$$\frac{\sum_{(r,w)\in T(w_1)\cap T(w_2)}(I(w_1, r, w)+I(w_2, r, w))+R(w_1, w_2)}{\sum_{(rw)\in T(w_1)} I(w_1, r, w)+\sum_{(rw)\in T(w_2)} I(w_2, r, w) + R(w_1, w_2)}$$

3.2 Iterative Approach

To deal with situations where we are given no pre-established sub-thesaurus and need to excavate for

our own, we propose the following iterative approach. Note that for a given corpus C (and hence a fixed S), we can iteratively run the aforementioned similarity computation on S and get a sequence $\{sim_1, sim_2, ...\}$. In the first iteration we set $sim_0 = 0$ and hence reduce the definition to Lin's similarity, i.e. $sim_1 = \text{lin\_sim}$. Then iterations after that can be seen as capturing previously left-away relatedness. In this section we will show that this iterative process point-wise converges to some similarity measure sim_∞ (Theorem 2). Furthermore, sim_∞ behaves in some way in accordance with lin_sim (Theorem 3, 4).

Theorem 1 The sequence $\{sim_n\}_n$ is monotonically increasing.

Proof. Fix any (w_1, w_2). Denote

$$A_\cap = \sum_{(r,w)\in T(w_1)\cap T(w_2)} (I(w_1, r, w) + I(w_2, r, w))$$

and

$$A = \sum_{(r_w)\in T(w_1)} I(w_1, r, w) + \sum_{(r_w)\in T(w_2)} I(w_2, r, w)$$

Then

$$\text{lin\_sim}(w_1, w_2) = sim_1(w_1, w_2) = \frac{A_\cap}{A} \leq 1$$

Note that for each sim_n we have a different hidden relatedness $R_n \geq 0$.

$$\begin{aligned} sim_2(w_1, w_2) &= \frac{A_\cap + R_1(w_1, w_2)}{A + R_1(w_1, w_2)} \\ &\geq \frac{A_\cap}{A} \\ &= sim_1(w_1, w_2) \end{aligned}$$

Since (w_1, w_2) is arbitrary, this implies that $sim_2 \geq sim_1$.

Then by examining the definition of R, since $T : S \to R \times S$ is invariant throughout, we have $R_2 \geq R_1$.

Now fix any (w_1, w_2) again.

$$\begin{aligned} sim_3(w_1, w_2) &= \frac{A_\cap + R_2(w_1, w_2)}{A + R_2(w_1, w_2)} \\ &\geq \frac{A_\cap + R_1(w_1, w_2)}{A + R_1(w_1, w_2)} \\ &= sim_2(w_1, w_2) \end{aligned}$$

since $R_2 \geq R_1$.

Thus $sim_3 \geq sim_2$.

Repeat this argument inductively, we have that $\{sim_n\}_n$ is monotonically increasing. □

Theorem 2 The sequence $\{sim_n\}_n$ is point-wise convergent.

Proof. For each (w_1, w_2), by Theorem 1., $\{sim_n(w_1, w_2)\}$ is monotonically increasing.

Also by definition of the similarity, $sim_n(w_1, w_2) \leq 1 \; \forall n$.

Therefore, $\{sim_n(w_1, w_2)\}$ is a monotone, bounded sequence, and hence convergent. □

Denote sim_∞ to be the limit of $\{sim_n\}$.

Theorem 3 $sim_\infty(w_1, w_2) = 1$ if and only if $\text{lin\_sim}(w_1, w_2) = sim_1(w_1, w_2) = 1$

Proof. If $sim_1(w_1, w_2) = 1$, since sim_n is increasing and bounded by 1, clearly $sim_\infty(w_1, w_2) = 1$.

If $sim_\infty(w_1, w_2) = \lim_{n\to\infty} sim_n(w_1, w_2) = 1$, then plugging it in the iterative process, we have

$$\frac{A_\cap + R_\infty(w_1, w_2)}{A + R_\infty(w_1, w_2)} = 1$$

This implies

$$sim_1(w_1, w_2) = \frac{A_\cap}{A} = 1$$

□

Theorem 4 If $\text{lin\_sim}(w_1, w_2) = sim_1(w_1, w_2) < 1$, then $\{sim_n(w_1, w_2)\}$ is strictly less than a bound less than 1.

