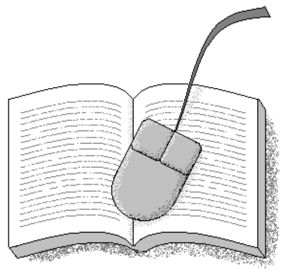
INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN
LABORATORIO DE LENGUAJE NATURAL Y
PROCESAMIENTO DE TEXTO



Word sense disambiguation through associative dictionaries

TESIS QUE PRESENTA

M. en C. Francisco Viveros Jiménez

PARA OBTENER EL GRADO DE
DOCTOR EN CIENCIAS DE LA COMPUTACIÓN

DIRECTORES:

Dr. Alexander Gelbukh
Dr. Grigori Sidorov

México, D. F., Junio 2014

SIP-14 BIS

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de México, D.F. siendo las 12:00 horas del día 4 del mes de Junio de 2014 se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación del:

Centro de Investigación en Computación

para examinar la tesis titulada:

"Word sense disambiguation through associative dictionaries"

Presentada por el alumno:

VIVEROS	**JIMÉNEZ**	**FRANCISCO**
Apellido paterno	Apellido materno	Nombre(s)

Con registro: | B | 1 | 0 | 2 | 2 | 5 | 3 |

aspirante de: **DOCTORADO EN CIENCIAS DE LA COMPUTACIÓN**

Después de intercambiar opiniones los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Directores de tesis

Dr. Alexander Gelbukh

Dr. Grigori Sidorov

Dr. Francisco Hiram Calvo Castro

Dr. Marco Antonio Moreno Ibarra

Dr. Miguel Jesús Torres Ruiz

PRESIDENTE DEL COLEGIO DE PROFESORES

INSTITUTO POLITÉCNICO NACIONAL
CENTRO DE INVESTIGACION EN COMPUTACION
DIRECCION

Dr. Luis Alfonso Villa Vargas

INSTITUTO POLITÉCNICO NACIONAL
SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, D.F. el día 12 del mes de Junio del año 2014, el que suscribe Francisco Viveros Jiménez alumno del Programa de Doctorado en Ciencias de la Computación, con número de registro B102253, adscrito(a) al Centro de Investigación en Computación, manifiesto que es el autor intelectual del presente trabajo de Tesis bajo la dirección del de los Dr. Alexander Gelbukh y Dr. Grigori Sidorov y cede los derechos del trabajo titulado Word sense disambiguation through associative dictionaries, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a las siguientes direcciones pacovj@hotmail.com. Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Francisco Viveros Jiménez

Nombre y firma del alumno

Resumen

La desambiguación de sentidos es una tarea útil para el procesamiento del Lenguaje Natural. Existen muchos métodos para la desambiguación de sentidos, pero a la fecha no se ha encontrado una solución perfecta. En este trabajo proponemos tres mejoras al proceso de desambiguación que son aplicados a varios métodos del estado del arte con buenos resultados.

Las mejoras consisten en lo siguiente:

1. **Filtrar algunas palabras del contexto.** Las palabras que se toman del contexto deben ser: únicas, diferentes a la palabra objetivo y útiles para la desambiguación
2. **Usar coocurrencias extraídas automáticamente.** Agregamos a las definiciones de las acepciones las palabras que coocurren en los textos con las palabras de las definiciones originales.
3. **Identificar las palabras que se resolverán exitosamente.** Las palabras que tienen un sentido por discurso se resuelven exitosamente por muchos métodos. Estás palabras representan la mitad del total a resolver.

Usted encontrará en cada capítulo de esta tesis una explicación detallada sobre el por qué y el cómo del funcionamiento de nuestras mejoras. También encontrará experimentos que confirman que estas mejoras son combinables entre sí y que llevan a algunos algoritmos a alcanzar una precisión cercana a la de las personas.

Abstract

The Simplified Lesk Algorithm is frequently employed for word sense disambiguation. It disambiguates through the intersection of a set of dictionary definitions (senses) and a set of words extracted of the current context (window). However, the Simplified Lesk Algorithm has a low performance. This work shows some improvements for increasing this (and some other knowledge-based methods' performance).

We propose the following changes:

1. **Changing the window selection procedure**. Window selection must: (1) search in the whole document instead of the words around the target word, (2) exclude duplicates and the target word from the window, and, (3) include words that lead to an overlap with any sense of the target word.
2. **Extending sense definitions with co-occurring words.** We add to the sense definitions words that co-occur with those words that are in the original sense definitions.
3. **Dsiambiguating only domain words.** We exclude non-domain (mostly functional or too general) words from consideration, boosting precision at the expense of somewhat lower reall.

Our work presents experiments for each proposed modification working separately, and, finally, a demonstration that confirms that all these modifications can work together for further performance improvement. Then, we test the integration of our modifications with some other dictionary-based methods. All experiments were carried out on Senseval-2 and Senseval-3 English-All-Words test sets.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Glossary

Bag of words Set of lemmas representing a text.

Coverage Measure indicating the amount of words a system disambiguates (100% coverage means that all words were disambiguated). However, coverage does not measure the quality of the answers.

Dictionary Database linking a lemma to definitions, samples and semantic data.

Gold standard Set of sample sentences including a list with the correct senses for some words.

Lemma A word as written in the dictionary.

Lemmatizer A program that transform words into lemmas.

POS tag Tag that tells you the word class. E.G N for Noun, V for Verb and R for Adverb

Semantic data Data defining relations of a given word. Common semantic data includes synonyms, antonyms and related terms.

Sense An entry in the dictionary for a given word. Each sense has a definition, examples, and semantic data.

Word sense disambiguation (WSD) Task consisting of choosing the meaning of a given word.

Chapter 1. Introduction

Words can have various meanings depending on their contexts, as in “I am an excellent **bass** player” or in “The **bass** got away from my fishing rod”, the word **bass** has two different meanings. Such meanings are usually located as different senses of the words in dictionaries. Word sense disambiguation (**WSD**) is the task of choosing automatically an appropriate sense for a given word (called target word) in a text (document) out of a set of senses listed in a dictionary (called sense inventory).

This chapter answers the following questions:

- What is WSD about?
- What is the scope of our work?
- What were our contributions?
- Why are they important?
- How did we reach these contributions?

1.1 Word Sense Disambiguation (WSD)

WSD is a complex task that is not useful by itself. WSD is important because it is a key task to other natural language processing tools such as:

- **Machine Translation.** Translate “pension” from English to any other language. Is it an “small hotel” or a “retirement benefit”
- **Information retrieval.** Find all the web pages about “CAT”, is it a “company” or an “animal”?
- **Location finding.** Find the city of Valencia: Venezuela or Spain?
- **Question Answering.** What is Paul Simon’s position on global water shortages? The politician or the singer?
- **Knowledge Acquisition.** The ball is made of leather. A spherical object or a dancing event?
- **Corpus tagging.**

Those tools are often used for assisting people in fully automatic or semi-supervised ways. The quality and coverage of WSD depends on the goals of the end system. For example, if you want to create a system that assist you in translating a document for a language that you already know, the user would be happy with a system that perfectly translate a few whole sentences and leaves to you the ones that it does not properly translate. In the other hand, suppose that you do not know the other language at all. It is preferable to have a system giving you complete translation with a somehow acceptable quality.

Discussion about quality is important because one of our findings transforms some WSD systems into high quality/medium coverage systems.

1.2 Scope

Word sense disambiguation methods usually work by extracting information from different knowledge resources and scoring the senses with these resources through different means. WSD methods can differ in the way they score a sense, the way they select its target words, and the way they load the knowledge resources. Generally speaking, if you improve one of these areas you usually improve several WSD methods sharing the original procedure. For example, if you have a definition such as:

Drink: "a single serving of a beverage; I asked for a hot drink; likes a drink before dinner"[1]

This definition is rather short for WSD algorithms. We believe it is also a little short for people learning English. So, if we extend that definition somehow to obtain a better one such as:

Drink: "Drinks, or beverages, are liquids specifically prepared for human consumption. In addition to basic needs, beverages form part of the culture of human society. Despite the fact that most beverages, including juice, soft drinks, and carbonated drinks, have some form of water in them; water itself is often not classified as a beverage, and the word beverage has been recurrently defined as not referring to water."[2]; "a single serving of a beverage; I asked for a hot drink; likes a drink before dinner";

This new definition is a lot clearer and more useful for WSD algorithms.

This research describes three novel improvements that are usable with several WSD methods. These improvements add some changes to the target word selection and knowledge resources loading procedures.

1.3 Main Contributions

We have contributed to WSD and additionally some contributions are usable into global numeric optimization. The scientific contributions can be summarized as follows:

- A novel window selection method.
- A procedure for extending definitions with co-occurring words.

[1] from WordNet

[2] from Wikipedia

- A procedure for identifying domain words (as we later confirm they are easier for WSD).

We also confirmed that our improvements are fully compatible with several bag-of-word methods including machine learning. Testing confirms that they bring a considerable performance boost.

We noted that Lesk algorithm take too much time and resources to complete its task. Therefore, we developed a novel optimization technique that obtains the same results requiring half of the time. We also tested this novel optimization technique in numerical optimization with good results.

We have done the following software products:

- Java WordNet connector.
- Java API for WSD.
- Gannu, a library for some NLP tasks.

They are all available in our website http://fviveros.gelbukh.com as free software for using as specified in the GNU license.

1.4 Methodology

Our research was accomplished by performing the following tasks:

1. Implementation of pre-processing software. This software has the following capabilities:
    a. Loading dictionary data from WordNet.
    b. Loading gold standard data in SemCor format.
    c. Filtering out non open-class words.
2. Implementation of benchmarking software. This software has the following capabilities:
    a. Measuring performance through precision, recall, coverage and F-measure.
    b. Generating data sheets containing detailed data for behavior analysis.

c. Comparing with other dictionary-based methods. This SW contains implementations of several dictionary-based methods.

3. Implementation.
    a. Implementing the proposed window selection.
    b. Implementing the proposed word filter.
    c. Implementing the proposed method for extending bag-of-words.
    d. Testing all the aforementioned ideas within the Lesk algorithm.
        i. Implementing the proposed optimization heuristic.
4. Analysis of the experimental results.

1.5 Organization of the document

This thesis is organized as following.

Chapter 2 contains basic knowledge for understanding WSD and the current methods that are being used.

Chapter 3 contains the description of methods and practices used for constructing our implementation. Also, it contains the description of the WSD methods used being improved by our proposed methods.

Chapters 4 to 6 contain the description and testing of the three proposed modifications. However, due to the somehow independent nature of each modification, each chapter contains each own state of the art, description and analysis subsections. In this way, the reader does not need to scroll through the document for understanding some specific contribution.

Chapter 4 contains the description of our window selection method that consists on looking for words producing overlaps and avoiding the target word and duplicates.

Chapter 5 shows the description of our filter for selecting easy words. You will confirm that words having one sense per discourse are easier for WSD methods. Hence, our filter leads to solve half of the words with good quality (i.e. a precision above 70%).

Chapter 6 depicts the use of co-occurring words for WSD. You will discover that using these words for extending the original definitions leads to a good precision boost (around 10%).

Chapter 7 shows an improvement to Lesk algorithm. We developed a new optimization heuristic that reduces the exhaustive time needed by the Lesk algorithm. This heuristic is also useful for global optimization purposes.

Chapter 8 concludes this thesis.

Appendix gives some details about our API.

Chapter 2. State of the Art

Word sense disambiguation is an open problem. There are many approaches that try to accomplish it. Approaches range from simple heuristics such as choosing the most frequent sense to machine learning.

This chapter answers the following questions:

- Why is WSD an open problem?
- How to measure WSD?
- What are supervised WSD methods and how do they work?
- What are dictionary-based WSD methods and how do they work?
- What is a back-off strategy and how much influence does it have?

2.1 Difficulties of WSD

Researchers have identified the following difficulties for the WSD task:

- Discreteness of the senses.
- Differences between dictionaries.
- Amount of samples and semantic knowledge available.

The amount of samples and semantic knowledge available can be solved by manually increasing them. However doing it is usually costly and undesirable. So, doing these automatically or using fewer resources is the normal way of proceeding.

Discreteness of the senses deals with the level of distinction that a sense should have for considering it a different sense of a word. The concept of word sense is controversial, causing disagreements among lexicographers around what should be considered a different word sense and what not. Researchers define two levels of discreteness of the senses: coarse-grained and fine-grained.

The coarse-grained level deals with homographs. A homograph is a word that shares the same written form of another word with different meaning. Examples of homographs are: bass (music instrument/fish), pen (writing instrument/enclosure) and pension (boarding-house/salary in retirement). Most of the homographs are easily distinguished by humans. WSD accuracy at the coarse-grained level in English is currently around 90%.

In the other hand, the fine-grained level is difficult even for humans. For example, the noun paper has seven senses in WordNet 3.1 (Miller 1995), eight senses in Merriam Webster online and five senses in the Cambridge dictionary. Lexicographers often disagree in the number of meanings of words (Kilgarriff 1997). In Senseval-2, human annotators only agreed in 85% of word occurrences (Edmonds 2000). WSD accuracy at the fine-grained level in English is currently around 65%.

Let's confirm the difficulty of the fine-grained level with an example. Table 1 shows the senses for the noun *paper* extracted from WordNet 3.1. The lexicographers define seven different senses. However, in our opinion, only the first three senses are necessary. How many senses do you believe necessary?

Table 1 Senses of the noun paper extracted from WordNet

Sense	Definition
1	paper (a material made of cellulose pulp derived mainly from wood or rags or certain grasses)
2	composition, paper, report, theme (an essay (especially one written as an assignment)) "he got an A on his composition"
3	newspaper, paper (a daily or weekly publication on folded sheets; contains news and articles and advertisements) "he read his newspaper at breakfast"
4	paper (a medium for written communication) "the notion of an office running without paper is absurd"
5	paper (a scholarly article describing the results of observations or stating hypotheses) "he has written many scientific papers"
6	newspaper, paper, newspaper publisher (a business firm that publishes newspapers) "Murdoch owns many newspapers"
7	newspaper, paper (the physical object that is the product of a newspaper publisher) "when it began to rain he covered his head with a newspaper"

As we stated previously, different dictionaries and thesauruses will provide different divisions of words into senses. WSD accuracy is tightly coupled to the used dictionary. Most of the researches choose a particular dictionary disregarding the fact that the selected dictionary is not perfect. Research results using coarse-grained dictionaries have been much better than those using fine-grained ones (Navigli et al. 2007, Pradhan et al. 2007).

Most of the WSD research use WordNet as a reference sense inventory for English. Other sense inventories used are Roget's Thesaurus (Yarowski 1992) and Wikipedia (Mihalcea 2007).

2.2 Measuring Performance of WSD

The most used metrics for evaluating the performance in WSD are (Navigli 2009): precision (**P**), recall (**R**), coverage (**C**) and F1-measure (**F1**) (harmonic combination of precision and recall). The corresponding equations are the following:

$$P = \frac{correct\ answers\ provided}{total\ answers\ provided} \tag{1}$$

$$R = \frac{correct\ answers\ provided}{total\ answer\ expected} \tag{2}$$

$$C = \frac{total\ answers\ provided}{total\ answer\ expected} \tag{3}$$

$$F1 = \frac{2PR}{P+R} \tag{4}$$

The most used test sets for English are: Senseval-2 (Cotton et al. 2001), Senseval-3 (Mihalcea and Edmonds 2004), Semeval-2007 (Navigli et al. 2007, Pradhan et al. 2007) and Semeval-2010 (Agirre et al. 2010).

All tests were carried out using Senseval-2 (Cotton et al. 2001)and Senseval-3 (Mihalcea and Edmonds 2004)test sets. We used WordNet 3.0 as sense repository. Stanford POS tagger (Toutanova and Manning 2000) was employed for tagging WordNet glosses.

2.2.1 WordNet

WordNet is an English lexical database freely and publicly available for download[3]. Open-class words are grouped into sets of synonyms called synsets. Synsets represent different concepts. WordNet is also a semantic network with semantic and lexical relations between synsets. Each synset have a short definition called **gloss** and some samples of its use. Figure 1 shows a sample semantic network for the first sense of the noun game.

[3]http://wordnet.princeton.edu/wordnet/download/

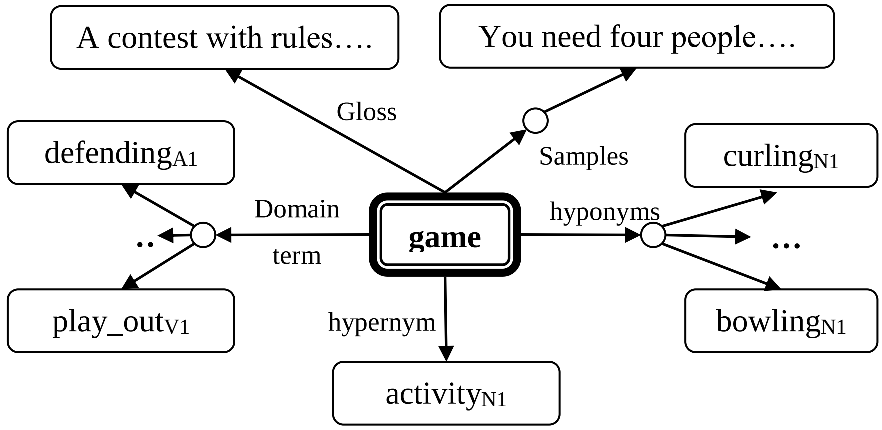


Figure 1 Sample semantic network for the first sense of the noun game

WordNet is the most commonly used lexical resource for the WSD task. Also, it has connectors available in several programming languages for its use.

2.3 Machine Learning for WSD

State-of-the-art approaches are commonly classified into two classes: supervised (machine learning) and dictionary-based. Supervised approaches usually see the WSD task as a classification problem. Classification is the problem of selecting a category for a new observation. Classification problem has the following elements:

1. **Categories**. Categories are the possible classes in which an observation should be assigned. For example, when classifying e-mails you will have the spam and good categories. Categories are pre-defined before defining everything else.
2. **Features**. Features are the values used for determining is an observation belongs to a class or another. For example, when classifying e-mails you can use the presence of certain words or phrases like you have won for deciding if a new observation belongs to a class or not. Features can have any type of value even categorical values.
3. **Training data**. The training corpus is a set of examples used for the algorithm for learning how to classify new instances.

In WSD, the classes are the senses extracted from the dictionary; the features are words in the context; and, the training data is a manually tagged corpus such as SemCor. Dictionary-based approaches mainly rely on knowledge drawn from sense repository and/or raw or tagged corpora.

2.3.1 Dictionary-Based Approaches

Dictionary based approaches are heuristics that use dictionary definitions and/or different resources for WSD. Common samples of other resources are the following:

- Samples
- Semantic networks
- Thesaurus classification systems
- Raw corpus
- Tagged corpus
- Web search engine counts

The simplest approaches only use dictionary definitions making them essentially fast. Take for example the first sense heuristic. This heuristic works by selecting the first sense in the dictionary. It has a performance of around 60% for all words, but in some domains and dictionaries achieves a performance of around 80%. What can be simpler and fast than selecting the first sense in the dictionary? Nothing!

All of these approaches use data from the source dictionary. Many of these approaches can be tweaked for using other resources improving their performance. They are often seen as baseline methods or cheap solutions in practical applications.

2.4 Influence of the back-off strategy

A back-off strategy provides an answer in cases when the algorithm cannot make a decision. In practice, WSD systems are complemented by a back-off strategy. Usually, simple heuristics are used as back-off strategies like the following:

- Most Frequent Sense (**MFS**). Selects the most frequent sense in a corpus

- First Sense. Chooses the first sense in the list of senses of the sense repository
- Random Sense. Selects the answer randomly

Note that in case of some algorithms like the Simplified Lesk Algorithm using a back-off strategy it's important because they have low recall, as shown in Table 2. In this table (and further in this thesis), P stands for precision, R for recall, F1 for F1-measure.All these values are always presented as percentages.

Table 2 Performance of the Simplified Lesk Algorithm with and without a back-off strategy. Tests were made with a window size of 4.

Back-off strategy	Senseval-2			Senseval-3		
	P	**R**	**F1**	**P**	**R**	**F1**
None	50.4	7.5	13.1	39.1	13.2	19.7
Most frequent sense	62.8	62.8	62.8	57.2	57.2	57.2

Table 2 shows that the Simplified Lesk Algorithm has rather low precision and very low recall working by itself. Low recall values give us a hint for encouraging this research: there are few overlaps between the words near the target word and the target word's dictionary definitions (WordNet glosses and samples in our case).

The usage of a back-off algorithm is important for practical applications, but it does not allow observing the real behavior of a WSD algorithm. For this reason, we perform the comparison of algorithms with and without back-off strategy, because otherwise it remains unclear when the decision is made by the algorithm itself and when by the back-off algorithm.

Chapter 3. Framework

There are many methods for word sense disambiguation. However, dictionary-based methods are easier to understand and have a much simpler implementation. They can be tweaked for using other resources such as corpus, and, as you will find in further sections, they can obtain a performance that rivals the performance of machine learning approaches.

In this chapter you will find

- A description of a WSD system.
- A description of some selected WSD algorithms

3.1 Components of a Bag-of-Words WSD System

The bag-of-words (**BoW**) model is frequently used in natural language processing. It defines a text as an unordered set of words. For the BoW model, grammar and word order are irrelevant. For example, the text "I am playing the bass" could form the following bag of words: $t = \{bass, I, am, playing, the\}$. This thesis is focused just in BoW model approaches.

BoW model approaches usually involve two processes: **lemmatization** and **part-of-speech tagging**. Lemmatization is "the process of grouping together the different inflected forms of a word so they can be analyzed as a single item."[4] Therefore, a lemmatizer is an algorithmic tool that returns the lemma (dictionary form) of a target word. Part-of-speech tagging is the process of adding the part of speech label to words. A part of speech or word class is a linguistic category of words commonly defined by its syntactic behavior. There exist two types of word classes: open and closed. Open word classes acquire new members frequently with the past of the time. There are four open classes in English language: nouns, verbs, adjectives, and adverbs. Closed word classes do not acquire new members.

Most of the BoW model systems are based on the process depicted in Figure 2. The first stage consists in pre-processing the target text. It consists in transforming a raw text like "I am playing the bass with my friends" into the set: $t = \{I_{PRP}, be_{VBP}, play_{VBG}, the_{DT}, bass_{NN}, with_{IN}, my_{PRP\$}, friend_{NNS}\}$. This set contains **lemmatized** words with part-of-speech tags (**POS tags)**.