Proof. Write $T_= = \{(w, w') \in T(w_1) \times T(w_2) \mid w = w'\}$.

Since $sim_\infty(w_1, w_2) < 1$, $R_\infty(w_1, w_2) < T$, where $T = |T(w_1) \times T(w_2) - T_=|$.

And since sim_∞ is the limit,

$$\begin{aligned} sim_\infty(w_1, w_2) &= \frac{A_\cap + R_\infty(w_1, w_2)}{A + R_\infty(w_1, w_2)} \\ &< \frac{A_\cap + T}{A + T} \\ &< 1 \end{aligned}$$

□

4 Algorithm

Using the above similarity, we will generate a thesaurus based on a given corpus. This section will introduce any datasets or resources we use in Section 4.1 and our algorithm logics in 4.2. Results will be posted on Section 4.3.

4.1 Datasets, Interface, and Package

We start introducing our algorithm by first introducing the datasets, interfaces and packages that are used in our project. More details on how we utilized them are explained in next section.

Due to limited computation power, we use the first 10000 lines of the Wall Street Journal (WSJ) corpus (79325 lines) in Penn Treebank 2 as the main input file. This corpus contains numerous articles splitted by a tag `.START`, and paragraphs are splitted by an empty line.

In order to obtain the dependency parser in triple format $I(w, r, w')$ for calculating the mutual information $I(w, r, w')$ later (as discussed in Section 3), we apply the package Stanford Parser. The package is a Java implementation of probabilistic natural language parsers. The package includes extensive works such as internationalization and language-specific modeling, flexible input/output, grammar compaction, lattice parsing, k-best parsing, and typed dependencies output. Our project mainly uses the method `dep_parser.parse()`, which takes a sentence as input and analyzes and generates dependency triples, and enable users to extract triples each in format `(u'jumps', u'VBZ'), u'nsubj', (u'fox', u'NN')`. These triples are later used to calculate mutual information and similarity as described in Section 3.

Before calculating the mutual information, we lemmatize words in the extracted triples by applying `WordNetLemmatizer` in WordNet interface. This allow us to avoid some ambiguities and redundancies while dealing with the same verb in different tenses and the same nouns in single and plural forms.

4.2 Algorithm

Our algorithm iteratively computes and utilizes similarities. The number of iteration is up to choice. Since in Section 3 we have proved its convergence, we can use the Elbow method to determine an optimal iteration number N, i.e. when the similarity improvements noticeably decreases. This number N is subject to the input corpus. The main logic of our algorithm goes as follows:

1. For a given text, we first parse the sentences into dependency triples (w, r, w'). We utilize the Stanford `CoreNLPDependencyParser` package from `nltk` package to do the dependency parsing.

2. Using the dependency triples (w, r, w'), we compute statistically the mutual information $I(w, r, w')$ and $T(w)$; refer to Lin [1998].

3. We set our sim_0 to be zero in the first place.

4. We then use these information to compute $sim(w_1, w_2)$ with the formula proposed in Section 3.1.3.

5. A similarity dictionary is built to record similarity of every pair of word with nonzero similarity for each sim_n.

6. Repeat step 4-6 by plugging back sim_n for sim_0.

7.After N iterations, we arrive at a final similarity dictionary with keys being w_1 and the corresponding value being a dictionary of key-value pairs $(w_2, sim(w_1, w_2))$.

8. For each w_1, we sort $sim(w_1, w_2)$ to get the top N similar words of w_1.

4.3 Experiments

We take the first 10000 lines in the training corpus (out of 79325 lines) as an experimental input to test our algorithm. The algorithm generates output files that are consistent with our expectation, given a relatively small input corpus. So far, to observe its performance, we have generated two output files. One contains the non-zero word similarities among all nouns to each other, sorted in descending order for each noun. The other output file specifically lists out the "top k" similar nouns in descending order for each noun by truncating the previous one.