Loading the dictionary/corpus data is the second stage. This stage consists in retrieving all the corresponding definitions, samples, and semantic data for each target word. Some researchers suggest reading samples from sources like WordNet and SemCor corpus[5]. Samples and definitions are lemmatized and POS tagged too.

Finally, the last stage consists on assigning senses to all open class words in the document. Usually, BoW model approaches use context data and knowledge data for weighting which sense should be selected.

---

[4]Collins English Dictionary, entry for "lemmatize"

[5]http://www.cse.unt.edu/~rada/downloads.html#semcor

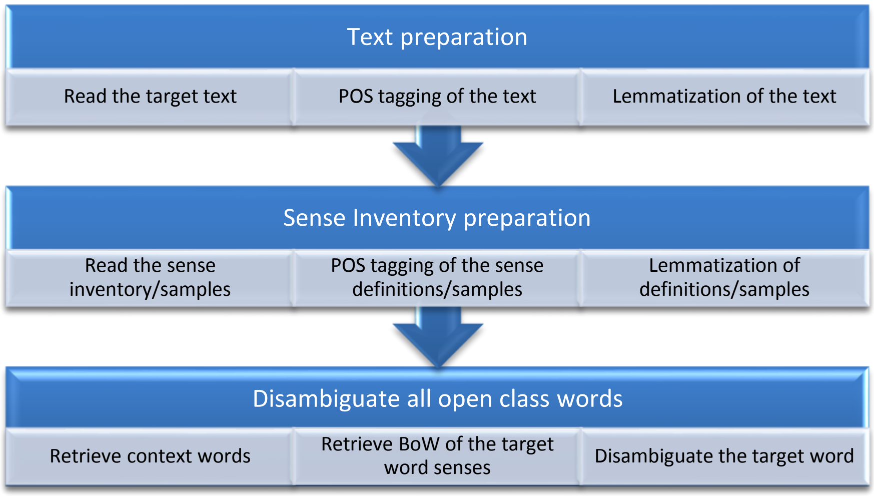


Figure 2 BoW model WSD process

3.2 Selected WSD Algorithms

We have selected some WSD algorithms for testing. They were selected because it's easy implementation and flexibility. The selected algorithms were the following:

- Lesk algorithm
- Simplified Lesk algorithm
- Graph In Degree
- First Sense

We also performed tests with some other algorithms (including conceptual density, lexical chains and machine learning) when available. In the following subsections you will found a detailed description of each one of the selected algorithms.

3.2.1 Lesk Algorithm

The original Lesk algorithm (Lesk 1986) is a dictionary-based approach that disambiguates by calculating the overlaps between all the possible senses of every word in a sentence. It chooses a set of senses having the greatest mutual overlap (one per word).Lesk algorithm sees WSD as a complex combinatorial optimization problem. A major problem of this algorithm is the amount of resources and time needed. Its complexity is exponential by the number of words per sentence (Gelbukh et al. 2005). There are many improvements of the original Lesk algorithm ranging from simply using different optimization heuristics to involving additional resources (Vasilescu et al. 2004, Gelbulkh et al. 2005, Banerjee and Pedersen 2002), but the problem of its prohibitively high complexity remains unsolved.

Let us disambiguate “pine cone” with the following dictionary definitions (Lesk 1986):

Pine:

1) Kinds of evergreen tree with needle-shaped leaves

2) Waste away through sorrow or illness

Cone:

1) Solid body which narrows to a point

2) Something of this shape whether solid or hollow

3) Fruit of certain evergreen trees

The resulting intersections of open class words are:

Pine#1 ∩ Cone#1 = 0

Pine#1 ∩ Cone#2 = 0

Pine#1 ∩ Cone#3 = 2

Pine#2 ∩ Cone#1 = 0

Pine#2 ∩ Cone#2 = 0

Pine#2 ∩ Cone#3 = 0

Lesk algorithm will select Pine#1 and Cone#3 as its answers.

3.2.2 Simplified Lesk Algorithm

The Simplified Lesk Algorithm (Kilgarriff and Rosenzweig 2000) has lineal complexity, while retaining performance comparable with the original Lesk algorithm. It is widely used for research and practical purposes because of its high speed, simplicity, and relatively acceptable performance (Vasilescu et al. 2004, Mihalcea 2006). This algorithm disambiguates each word in the document independently. Given a word, the algorithm chooses the sense having the greatest overlap between its dictionary definition and its context (Mihalcea 2006); see Algorithm 1.

Algorithm 1 Simplified Lesk Algorithm

For each word *W* of the *document*
  Fill the window Win with *N* words around W
  For each sense s_i of W
    Compute $Overlap(s_i, Win)$
  Select the sense $argmax_i Overlap(s_i, Win)$ for W and use the most frequent sense criterion in case of tie

3.2.3 Graph-Based Approaches

Graph based approaches work by modeling word sense dependencies in text as graphs and using graph centrality algorithms for disambiguation. The algorithm can be explained as following: given a sequence of words $\mathrm{W} = \{\mathrm{w}_1, \mathrm{w}_2, \ldots, \mathrm{w}_\mathrm{n}\}$,each word $\mathrm{w}_\mathrm{i}$will have a corresponding admissible labels (senses) $\mathrm{L}_{\mathrm{w}_\mathrm{i}} = \{\mathrm{l}^1_{\mathrm{w}_\mathrm{i}}, \mathrm{l}^2_{\mathrm{w}_\mathrm{i}}, \ldots, \mathrm{l}^{\mathrm{N}_{\mathrm{w}_\mathrm{i}}}_{\mathrm{w}_\mathrm{i}}\}$.The label graph G = (V,E) will have a vertex (having a centrality score) $v \in V$ for every possible label and an edge (having a similarity score) for connecting them to vertices of other words. Hence, the graph will depict relations and degree of relationship that each sense has. The sense (vertex) having the greatest centrality score will be selected as the answer. Figure 3 shows an example of a graphical structure for a sequence of four words. Note that the graph does not have to be fully connected, as not all label pairs can be related by a dependency.

For instance, for the graph drawn in Figure 3, the word w_1 will be assigned with label $l^1_{w_1}$, since the score associated with this label (1.39) is the maximum among the scores assigned to all admissible labels associated with this word.

Graph based algorithms take into account information drawn from the entire graph. They depict relationships among all the words in a sequence. This makes them superior to other approaches that rely only on local information individually derived for each word.

Semantic similarity measures are used for weighting the edges. They quantify the degree to which two words are semantically related using information drawn from semantic networks –see e.g. (Budanitsky and Hirst 2001) for an overview. There are six measures found to work well on the WordNet hierarchy: Leacock & Chodorow, Lesk, Wu & Palmer, Resnik, Lin, and Jiang & Conrath (Leacock and Chodorow 1998; Wu and Palmer 1994; Resnik 1995; Lin 1998; Jiang and Conrath 1997). All these measures assume as input a pair of concepts, and return a value indicating their semantic relatedness. These measures have good performance in other language processing applications and a relatively high computational efficiency.

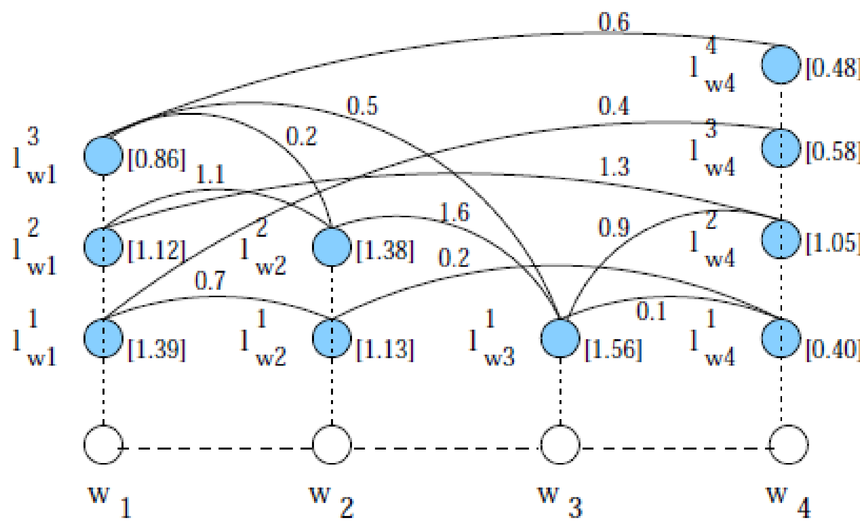


Figure 3 Sample graph built on the set of possible labels (shaded nodes) for a sequence of four words (white nodes).Label dependencies are indicated as edge weights. Scores computed by the graph-based algorithm are shown in brackets, next to each label.

Centrality measures give us a hint of the importance of a vertex in a graph. So, they will tell the algorithm how influential a sense is. There are four centrality measures: indegree, closeness, betweenness and PageRank.

In (Sinha and Mihalcea 2007) there are tests with different semantic similarity measures for weighting edges and with several centrality algorithms for scoring vertices. We decided to use the best performing measures: indegree centrality algorithm and Lesk (intersection) as a similarity measure. The Lesk similarity of two concepts is defined as a function of the overlap between the corresponding definitions. The application of the Lesk similarity measure is not limited to semantic networks and it can be used in conjunction with any dictionary that provides word definitions. The Indegree measure is defined as follows:

$$Indegree(V_a) = \sum_{(V_a,V_b)\in E} W_{a,b} \qquad (5)$$

3.2.4 Lexical Chains

A lexical chain is a sequence of related words in writing, spanning short (adjacent words or sentences) or long distances (entire text). A chain is independent of the grammatical structure of the text: it is a list of words. Chains try to capture a portion of the cohesive structure of the text. A lexical chain can provide a context for the resolution of an ambiguous term and enable identification of the concept that the term represents. In later sections, we used Jaccard score between the glosses of words in the text as proposed in (Vasilescu et al. 2004).

$$J(sense_1, sense_2) = \frac{sense_1 \cap sense_2}{sense_1 \cup sense_2} \qquad (6)$$

Examples of lexical chains are the following:

Rome → capital → city → inhabitant

Wikipedia → resource → web

3.2.5 Conceptual Density

Conceptual density is based on the conceptual distance concept. Conceptual distance tries to provide a basis for measuring closeness in meaning among words, taking as reference a structured hierarchical net, such as WordNet. Conceptual distance between two concepts is defined in (Rada et al. 1989) as the length of the shortest path that connects the concepts in a hierarchical semantic network.

Conceptual density uses the following:

- The length of the shortest path that connects the concepts involved: shorter paths mean that the concepts are closely related.

- The depth in the hierarchy: concepts in a deeper part of the hierarchy should be ranked closer.
- The density of concepts in the hierarchy: concepts in a dense part of the hierarchy are relatively closer than those in a sparser region.

Given a concept c, at the top of a sub hierarchy, and given a mean number of hyponyms per node (*nhyp*), the Conceptual Density for c when its sub hierarchy contains a number m (marks) of senses of the words to disambiguate is given by the formula below:

$$CD(c,m) = \frac{\sum_{i=0}^{m-1} nhyp^{i^{0.20}}}{descendants_c} \qquad (6)$$

Conceptual density disambiguation consists in looking for the maximum sense tree extracted from the senses of nouns of the target text.

Chapter 4. Our Window Selection Procedure

This section describes the modifications for the proposed window selection that improves the Simplified Lesk Algorithm's performance. Each subsection describes one proposed modification. We introduce each modification separately for a more clear description.

4.1 State of the Art in Common Window Selection Procedures

Window selection is the process of selecting words from the text containing the target word. These words are used for weighting the possible senses along with the knowledge data extracted from the dictionary and other resources. The most common practice is to select all the words in the sentence containing the target word. However, you will find out that this is not the best practice for selecting a context window. This section contains an analysis of the effects of changing the window size (number of words in the window), using duplicates, and including the target word.

4.1.1 Effects of the Window Size

It is usually assumed that the adequate window size is the sentence. However, what is the reason for this? Smaller window sizes usually lead to higher precision, while bigger window sizes lead to higher coverage at the cost of some precision. In addition, a higher precision/low coverage system is desirable when using back-off chains as frequently used in real life.

Now, let us analyze the effects of the window size on the Simplified Lesk algorithm for illustrating this behavior. Using a wider window allows the algorithm to try disambiguating more words, therefore its recall increases as shown in Table 3.

Table 3 Performance obtained by the Simplified Lesk Algorithm with different window sizes (N) and no back-off strategy. A wider window increases F1 measure by increasing recall.

Window size (N)	Senseval-2			Senseval-3		
	P	**R**	**F1**	**P**	**R**	**F1**
4	**50.4**	7.5	13.1	**39.1**	13.2	19.7
16	45.8	18.9	26.7	36.3	24.6	29.3
64	45.0	32.3	37.6	33.8	28.8	31.1
256	44.6	40.4	42.4	33.5	30.7	32.0
Whole document	43.6	**41.7**	**42.6**	32.4	**30.9**	**31.7**

We performed some testing for observing the changes linked to increasing the window size. Table 4 shows some sample overlapping words between the window and the correct sense when using different window sizes. We observed that using the whole document as a window increases the overlaps with the correct sense. Please observe that the new words producing overlaps can be considered domain words (E.G. sunday_N, rule_N, worship_N, follow_V ,service_N).

Table 4 Sample overlapping words between the window and the correct sense extracted from Senseval-2 test set with four words and the whole document as a window.

Target word	4 words window	Whole document window
Art_1	( )	(art_N, work_N)
Bell_1	(sound_N)	(sound_N, ringing_N, make_V)
Service_3	( )	(sunday_N, rule_N, worship_N, follow_V ,service_N)
Teach_1	( )	(knowledge_N, french_J)
Child_2	( )	(human_J, child_N, college_N, kid_N)

We can conclude that words needed for WSD exists in the document, but are not visible when using a small window. A greater window leads to better recall, though precision is decreased slightly. So, **how can we retain the precision while increasing the coverage?**

Now, let us take a look at Figure 4. We can observe that overlap counts become bigger with greater window sizes while the number of words does not grow that much. Increasing the window size increases the influence of common words. Excluding common words such as *be*, *do* or *not* is not the proper solution. These words often mislead WSD algorithms into choosing wrong senses, but they are still are necessary for disambiguating some words.

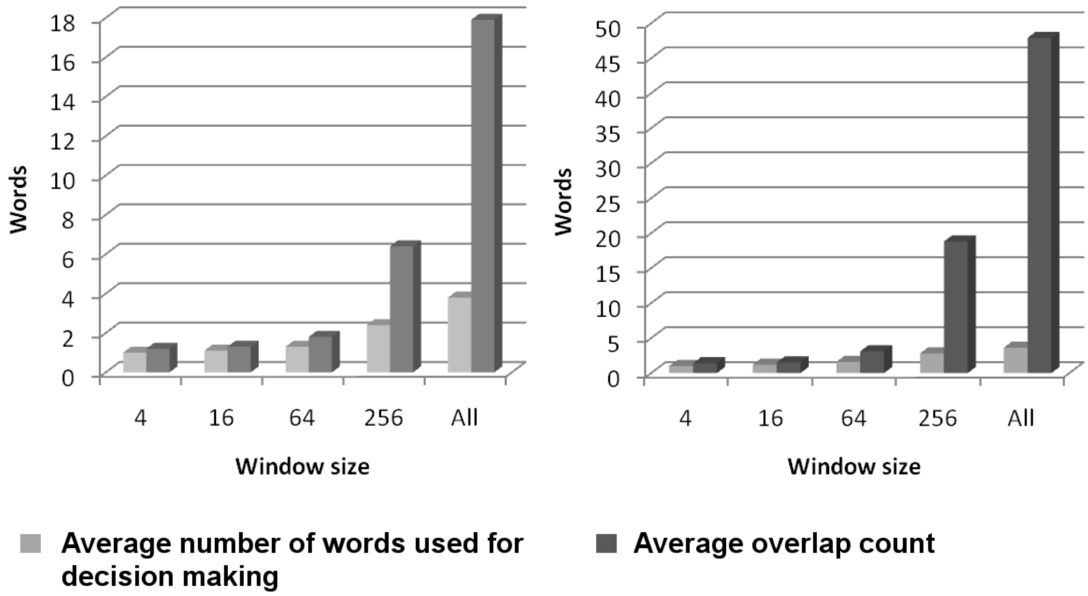


Figure 4 Overlap count observed with different window sizes in Senseval-2 (left) and Senseval-3 (right) test sets.

Evidence in Figure 4 does not confirm that words producing such big overlap counts are common words (although it sounds like the most logical explanation). We need to measure of the commonness of a word. We selected the dictionary-based version of IDF measure (**IDF_D**) (Kilgarriff and Rosenzweig 2000). IDF_D is calculated using the following equation:

$$IDF_D(w) = -log\left(\frac{|g: w \in g|}{G}\right) \qquad (7)$$

where *G* is the total number of glosses in the dictionary and $|g: w \in g|$ is the number of glosses where the lemma *w* appears. Words that appear too often in the dictionary such as *be*, *have* or *not* have low IDF_D values. For example, we observed that these three words have an $IDF_D<3.5$ while the average value is $IDF_D=10.7$.

Table 5 shows us the words producing more overlaps. We can confirm that the Simplified Lesk Algorithm often uses words like *be*, *have* and *not* for making decisions. Also, you can observe that the word *be* have a huge influence in its disambiguation process.

Table 5 Top 10 words used for making decisions. Tests were made with the Simplified Lesk Algorithm using 4 words and whole document windows.

4 words window					
Word	**Senseval-2**		**Word**	**Senseval-3**	
	IDF_D	**Decisions made**		**IDF_D**	**Decisions made**
not_R	3.2	35	be_V	1.5	379
$other_J$	4.4	12	$have_V$	2.4	25
$gene_N$	7.0	10	man_N	5.0	13
$bell_N$	7.3	10	do_V	4.3	12
$cell_N$	5.5	7	$time_N$	4.5	11
$child_N$	4.8	7	$take_V$	4.7	11
$study_N$	5.6	7	$make_V$	3.3	10
new_J	5.0	6	get_V	5.4	8
$make_V$	3.3	6	$policy_N$	6.5	6
$year_N$	5.2	6	$house_N$	5.6	6
Whole document window					
Word	**Senseval-2**		**Word**	**Senseval-3**	
	IDF_D	**Decisions made**		**IDF_D**	**Decisions made**
not_R	3.2	546	be_V	1.5	1481
be_V	1.5	298	$have_V$	2.4	322
$make_V$	3.3	206	$make_V$	3.3	223
new_J	5.0	164	do_V	4.3	169
use_V	3.0	147	$house_N$	4.5	123
$other_J$	4.4	126	$people_N$	4.7	121
$child_N$	4.8	115	man_N	5.0	100
$take_V$	3.3	110	$time_N$	4.5	85
$person_N$	3.8	101	$state_N$	4.4	74
$year_N$	5.2	100	$money_N$	5.2	73

Finally, let us confirm that smaller window sizes lead to a better integration with the first sense back-off strategy. Bigger window sizes decrease performance when using a back-off strategy as shown in Table 6. Performance of the first sense heuristic is better than the performance of Simplified Lesk algorithm. Therefore, a major participation of the Simplified Lesk Algorithm will provoke a minor participation of the back-off strategy, hence, a lower performance. Please remember that the first sense heuristic is already in the top of its performance and it returns an answer (good or bad) for all words, so, it is best to use it as back-off strategy or standalone algorithm.

Table 6 F1 measure obtained by using Simplified Lesk Algorithm with different window sizes and most frequent sense back-off. A wider window decreases integration with the most frequent sense back-off strategy.

Window size	Senseval-2	Senseval-3
4	62.5	56.9
16	59.5	48.9
64	52.9	40.6
256	47.7	35.5
Whole document	43.4	33.9

4.1.2 Effects of Duplicates in the Context Window

In the previous subsection, it was stated that some words produce more than one overlap. This means that some words appear several times in the text, i.e. their term frequency (**TF**) is greater than 1. However, what will happen if we reduce this effect by removing duplicates in the context window? Table 7 shows the performance of the Simplified Lesk Algorithm, with and without taking into account TF of words. **Removing duplicates from the window improves precision.**

4.1.3 Effects of Including the Target Word in the Context Window

The Simplified Lesk Algorithm sometimes includes the target word in the window. This will happen when using big window sizes like the whole document. The target word will surely influence WSD because: (1) definitions often contain the word that they are describing, and, (2) documents often include some repeated words.

Table 7 Performance of the Simplified Lesk Algorithm with and without duplicates in the window. Tests were made by using the whole document as window.

Duplicates	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
Yes	43.6	41.7	42.6	32.4	30.9	31.7
No	46.5	42.6	44.5	36.4	33.7	35.0

For example, Table 8 contains the first five definitions of the word $bell_N$. We can observe that senses 3, 4 and 5 include the word $bell_N$. $bell_N$ appears 22 times in the first document of Senseval-2 test set, hence, it will add a 22 overlap count to senses containing it when using the whole document as window. It will add an overlap count of one to senses including it even after removing duplicates from the window. The inclusion of the target word negatively affects performance as seen in Table 9. Therefore, **context window should not include the target word.**

Table 8 First five definitions of the word $bell_N$.

Sense	Definition
$Bell_1$	A hollow device made of metal that makes a ringing sound when struck
$Bell_2$	A push button at an outer door that gives a ringing or buzzing signal when pushed
$Bell_3$	The sound of a bell being struck
$Bell_4$	(nautical) each of the eight half-hour units of nautical time signaled by strokes of a ship's bell; eight bells signals 4:00, 8:00, or 12:00 o'clock, ...
$Bell_5$	The shape of a bell

Table 9 Performance of the Simplified Lesk Algorithm with and without the target word in the window. Test were made by using the whole document as window.

Target word	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
Yes	43.6	41.7	42.6	32.4	30.9	31.7
No	48.5	46.1	47.3	34.9	33.1	34.0

4.2 Our Window Selection Procedure

We propose selecting a context window having useful words while avoiding the target word and repetitions. In the previous subsection, we stated that removing duplicates and avoiding the target word leads to a performance boost. We also want to add a third filter and combine all three modifications together. We believe that the whole document is not needed for WSD. We confirmed in the next sections that algorithms only need few words from different places of the documents.

For example, the Simplified Lesk Algorithm does not really use all words from the document, as it was shown in Figure 4. In fact, it used an average of four words when using the whole document as the context window. Hence, we propose using only these “useful” words as the context window instead of using all words, i.e., instead of the whole document. In this manner, we filter out words not having overlaps with any sense of the target word. Algorithm 2 shows the proposed method for extracting these useful words. Words will be selected from the closest possible context of the target word, but they could be extracted from any place of the document. Our context window will contain fewer words sometimes – this will happen when having small definitions or small documents.