For clarification, we provide four example outputs of our program here. The following Figure 1, 2, 3, 4 are derived by 10000 lines training corpus and iteration number $N = 6$.

vinken	agnew	0.5073050358188422
vinken	reupke	0.50439154580673
vinken	gringo	0.4732048584566266
vinken	duel	0.4732048584566266
vinken	stevens	0.4705212144274621
vinken	vitulli	0.4307633838667834
vinken	donovan	0.4296284961108692
vinken	tashi	0.3973007324549207
vinken	stearn	0.3938544101452426
vinken	hollander	0.38799803440805386

Figure 1: Top 10 similar words and the corresponding word similarities of word "Vinkin" (a person's name)

year	month	0.3086544326331497
year	week	0.25655660886069914
year	day	0.2110443617058008
year	quarter	0.16752071565302032
year	time	0.14517231309132667
year	point	0.13893232152381335
year	state	0.13390333912424424
year	part	0.13368348439345243
year	country	0.13112217560232556
year	company	0.12478135057856683

Figure 2: Top 10 similar words and the corresponding word similarities of word "year"

death	hazard	0.20986388933587716
death	senate	0.19992100936890028
death	ten	0.18522237086711144
death	demise	0.18241279561869483
death	marketplace	0.16238157168773393
death	east	0.158266992729269
death	category	0.15591546783881682
death	cancer	0.15556474771789044
death	sec	0.15298802343363657
death	chamber	0.14933840026790923

Figure 3: Top 10 similar words and the corresponding word similarities of word "death"

result	part	0.17162766337584856
result	earnings	0.15772578637427626
result	boost	0.1569044576877159
result	rise	0.1416885872398846
result	trade	0.1373013577017365
result	profit	0.1370311544761754
result	fall	0.13683606227983086
result	drop	0.13479048511021122
result	loss	0.13071776959146877
result	subject	0.12810203676240747

Figure 4: Top 10 similar words and the corresponding word similarities of word "result"

5 Evaluation

In this section, we evaluate our algorithm by comparing the thesaurus our algorithm generated to that in WordNet, and we also compare our result with Lin's result [1997]. Note that we choose WordNet because it is handcrafted, widely acknowledged, and readily tested by many others.

5.1 Thesaurus File of Algorithm

Similar to Figure 1., for any word w_i, our algorithm sorts the similarity between w_i and all the other words, i.e., $sim(w_i, w_j)$ with $i \neq j$, in descending order. Then we store the "top 100" most similar words of each w_i, and let the stored file be our thesaurus file. In other words, the thesaurus file contains top 100 most similar words of each word, along with their similarities. Note that above procedures are all done after the last iteration finishes.

5.2 Thesaurus File of Wordnet

For the Wordnet, since it does not contain numerical values to represent the similarity between words, we first define a formula to compute word similarity between w_1 and w_2 as follows, based on the structure of Wordnet and Lin's paper [1998].

$$\max_{c_1 \in S(w_1) \wedge c_2 \in S(w_2)} \left(\max_{c \in super(c_1) \cap super(c_2)} \frac{2 \log P(c)}{\log P(c_1) + \log P(c_2)} \right) \tag{1}$$

where $S(w)$ is the set of senses of w in WordNet, $super(c)$ all superclasses of c, and $P(c)$ its sense probability (to be explained later).

And let $sim_{wn}(w_1, w_2)$ denote the value computed in Eq (1).

In Eq (1), $S(w)$ can be retrived by command `.synsets(w)`, and $super(c)$ can be obtained using recursion and command `.hypernyms(c)`. $P(c)$ denotes the probablity that a random selected noun is an instance of c which is a synset. Lin approximates this value by the frequency of concepts in SemCor [Miller et al, 1994], which is a sense-tagged subset of Brown corpus. Here instead, based on the definition and intention of $P(c)$, we define its formula as follows.

$$P(c) = \frac{\text{frequency of words in synset } c \text{ and its sub-classes}}{\text{frequency of all words with same POS}} \tag{2}$$

A few comments are made on the formula in Eq (1). First, note that the similarity is between 0 and 1. Since we pick $c \in super(c_1) \cap super(c_2)$, c must be some super class of both c_1 and c_2. Given that the frequency of words with same POS is fixed, words in c_1 and c_2's sub-classes must be

in sub-classes of c as well, and therefore the nominator (in Eq.2) of $P(c)$ is larger than the nominator of $P(c_1)$ and $P(c_2)$. This implies $P(c) > P(c_1)$ and $P(c) > P(c_2)$. Since $P(*) \leq 1$, $\log P(*) \leq 0$. Thus $|\log P(c)| < |\log P(c_1)|$ and $|\log P(c)| < |\log P(c_2)|$, which leads to the result $\frac{2\log P(c)}{\log P(c_1) + \log P(c_2)} \leq 1$ under all circumstances.