Algorithm 2 Window construction algorithm that selects only N words that have overlaps.

Set *i=1*
    Look for a word W_p at *i* positions to the **right** of the target word *W*
    If W_p exists in any sense of *W*
        Add W_p to *Win*
    Look for a word W_p at *i* positions to the **left** of the target word *W*
    If(W_p exists in some sense of *W* and $sizeOf(Win)<N$)
        Add W_p to *Win*
    Set *i=i+1*

4.3 Performance Analysis of Our Window Selection Procedure

Let us look the effect of only using useful words as context window. In Table 10, we made a comparison between three different context windows: the closest four words, the whole document and the closest 4 useful words. Using the first four overlapping words as the window gives better results than the other two window selection strategies.

Table 10 Performance of the Simplified Lesk Algorithm using three strategies of the context window selection.

Window selection	Senseval-2			Senseval-3		
	P	**R**	**F1**	**P**	**R**	**F1**
4 words	50.4	7.5	13.1	39.1	13.2	19.7
Whole document	43.6	41.7	42.6	32.4	30.9	31.7
4 overlapping	48.0	45.9	46.9	39.1	37.4	38.2

The performance is further improved if we filter out duplicates and the target word as shown in Table 11. We detected the following behaviors:

- The proposed window selection procedure allows the algorithm to discriminate more wrong senses as shown in Figure 5.

- The proposed window selection procedure allows the algorithm to address the proper sense with a precision competitive to state-of-the-art systems. However, wrong senses had better scores that the correct sense many times.

This means that the algorithm can be used for telling for discarding half of the senses (wrong senses) with a good precision.

Table 11 Comparisons of the proposed modifications and their combination.

Window selection	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
4 words (baseline)	**50.4**	7.5	13.1	39.1	13.2	19.7
Whole document	43.6	41.7	42.6	32.4	30.9	31.7
Removing repetitions	46.5	42.6	44.5	36.4	33.7	35.0
Excluding the target word	48.5	46.1	47.3	34.9	33.1	34.0
4 overlapping words	48.0	45.9	46.9	39.1	37.4	38.2
All proposed modifications	50.2	**47.9**	**49.0**	**39.4**	**37.5**	**38.4**

4.3.1 Integration with other Dictionary-Based Methods

First, let us confirm that our window selection makes the Simplified Lesk Algorithm competitive against other dictionary-based methods that are better than the Simplified Lesk Algorithm. The selected dictionary-based methods were:

- Conceptual density (Agirre and Rigau 1996).
- Graph indegree (Sinha and Mihalcea 2007).
- The Simplified Lesk Algorithm with a lexical chain window (Vasilescu et al. 2004). This modified version of the Simplified Lesk Algorithm considers only words that

form a lexical chain with the target word in the window. It outperforms the original version of the Simplified Lesk Algorithm.

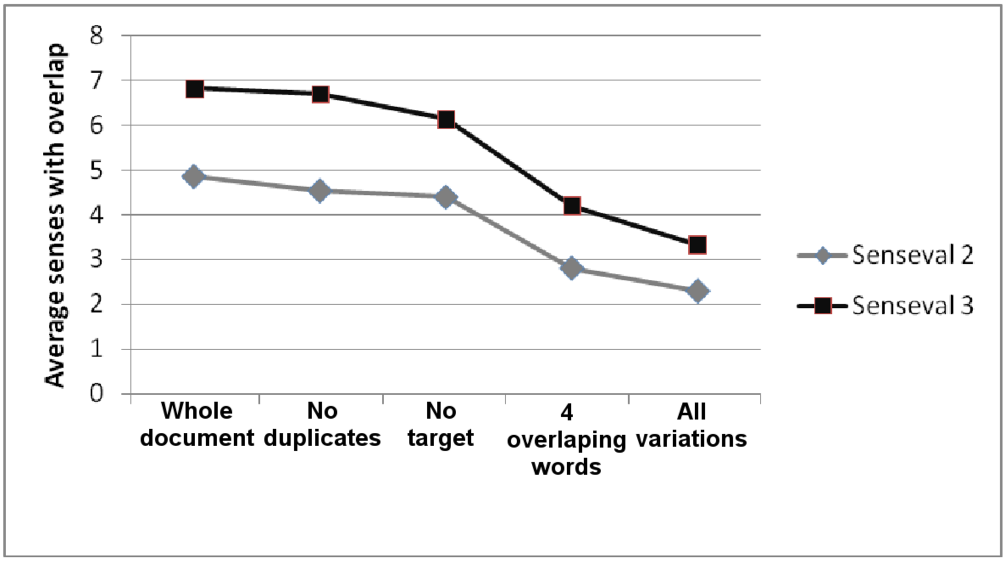


Figure 5 Average number of senses with an *overlap>0* for each attempted word in Senseval-2 and Senseval-3 test sets

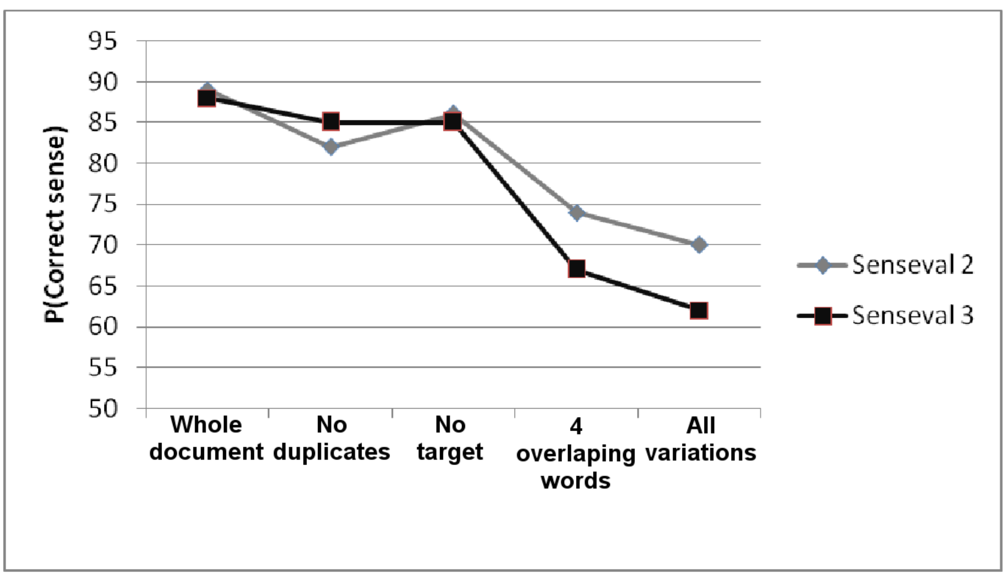


Figure 6 Probability of having the correct sense among the answers.

The improved Simplified Lesk Algorithm outperforms other dictionary-based methods like conceptual density and graph indegree as shown in Table 12. It also outperforms the lexical chain window.

Table 12 Comparison of the improved Simplified Lesk Algorithm with other dictionary-based algorithms.

WSD method	Senseval-2			Senseval-3		
	P	**R**	**F1**	**P**	**R**	**F1**
Simplified Lesk Algorithm (baseline)	**50.4**	7.5	13.1	39.1	13.2	19.7
Conceptual density	25.1	4.2	7.2	25.6	5.8	9.5
SLA with Lexical chain window	48.6	25.6	33.4	52.6	27.8	36.4
Graph indegree	45.4	37.2	40.1	35.1	30.4	32.6
Improved Simplified Lesk Algorithm	50.2	**47.9**	**49.0**	**39.4**	**37.5**	**38.4**

Now, let us check if the proposed modifications can be applied in the selected dictionary-based methods. Figure 7 shows F1-measure of the afore-mentioned dictionary-based methods alone and combined with the proposed window selection strategies. It can be seen that the proposed strategies work well with the conceptual density and the graph indegree approaches. However, they cannot be used for the lexical chain window algorithm.

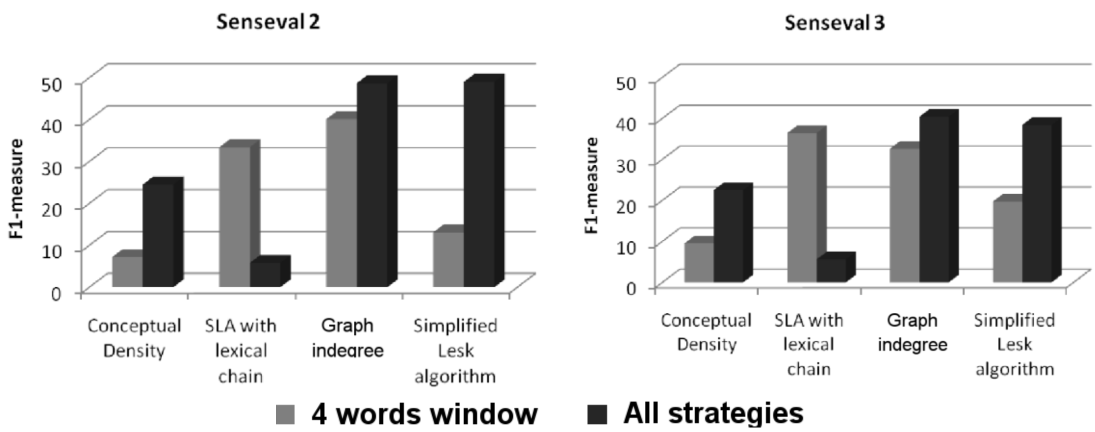


Figure 7 F1-measure of dictionary-based methods alone and combined with the window selection procedures.

Chapter 5. Using One Sense per Discourse for Disambiguating Domain Words

We discovered that words known to have one sense per discourse (OSD words) can be disambiguated easily. Coincidently, OSD words have its sense being defined by the document domain rather than the sentence. You will find in this chapter that most of the current methods have a precision of around 75% in the domain words and a low precision in local words (a maximum of 50%).

5.1 State of the Art in the Use of One Sense per Discourse Heuristic

The one sense per discourse condition (**OSD**) tells us that all instances of a word will have a single meaning through the whole document (Gale et al. 1991). This rule has a probability of above 90% in homographs and a maximum of 70% in other words (Martínez and Agirre 2000). For example, the word wolf has more than 120 senses in Wikipedia, see http://en.wikipedia.org/wiki/Wolf_(disambiguation). However, these senses can be clustered into the following nine categories: animals (17 senses), people (1 sense), sports teams (43 senses), places (14 senses), vehicles (9 senses), music (24 senses), radio and television stations (11 senses), titles (13 senses) and others (4 senses). Supposing we are disambiguating the following sentence:

"Now the wolves have taken a three point lead."

It is fairly easy to discern that *wolves* is referring to a sports team, but it is really hard to tell which one –even for most of us who doesn't know about a particular sport.

OSD assumption has been used for WSD. WSD systems will do the following when forcing the OSD assumption:

- Context window will be filled with words extracted from all the sentences containing the target word instead of just using the current sentence.
- The selected sense will be assigned to all instances of the target word.

Forcing OSD assumption often increases recall of WSD systems. The OSD assumption is implicitly used when using the whole text as context window.

In addition, OSD was used for disambiguating some selected nouns in with some success (Yarowsky 1995). It is relevant to note that the words were disambiguated by using domain information, so, that give us a hint of what to do.