We then do similar processing for the computed WordNet similarity to get a thesaurus. For any word w_i, we sort the similarities $sim_{wn}(w_i, w_j)$ for all $j \neq i$ in descending order, and store the top 100 most similar words. Denote the file as thesaurus file of WordNet.

And we are now ready to define the similarity between thesaurus, which is the main standard for our evaluation.

5.3 Similarity between Thesaurus

Based on discussion in Section 5.1 and Section 5.2, we have thesaurus files for our algorithm and Wordnet in format as follows:

$$w : w_1, s_1; w_2, s_2; ...; w_{100}, s_{100}$$

$$w' : w'_1, s'_1; w'_2, s'_2; ...; w'_{100}, s'_{100}$$

The first line represents the thesaurus file of our algorithm, and the second line represents the thesaurus file of Wordnet, with the top 100 most similar words and their corresponding similarities. Their similarity of each word is defined (similarly to cosine similarity) as:

$$\frac{\sum_{w_i = w'_j} s_i s'_j}{\sqrt{(\sum_{i=1}^{100} s_i^2)(\sum_{j=1}^{100} s_j'^2)}} \tag{3}$$

And we define the similarity of two complete thesaurus by averaging the similarity of each pair between the two thesaurus files.

5.4 Algorithm for Evaluation

Our goal for this section is to compare any two thesauri using WordNet as our baseline. Using the measure in Section 5.3, we first define the similarity $sim(S, S')$ for two thesauri respectively on some words S and on potentially different words S':

1. For each common word w in both S, S', retrieve for w the top 100 similarity pairs (w_i, s_i) and (w'_i, s'_i).

2. Compute the cosine similarity for the word w using Eq. (3).

Iteration Step	Thesaurus Similarity
1	0.018953149630575986
2	0.020378508309906115
3	0.020583381498826034
4	0.02062454294707912
5	0.020630286749300116
6	0.02064214902718897
7	0.0206356120734704
8	0.020635621867417662
9	0.02063562375366163
10	0.020635624117043352
11	0.02063562418705119
12	0.020635624200538886
13	0.020635624203137387
14	0.02063562420363805
15	0.020635624203734514
16	0.020635624203753086
17	0.020635624203756676
18	0.02063562420375735
19	0.020635624203757506
20	0.020635624203757533

Table 1: Similarity between two thesaurus files for each iteration step

3. Compute the average of all those $sim(w)$ to arrive at a similarity between the two thesauri on S and S' respectively.

Let W be the thesaurus generated by WordNet. To evaluate performance of S given S', we simply compare $sim(S, W)$ and $sim(S', W)$.

5.5 Evaluation Result of Experiments

We printed out the similarity between two thesaurus files for each iteration. In our test experiment, we use 20 iterations, and the results indicate that indeed the word similarities increase and converge in general. The results are shown in Table 1.

Since the value range is relatively small, we include a plot of the thesaurus similarity versus number of iteration in Fig. 5.

In order to provide a clearer picture for the iteration in our algorithm, we store the results of similarities between five pairs for 5 iterative steps. The results are shown in Table 2-6. The superimposed plot is in Fig. 6.