5.2 Our Intended Use of One Sense per Discourse Heuristic

We give OSD rule a different role than unifying answers and using extended context windows. We have discovered that the selected dictionary-based methods have trouble solving words known for not having OSD. We propose that methods should avoid disambiguating these words. We used the SemCor corpus (Miller et al. 1994) for calculating OSD. A word is considered to have OSD when:

- It appears in the corpus with a maximum of one sense assigned per document.
- It does not exist in the corpus.

Words of all classes will be filtered out (as seen in Table 13) on the selected test sets. The amount of words filtered out range from 14% to 58%. Most of the OSD words can be considered domain words, E.G. *scientist, cell, cancer, strategy* and *treatment*.

Table 13 Average words discarded of each class.

	Noun	Verb	Adjective	Adverb
Senseval 2	39%	74%	43%	50%
Senseval 3	47%	81%	38%	0%

On the other hand, words not having OSD have some of these traits:

- Their senses are described with definitions that are too similar between them – some of these definitions are too close that even people can discern between them.
- Their senses are described with definitions that are too short. Such definitions include less than three open-class words.
- Their meaning is linked to their current syntactic relations rather than the document domain. Verbs meaning is often defined by its complements rather than document domain.

See Table 14 for some sample definitions that are too similar or too short for WSD systems.

Table 14 Definitions that are too similar or too short for WSD systems.

Sense	Definition
$World_2$	People in general; especially a distinctive group of people with some shared interest
$World_5$	People in general considered as a whole
$Medical_1$	Relating to the study or practice of medicine
$Medical_2$	Requiring or amenable to treatment by medicine as opposed to surgery
$Here_1$	In or at this place; where the speaker or writer is
$Here_3$	To this place (especially toward the speaker)
$Bell_5$	The shape of a bell
$Time_4$	a suitable moment
$Recent_1$	New

Verbs were the words discarded more often. Common verbs (like *be, have* and *do*) have more than ten definitions in WordNet and are used widely across all domains. Please remember that verbs' meaning is more likely to be defined by its complements. For example, in the following text:

"*I started drinking some soda. Later, I decided to drink a cold beer*."

Now let us disambiguate using the following definitions extracted from WordNet: [$drink^1$:take in liquids] and [$drink^2$:consume alcohol]. In this example, both definitions are clear for people but they are rather short for WSD algorithms. You can easily select the sense of the verb drink by looking at the direct object in both cases. Most of dictionary based methods do not disambiguate both instances of the verb correctly. The verb drink does not have OSD, so it is recommended that dictionary-based methods do not disambiguate this word.

Avoiding such "difficult" words will allow systems to have high precision with low coverage, closing in to a 100% precision solution. Such solution should be used first for solving easy problems and identifying hard problems. We believe that by putting effort in such solution will allow us to be one step closer to a 100% accuracy WSD system.

5.3 Performance Analysis

Test results displayed on Tables 15, 16 and 17 confirm that disambiguating just the words with OSD increases precision at the cost of coverage. Figure 8 provides you a graphical alternative for you to observe these performance changes. We can conclude the following from these tables and figures:

- The precision boost ranged from 3% to 25% (an average of 16%).
- The coverage loss ranged from 11% to 57% (an average of 34%).
- The improved first sense heuristic was the best approach in the tests: it obtained a precision of at least 79%.

Additionally, we observed that forcing the OSD assumption does not lead to a consistent increase in precision (although, it often leads to a coverage boost). We have added Table 18 for further reference. Table 18 contains the best results observed in Senseval 2 and Senseval 3 (see Table 18). Please note that our improved first sense heuristic overcome the precision of the best systems in these competitions.

Table 15 Test results corresponding to Conceptual Density and Naive Bayes algorithms observed in Senseval 2 and Senseval 3 competitions.

WSD method	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
OSD Conceptual Density	57.1	5.8	10.5	64.7	13.4	22.2
Conceptual density	25.1	4.2	7.2	25.6	5.8	9.5
OSD Naïve Bayes	73.7	36.0	48.3	74.5	30.6	43.4
Naïve Bayes	58.4	57.0	57.7	54.9	54.2	54.6

Table 16 Test results corresponding to GETALP system at Semeval 2013 competition.

WSD method	Semeval 2013		
	P	R	F1
OSD GETALP	65.7	37.9	48.1
GETALP (Schwab et al 2013)	51.6	51.6	51.6

Table 17 Performance comparison of some bag of words algorithms. All methods exhibit a precision boost and coverage lost when solving just OSD words. FG means forcing one sense per discourse and OSD means solving OSD words exclusively.

WSD method	Senseval-2			Senseval-3		
	P	**R**	**F1**	**P**	**R**	**F1**
OSD Simplified Lesk Algorithm	61.0	12.1	20.2	52.7	10.4	17.4
FG Simplified Lesk Algorithm	45.5	31.0	36.8	30.9	23.3	26.6
Simplified Lesk Algorithm	50.4	7.5	13.1	39.1	13.2	19.7
OSD Graph indegree	78.1	39.9	52.9	70.1	29.5	41.5
FG Graph indegree	57.5	57.4	57.4	51.1	50.9	51.0
Graph indegree	45.4	37.2	40.1	35.1	30.4	32.6
OSD Lesk	67.6	31.9	43.3	64.3	37.8	35.2
FG Lesk	49.4	49.4	49.4	49.4	49.4	49.4
Lesk	48.1	46.0	47.0	38.4	36.7	37.8
OSD First Sense	78.8	40.0	53.1	79.3	33.1	46.5
First Sense	62.8	62.8	62.8	57.2	57.2	57.2

Table 18 Systems having the highest precision in Senseval 2 and Senseval 3 competitions.

WSD method	Senseval-2		
	P	**R**	**F1**
OSD First Sense	78.8	40.0	53.1
IRST (Magnini et al. 2001)	74.8	35.7	48.3
SMUaw (Mihalcea & Moldovan 2001)	69.0	69.0	69.0
CNTS-Antwerp (Hoste et al. 2001)	63.6	63.6	63.6
	Senseval-3		
OSD First Sense	79.3	33.1	46.5
IRST-DDD-09-U (Strapparava et al. 2004)	72.9	44.1	54.9
IRST-DDD-LSA-U	66.1	49.6	56.6
Gambl-AW-S (Decadt et al. 2004)	65.1	65.1	65.1

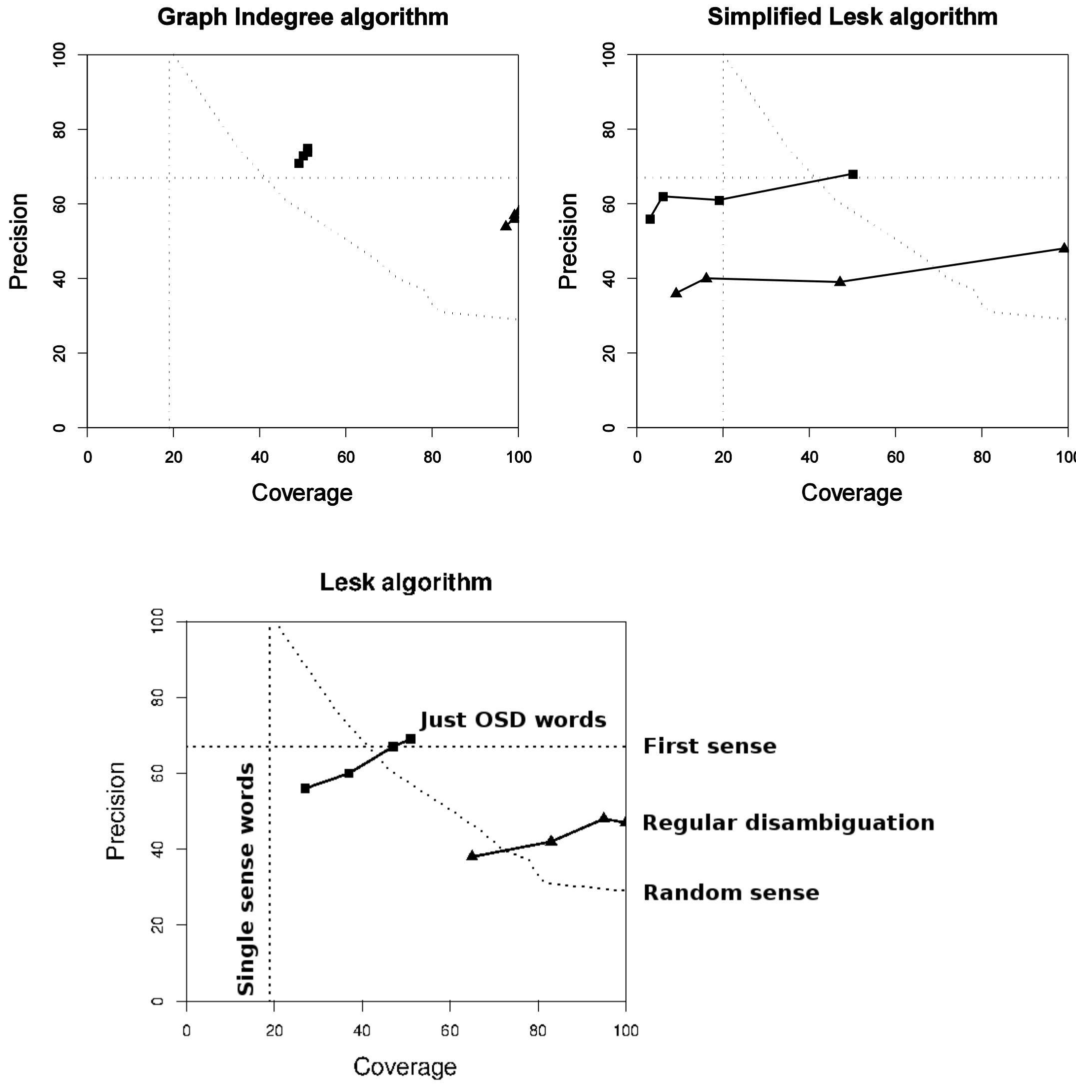


Figure 8 Precision/coverage graph for the Simplified Lesk, Graph Indegree and Lesk algorithms observed on Senseval 2 test set. We used different window sizes (ranging from 1 to the whole text). Algorithms disambiguating just the OSD words (squares) overcome the baselines (dotted lines) and its original performance (triangles).

Chapter 6. Using Co-Occurring Words for Improving WSD

Extending glosses improves the quality of WSD by adding useful words to the bag-of-words. The most common practice is to use related terms existing in the dictionary and specified by the lexicographer. We propose adding co-occurring words for extending glosses. Co-occurring words are the ones that appear together through the corpus in different documents. For example, the words *art*, *popularity*, *folklore* and *cultural* can be seen as co-occurring words. Co-occurring words are automatically extracted from a corpus. We have discovered that this practice gives a consistent performance boost to dictionary-based methods and can be used along with other gloss extending practices.

6.1 State-of-the-Art Practices for Extending Glosses

Extending glosses is a common practice for improving WSD. The most common ones are:

- Using synonyms: consist of adding the synonyms of the current sense.
- Using related terms: consist of adding semantic related terms such as hyperonyms, antonyms, and so on.
- Using glosses of related terms: consist of adding the complete glosses of related terms.
- Using corpus samples: consist of adding sentences where the target sense appears.