From the experiment results, theories introduced in Section 3 are proved to be sound and reasonable. First note that by Fig. 6 and Table 2-6, we can observe that the word similarities in general indeed increase and converge as more iterative steps con-

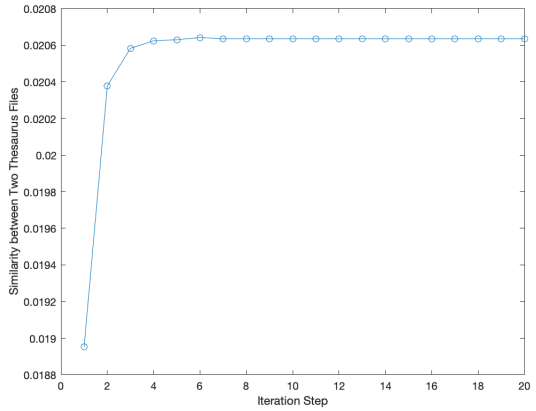

Figure 5: Similarity between two thesaurus files of 20 iteration steps

Table 2: Similarity between **'year'** and **'month'**

Iterative Steps	Word Similarity
1	0.301543724147978
2	0.30729926584770195
10	0.308656342087921
19	0.30865634472423276
20	0.3086563447242336

Table 3: Similarity between **'director'** and **'president'**

Iterative Steps	Word Similarity
1	0.130992536569098768
2	0.14637143585668116
10	0.1479397849645095
19	0.14793978739829378
20	0.14793978739829447

Table 4: Similarity between **'nov.'** and **'dec.'**

Iterative Steps	Word Similarity
1	0.17041804116039275
2	0.17041804116039275
10	0.17041804116039275
19	0.17041804116039275
20	0.17041804116039275

Table 5: Similarity between **'chairman'** and **'president'**

Iterative Steps	Word Similarity
1	0.16306673259912902
2	0.17023895707519487
10	0.1707493734139971
19	0.17074937414239982
20	0.17074937414240007

Table 6: Similarity between **'death'** and **'hazard'**

Iterative Steps	Word Similarity
1	0.20986388933587716
2	0.20986388933587716
10	0.20986388933587716
19	0.20986388933587716
20	0.20986388933587716

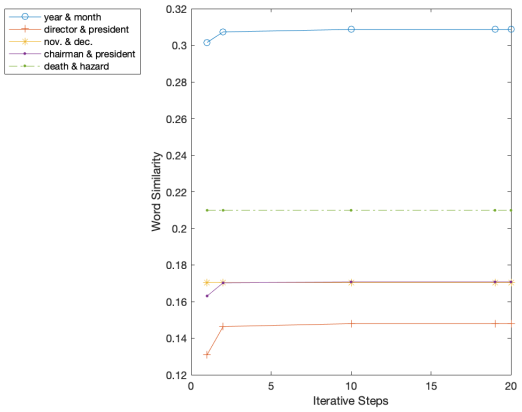

Figure 6: Word Similarities for six pairs in each of 5 iterations of our algorithm

ducted. In three among five cases (i.e., Table 1, 2, 4), the similarity between two certain words first increase relatively fast, from the first iteration to the second, and then the increasing speed decreases a lot, which is almost a horizontal line in the plot. Two among five cases remain the same starting from the first iteration. This is possible as well under the situation that: within the test experiment file, no information is lost at the very first similarity computation step, and therefore the second iteration does not recapture any information, which keeps the similarity being the same throughout the 20 iterations.

Second, from the Table 1 and Fig. 5, we can observe that the similarity between two thesaurus files increases as more and more iterative steps are conducted. Similar to the single word similarity computation, the thesaurus similarity first increases relatively fast, and then the converging speed decreases a lot, but is indeed still increasing. Combining this observation with the discussion and conclusion in previous paragraph, we are able to conclude that the similarities computed by our algorithm not only increases and converges, but in effect is getting closer to the result of WordNet thesaurus. In other words, the accuracy is improved

with more iterative steps conducted.

We then compare our algorithm performance to Lin's [1998]. An important point to note here is that our algorithm intends to enhance Lin's algorithm by implementing iterations on Lin's original algorithm and collected the hidden relatedness. As shown in Section 3.2, the running results of the first iterative step of our algorithm, i.e. when sim_0 is set zero, is exactly the results of Lin's algorithm on our input data sets file. Note that from both Table 1 and Fig.5, we can conclude that the thesaurus similarity is improved after iteration 1. Besides, the enhancement is relatively prominent. This implies that our algorithm indeed enhances Lin's algorithm [1998], and with more iterative steps conducted, the accuracy is improved further.

6 Discussion

In this section, we will discuss the potential improvement on using larger input corpus and an observation on our project about the essential information-based similarity formula.

6.1 Impact of Data Insufficiency

As previously mentioned, we are only using 10000 lines out of 79325 due to computation limitation. This reduction in training data can have noticeable impacts on the results of our algorithm. We demonstrate such impacts via comparing the word "director" in the results of running on 2000 lines and on 10000 lines respectively, each for 1 iteration and for 6 iterations.

```
director → fledgling → 0.171510496683982
director → student → 0.15587454150433794
director → producer → 0.15428984678373062
director → consultant → 0.15426269676413185
director → pirate → 0.15246578149943432
director → result → 0.15219332177942038
director → symbol → 0.1495592111404607
director → valuation → 0.14880176738594597
director → presence → 0.13363473143092106
director → amphobiles → 0.12006116138522523
```

Figure 7: Top 10 similar words of the word "director" on 2000-line input after 1 iteration

First, by comparing Fig. 7 and Fig. 9, we can see that increasing the input from 2000 to 10000 lines makes the resulted similar words of "director" much closer to our intuition. Therefore, the size of input training data is crucial to the results of our thesaurus generation algorithm, as well as to many other algorithms.

```
director → fledgling → 0.171510496683982
director → student → 0.16145724573386513
director → producer → 0.1570656450527005
director → consultant → 0.15426269676413185
director → pirate → 0.15246578149943432
director → result → 0.15219332177942038
director → symbol → 0.1495592111404607
director → valuation → 0.14880176738594597
director → presence → 0.13363473143092106
director → amphobiles → 0.12006116138522523
```

Figure 8: Top 10 similar words of the word "director" on 2000-line input after 6 iteration

```
director → president → 0.13099253656909876
director → executive → 0.1286845927391187
director → customer → 0.1074892413031651
director → chairman → 0.10539311522090195
director → partner → 0.1030464035921823
director → official → 0.09927767491073608
director → manager → 0.09895522734737462
director → unit → 0.09609172972710912
director → subsidiary → 0.09530979174780745
director → board → 0.09349506926872826
```

Figure 9: Top 10 similar words of the word "director" on 10000-line input after 1 iteration

```
director → president → 0.14793801777798546
director → executive → 0.1446999319192283
director → official → 0.12601828841472681
director → partner → 0.11794341232571139
director → chairman → 0.1173472522109398
director → manager → 0.11321250390334575
director → customer → 0.11300062221398878
director → unit → 0.11162522859820669
director → subsidiary → 0.11158494173038308
director → stock → 0.1093573149089609
```

Figure 10: Top 10 similar words of the word "director" on 10000-line input after 6 iterations

Second, by comparing Fig. 7 and Fig. 8, Fig. 9 and Fig. 10, we can observe that a larger size of the input enables our algorithm to better collect the hidden relatedness. From Fig. 7 to Fig. 8, only little changes have been made by the 6 iterations. Yet from Fig. 9 to Fig. 10, we have changes in both the numerical similarities and the order. This is intuitive, as more occurrences of a word lead to more words in its neighborhood, and hence more hidden relatedness to be excavated, and more work the iterations can do.

6.2 Information and Frequency

On working with the formulas based on information theory, we come up with a question on a potential constraint of this approach. In Lin's earlier paper [1997], he defines abstractly the similarity to

be

$$sim(A, B) = \frac{\log P(common(A, B))}{\log P(describe(A, B))}$$

, where $common(A, B)$ is a measure of the commonality of the words A and B, given a corpus, and $describe(A, B)$ is a measure that describes A and B. One example of $describe(A, B)$ may be $T(A)$ and $T(B)$ as defined in Section 3.1.2. And $common(A, B)$ may be the intersection of them, as used by Lin [1998].