Table 19 shows that all these methods improve Simplified Lesk Algorithm's performance. The best method is using corpus samples. Currently, there is no study of gloss extending practices and its interactions between them.

Table 19 Performance comparison of the Simplified Lesk when using different gloss extending methods.

WSD method	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
Regular glosses	43.6	41.7	42.6	32.4	30.9	31.7
Adding synonyms	48.6	48.2	48.4	36.6	36.1	36.4
Adding related terms	49.8	49.4	49.6	48.0	47.1	47.6
Adding related glosses	59.6	59.5	59.6	53.3	53.1	53.2
Corpus samples	66.3	66.0	66.1	67.0	66.7	66.9

6.2 Using Co-occurring Words for Extending Glosses

Co-occurring words are commonly used for aiding other tasks of natural language processing such as information retrieval and keyword extraction. They can be extracted from a corpus using statistical methods. We propose to use co-occurring words in WSD as a mean to extend glosses.

There are many ways of extracting co-occurring words. The standard way of doing is using a statistical measure of word relatedness. These measures identify co-occurring words with regular quality. Please note that such measures need to take into account that there are many common words that appear in most of the documents like *be*, *do* and *have* so. Here they are some common measures:

- Conditional probability
- Point wise mutual information (Church & Hanks 1990)
- Semantic similarity measures (Agirre & Edmonds 2006)

We have selected conditional probability because it has greater precision than the other measures (as shown in Cimiano et al. 2005). Conditional probability is calculated with the following equation:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

We used the same equation for all test sets. We used SemCor corpus as our base corpus. The algorithm looks for co-occurring words for each sense of the target word as following:

1. Look for all the corpus documents containing the target sense.
2. Calculate the conditional probability of all the words in the documents and the target sense.
3. The target will be *A* and the possible word will be *B*.
4. Select the words having a $P(A|B)>0.3$ (this threshold was determined by testing with values in the range of $[0.0, 0.1, \ldots, 1.0]$).
5. Remove duplicated words if any.

Let us look at some example co-occurring words extracted by our algorithm:

- [*rookie_N, pitching_N, monday_N, indianapolis_N, husky_J, left-hander_N*]
- [*wizard_N, violin_N, recital_N, thursday_N, 20th_J, slashing_J, demon-ridden_J, cadenza_N*]
- [*angry_J, turmoil_N, briefing_N, insult_V, hulk_N*]

However, our method still selects some common or unuseful words such as *thursday_N* and *20th_J*. Future work will be aimed to solve this issue.

6.3 Performance Analysis

Using co-occurring words gives the Simplified Lesk Algorithm a performance rivaling the use of corpus samples as shown in Tables 19 and 20. Table 21 shows that co-occurring words can be combined with other methods with some success.

Table 20 Using co-occurring words for extending the gloss is an effective way of increasing performance.

WSD method	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
Simplified Lesk Algorithm with co-occurring words	68.5	61.8	65.0	65.4	61.4	63.4
Simplified Lesk Algorithm with Corpus samples	66.3	66.0	66.1	67.0	66.7	66.9
Simplified Lesk Algorithm	50.4	7.5	13.1	39.1	13.2	19.7

Table 21 Combining all the gloss extending methods with co-occurring words.

WSD method	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
Adding synonyms with co-occurring words	66.5	66.3	66.4	59.9	59.7	59.8
Adding related terms with co-occurring words	65.9	65.7	65.8	60.9	60.6	60.8
Adding related glosses with co-occurring words	67.1	67.1	67.1	61.5	61.5	61.5
Corpus samples with co-occurring words	66.9	66.6	66.8	67.4	67.0	67.2

Table 22 Comparison of our proposed modifications.

WSD method	Senseval-2			Senseval-3		
	P	R	F1	P	R	F1
All of our modifications	**82.4**	40.4	54.2	**81.0**	32.7	46.6
Useful words window	50.2	47.9	49.0	39.4	37.5	38.4
OSD	61.0	12.1	20.2	52.7	10.4	17.4
Co-occurring words	68.5	**61.8**	**65.0**	65.4	**61.4**	**63.4**
Simplified Lesk Algorithm	50.4	7.5	13.1	39.1	13.2	19.7

6.4 Integrating All of Our Modifications

We tested all of our modifications together. Test results can be seen in Table 22. Tests confirm that:

- Using our modifications combined leads the Simplified Lesk Algorithm into good precision level (greater than 80%).
- Our system has almost the same precision than a human when solving domain words (inter annotator agreement for both test sets was 85%).
- Our system has better precision than the best performing systems observed in Senseval 2 and 3 contests.
- The greater recall was obtained when using co-occurring words.
- All of our modifications greatly overcome the original Simplified Lesk Algorithm.

Chapter 7. Integration with the Original Lesk Algorithm

We also wanted to find out if our modifications can be used with the original Lesk algorithm. It is believed (but not confirmed) that the original Lesk algorithm can be better than its simplified counterpart. However, one of the major drawbacks of this algorithm it's the great amount of computational resources it needs. For that reason, we developed a heuristic that can return good results in short time lapses called Simple Adaptive Climbing (**SAC**). First, we tested our heuristic as a global optimizer in well-known benchmark problems. Then, we tested it within the Lesk algorithm. We have noted that our heuristic need very little time for obtaining comparable results to other heuristics. So, we have successfully reduced the Lesk algorithm limitation.

After that, we tested our modifications into the Lesk algorithm with great success. They obtained good precision. We also confirmed that the simplified version is better for solving domain words.

7.1 State of the Art: Hill-Climbing Like Algorithms

7.1.1 Global optimization problem

In the most general case, global optimization is the task of finding the point (x^*) with the smallest (minimization case) or bigger (maximization case) function value ($f(x^*)$). There are a wide number of solutions for the global optimization problem, but it is still considered an open problem. The first state-of-the-art solution consisted in improving a solution until reaching the time limit. These solutions were called hill-climbers. Hill-climbers were quickly left out because they are quite limited for solving complex problems. Currently, evolutionary algorithms have been widely utilized to find optimal solutions. They are called evolutionary because they somehow mimic the natural evolution process. Its main features are: relatively good performance, low resource needs and parallelization capabilities. They are able to find a good answer quite fast. However, they often are incapable of finding the best answer. The most commonly used Evolutionary Algorithms are the following: Differential Evolution, Genetic Algorithm and Particle Swarm Optimization.

Global optimization currently need:

- Optimizers capable of finding the best answer for solving problems without a time limit.
- Optimizer capable of finding a good answer for solving problems in which time is more important that quality.

In this work we improved a hill-climber algorithm with some evolutionary algorithm for creating an optimizer capable of finding a good answer faster than most of the state-of-the-art algorithms.

7.1.2 Brief Review of Hill-Climbing Like Algorithms

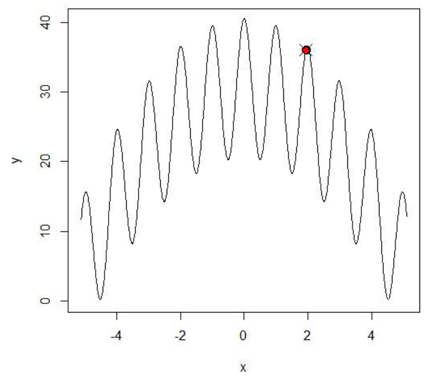
Let us imagine that you are a mountain climber trying to reach a mountain peak while having really thick fog. Imagine that you have forgotten some important gadgets like a compass and a map, but at least you have a lot of food, the perfect climbing suit and equipment, and a machine that tells your current altitude. How will you find the mountain peak? Hill-climbing like algorithms are heuristics for getting you to the highest point surrounding you.

Hill-climbing algorithms are single point optimizers with adjustable search radius. Algorithm 3 depicts a generalization of hill-climbing algorithms. As we can observe in Algorithm 3, these algorithms are greedy strategies that try to move only to a highest next point. The main difference between hill-climbers lies in the specific implementation of the following functions:

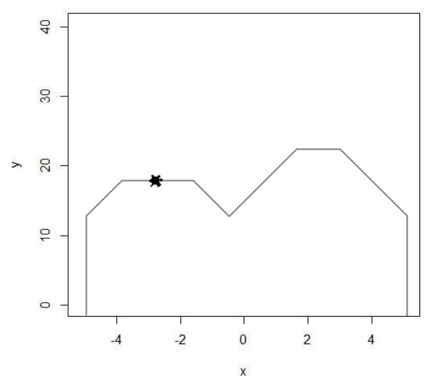
- *mutate(Xbest,steps,parameters)*: Function for searching a new point.
- *adjustStepswhenSuccess(steps,parameters)*: Function for adjusting the search control values –commonly the search radius- when finding a better location than the current one.
- *adjustStepswhenFailure(steps,parameters)*: Function that adjust the search control values when finding a worst location than the current one.

Algorithm 3 Generic algorithm for hill climbers (minimization case)

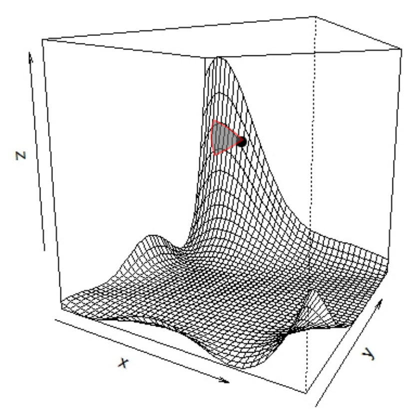
```
Set X as a random initial solution
Initialize the search radius vector (steps) using parameters
While(terminationCriterion())
    Set Xnew=mutate(Xbest,steps,parameters)
    If f(Xnew)<f(Xbest)
        Set steps=adjustStepswhenSuccess(steps,parameters)
        Set Xbest=Xnew
    Else
        Set steps=adjustStepswhenFailure(steps,parameters)
```



a) Foothill problem: the searching process is stuck in a local optimum



b) Plateau problem: the searching process is stuck in a flat surface



c) Ridge problem: the searching area (dark gray) does not allow improving.

Figure 9 Most common problems found in Hill-Climbing algorithms (maximization case).

Hill climbing algorithms are known for being fast on getting to the top of the current hill (local optimum value). However, most of these strategies fail when the mountain have multiple hills (as many real mountains do). The main reasons behind this behavior are the following (Winston 1992):

- **The foothill problem:** the optimizer gets stuck in a local optimum value (frequently, the hill where the climber started).
- **The plateau problem:** the optimizer gets stuck in flat surfaces with some sharp peaks.
- **The ridge problem:** the optimizer gets stuck because the direction of the ascent is not within the set of possible search directions.

You can find a graphical description of these problems in Figure 9.

7.2 Our Algorithm

SAC is a simple single-point optimizer. Its implementation is smaller than most of the evolutionary algorithms, while maintaining a competitive performance. Algorithm 4 shows the algorithm for SAC (minimization case). The general ideas behind the algorithm are the following:

1. Increase the search radius when improving –it promotes a better exploration of the search space.
2. Reduce the search radius when failing –it allows a faster exploration of the current hill.
3. Restart when failing too much -it prevents getting stuck in local optimum values.

Figure 10 shows a graphical example of SAC main behaviors.

SAC requires configuration of two user parameters. These parameters allow adjusting the kind of search behavior. The two user parameters are the following:

- Base step sizes (**B**∈[0.0,0.5]). **B** is the initial and also the maximum possible search radius. It represents a proportion of the whole search space. So, a *B=0.5* value is equivalent to half the search space, in which case the algorithm has the possibility of going to any point in the search space. A value greater than *0.5* will encourage exploration for a longer time and could transform SAC into a random search. Hence, greater values of B encourage more exploration of the search space and slower convergence while lower values encourage a hill-climbing like behavior.