Now note that if $T(B)$ is much smaller than $T(A)$, the intersection is limited by $T(B)$. Consequently, $common(A, B)$ is largely decreased due to the rarity of B. Speaking intuitively, even if A and B have almost the same meaning, and if A is daily used while B is elliptical and rarely used, then their computed similarity can turn out to be small. In other words, this similarity may take into account the "natural" frequencies of the words too (the frequency that a word appears in a random position of a randomly given corpus). Note that if this conjecture holds, then this bias is not erasable by simply increasing the size of the input corpus.

The above is only a conjecture rather than a rigorous theoretical analysis of this information-based approach. By examining the results of our algorithm, we can find some evidences that may validate this conjecture. For example, in the similarity computed after 20 iterations on the 10000 lines input, we see:

Table 7: Relationship between one's similarity with "japan" and occurrence for a part of words

word frequency	similarity
germany 25	0.28466511250869053
england 25	0.25910977094877824
france 18	0.25590063507356364
poland 13	0.19180103956042388
italy 10	0.15298197762041638
finland 5	0.1265313375811157

We choose "japan" as the object to compare with because it has the frequency 311 in the input corpus, relatively high to others. Intuitively, we do not expect that the similarity between Japan and Germany is almost two times higher than Japan and Italy, and even more than two times higher than Japan and Finland. This table may have illustrated a proportional increment between frequency and similarity, in the comparison with a frequent word.

However, note that there are counterexamples for sure. In particular, the frequency of Korea is 37 in the input, and yet the similarity between Korea and Japan is 0.17126459856398157, which is even lower than Poland.

7 Future Work

For the remaining goals to fulfill in this project, we would first use the entire training file to run the program, instead of parts of it.

Furthermore, it is worth studying the tree structures that can be generated by our thesaurus. With a computed similarity, we can build a tree that sheds light on the inheritance of the senses of words. For instance, if "bug" is similar to both "error" and "insect", then in the tree "bug" has "error" and "insect" as its children because each of them inherits a sense of word of "bug". This may be achieved using the hierarchical clustering.

Another goal we wish to accomplish is to cluster all nouns in a file in an unsupervised way. The objective is to generate generalized synsets based on our similarity. These generalized synsets are not classified by specific sense of word but rather the general closeness. Toward this purpose of clustering, we may need to alter our similarity to play the role of a distance function.

References

[1] Adam Kilgarriff. 2003. Thesauruses for Natural Language Processing.

[2] Claudia Leacock, and Martin Chodorow. 1998. Combining local context and WordNet similarity for word sense identification. In *Fellbaum, C., ed., WordNet: An electronic lexical database*. MIT Press. 265–283.

[3] Christiane Fellbaum. 1998. WordNet: An Electronic Lexical Database. The MIT Press, Cambridge, MA

[4] Dekang Lin. 1997. Using Syntactic Dependency as Local Context to Resolve Word Sense Ambiguity. In *Proceedings of ACL/EACL-97*, pages 64-71, Madrid, Spain, July.

[5] Dekang Lin. 1998. Automatic Retrieval and Clustering of Similar Words.

[6] Donald Hindle. 1990. Noun Classification from Predicate-argument Structures. In *Proceedings of ACL-90*, pages 268-275, Pittsburg, Pennsylvania, June.

[7] George A. Miller, Martin Chodorow, Shari Landes, Claudia Leacock, and robert G. Thomas. 1994. Using a Semantic Concordance for Sense Identification. In *Proceedings of the ARPA Human Language Technology Workshop*.

[8] Hinrich Schutze, Jan O. Pedersen. 1997. A Cooccurrence-based Thesaurus and Two Applications to Information Retrieval. Inform. Proc. Manage. 307–318.

[9] Jay J. Jiang, David W. Conrath, Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy, 1997.

[10] Philip Resnik. 1995. Using Information Content to Evaluate Semantic Similarity in a Taxonomy. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence, Vol. 1*, pages 448-453, Montreal, August 1995.

[11] Roy Rada, Hafedh Mili, E. Bicknell, and M. Bletner. 1989. Development and Application of a Metric on Semantic Nets. In *IEEE Transactions on Systems, Man, and Cybernetics, Vol. 19, No. 1*, pages 17-30.

[12] Ted Pedersen, Siddharth Patwardhan, Jason Michelizzi. 2004. WordNet::Similarity - Measuring the Relatedness of Concepts.