- Maximum consecutive errors ($R>0$). **R** indicates the maximum consecutive errors necessary to restart the searching process. Very small values of *R* could avoid convergence in a precise search situation. Long values of *R* could provoke SAC gets stuck for a long time in a local optimum value. A zero value will turn SAC into a hill-climbing like algorithm.

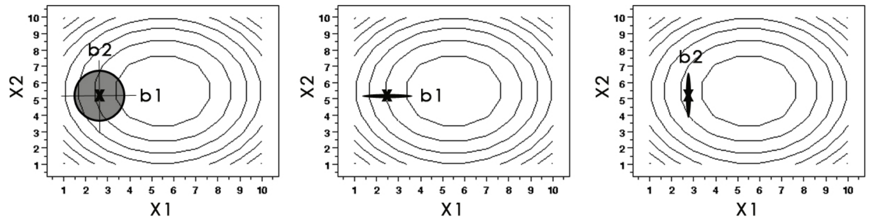
SAC searches by performing explorations in random dimension subsets -this feature allows SAC to overcome the aforementioned *ridge problem*. SAC uses adaptive step sizes for each dimension of the problem (b_j, $j=1,\ldots, D$). These *b* step sizes are the key of SAC exploration process. SAC adjusts them accordingly to the current success/failure state of the search:

- *b* values become greater when improving the current solution, encouraging exploration of the search space (see line 17 on Algorithm 4).
- *b* values became smaller when failing, encouraging exploration of nearby areas (see line 24 on Algorithm 4).

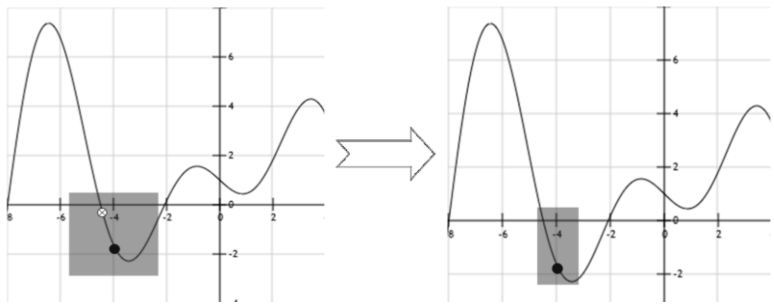
In SAC, the search space is connected at the beginning and the end of all problem dimensions. Hence, if SAC tries to explore outside the upper limit, it will explore a valid region near the lower limit, and vice versa. SAC keeps track of the consecutive unsuccessful explorations (*restart* on Algorithm 4) to avoid premature convergence (foothill problem) and optimizing in almost flat surfaces (plateau problem). When *restart* reaches the user-defined limit (*R*), the step-sizes and the current position are restarted, as seen in line 26 on Algorithm 4.

Algorithm 4 Algorithm for SAC (minimization case).

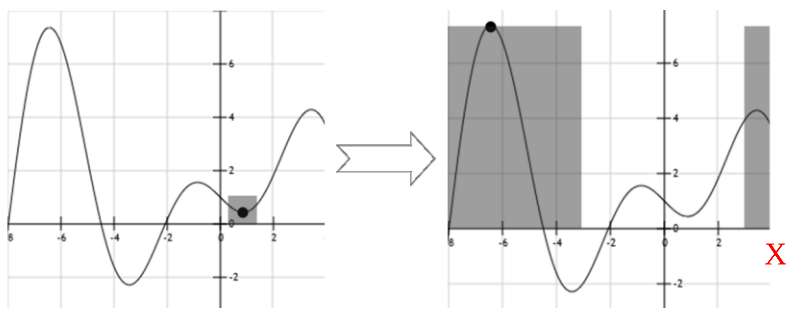
Set X as a random initial point, $Xbest=X$ as the best known solution and $restart=0$;
Set $b_j=B,\ j=1,\dots,D$ as the initial stepsizes;
For($g=1$ To $MaxFes$)
    Set $P_a = \frac{rnd(1,D)}{D}$
    If($flip_j(P_a),\ j=1,\dots,D$)
        $O_j=X_j+rndreal(-b_j,b_j)\times(up_j-low_j)$;
        $If(O_j<low_j)$
            Set $O_j=low_j+(O_j-up_j)$;
        $If(O_j>up_j)$
            Set $O_j=up_j-(low_j-O_j)$;
    Else
        $O_j=X_j$;
    If $f(O)<f(X)$
        Set $restart=0$;
        Set $b_j=rndreal(b_j,B)$ for all j dimensions where $O_j\neq X_j$;
        Set $X=O$
        If($f(O)<f(Xbest)$)
            Set $Xbest=O$;
    Else
        Set $restart=restart+1$;
        If($restart<R$)
            Set $b_j = \left|\frac{x_j-o_j}{up_j-low_j}\right|$ for all j dimensions where $O_j \neq X_j$;
        Else
            Set $restart=0$, $X=O$ and $b_j=B_j,\ j=1,\dots,D$;



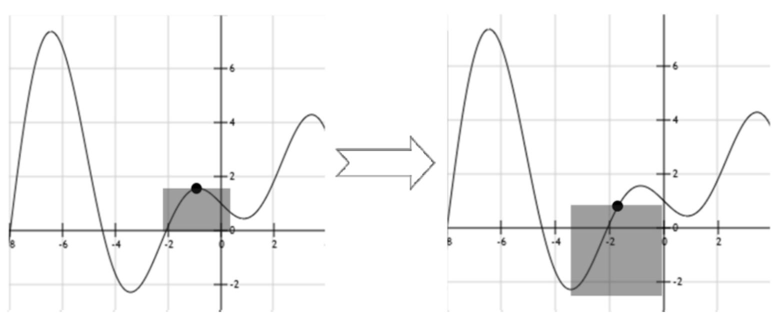
a) SAC could search in any dimension subset with a maximum radius of b_j



b) When failing (white dot) SAC decreases the search space



c) Restart occurs after *R* consecutive failures. Please note that search space is considered connected at the extremes



d) SAC increases the search space when improving

Figure 10 SAC main behaviors (minimization case).

Figure 10 shows how SAC works in 2-D functions. From those figures, we can conclude that:

- SAC is a greedy algorithm, i.e. it looks to improve all the time.
- SAC jumps from one hill to another most of the time.
- The restarting mechanism successfully prevents SAC of getting stuck in a local optimum value.

The proposed algorithm can be considered as a variation of the Elitist Evolution (**EEv**) (Viveros et al. 2012), which uses a very small population (micro-population algorithm).

7.3 Analysis of Test Results

We performed three different tests. First, we performed a comparison of SAC against other state-of-the-art optimizers in some benchmark problems for justifying the use of it. Then, we repeated the comparison but this time in the actual problem of WSD. Finally, we used our proposed modifications of WSD with the Lesk Algorithm. Test results allow us to conclude the following:

- Our proposed optimizer (SAC) is competitive against state-of-the-art algorithms.
- SAC allows the Lesk Algorithm to disambiguate twice as fast while retaining the same quality.
- Our proposed modifications give the Lesk Algorithm a good performance boost.

7.3.1 Comparison of SAC with Some Other State-of-the-Art Optimizers

We performed a comparison of SAC against some selected state-of-the-art optimizers. We tested over 15 well-known benchmark functions (shown in Table 23). These functions can be classified in unimodal (functions having a single local/global optimum value) and multimodal (functions with multiple local optimum values). We decided to show a summary per each type of function per each algorithm. We selected the following average measures:

- **Success rate.** It is the number of trials in which an algorithm found the global minimum value. We performed 30 trials per test function.
- **Coverage.** It is the percentage of functions in which an algorithm found a global minimum value in at least 1 trial.
- **Time.** It is the average number of function evaluations needed by an algorithm to found the global minimum value in the success trials.
- **Error.** It is the average distance between an algorithm's answer and the global minimum value in the failure trials.

Table 23 Test functions (Mezura et al. 2006).

Unimodal functions		Multimodal functions	
f_{sph}	Sphere model	f_{sch}	Generalized Schwefel's problem 2.26
$f_{2.22}$	Schwefel's problem 2.22	f_{ras}	Generalized Rastrigin's function
$f_{2.21}$	Schwefel's problem 2.21	f_{ros}	Generalized Rosenbrock's function
f_{stp}	Step function	f_{ack}	Ackley's function
f_{qtc}	Quartic function	f_{grw}	Generalized Griewank's function
$f_{1.2}$	Schwefel's problem 1.2	f_{sal}	Salomon's function
		f_{whi}	Whitley's function
		$f_{p1,2}$	Generalized penalized functions

We performed a comparison against the following approaches:

- **EEv**: The best micro-population algorithm so far as the author’s knowledge and SAC's precursor.
- **μ-PSO** (Fuentes & Coello 2007): An approach competitive to PSO.
- **Simple Adaptive Differential Evolution (SaDE)** (Qin et al. 2009) selected because it is the best differential evolution (Storn & Price 1997) variant so far. It is simple and competitive with other state-of-the-art techniques.

- **Restart CMA-ES** (Auger and Hansen 2005) selected for measuring the gap against a technique that uses Hessian and covariance matrices. This was also the best technique on CEC 2005 special session on real-parameter optimization.

Table 24 shows a summary of the test results. Tests allow us to confirm that:

- SAC was the algorithm having the best average speed in the success trials and the one having the smallest average error in failure trials.
- SAC has a competitive performance in global optimization problems with high dimensionality. It is competitive against state-of-the-art approaches like SADE and CMA-ES.
- SAC could solve 80% of the functions at least one time. It only had less coverage than SADE which obtained 87%.
- SAC has a good success rate of 72%. It had less success rate than SADE (74%) and CMA-ES (73%).

Table 24 Summary of test results for functions with a $D=30$. Average measures were calculated by function type: unimodal (7 functions), multimodal (8 functions) and global (15 functions). SAC's global results have its ranking as a prefix.

Success rate	**SAC**	**μ-PSO**	**EEv**	**CMA-ES**	**SADE**
Unimodal	83%	67%	67%	100%	83%
Multimodal	64%	37%	51%	55%	69%
Global	72% (3rd)	49%	58%	73%	**74%**
Coverage					
Unimodal	83%	67%	67%	100%	83%
Multimodal	78%	44%	67%	67%	89%
Global	80% (2nd)	53%	67%	80%	**87%**
Time					
Unimodal	4.6E+4	1.0E+5	4.3E+4	1.0E+4	4.9E+4
Multimodal	5.6E+4	1.1E+5	1.2E+5	1.5E+5	7.8E+4
Global	**5.5E+4 (1st)**	1.1E+5	9.2E+4	7.8E+4	6.7E+4
Error					
Unimodal	3.2E-4	1.6E-2	7.9E-3	0.0	4.5
Multimodal	4.1	2.4E+2	3.1E+2	2.5E+3	1.8E+1
Global	**4.1 (1st)**	2.4E+2	3.1E+2	2.5E+3	1.8E+1

7.3.2 Testing SAC with Lesk Algorithm

We performed a comparison of optimizers with the goal of seeking the best optimizer to pair with the Lesk algorithm. We observed that all optimizers gave the same WSD quality (E.G. the algorithm obtained the same performance in both test sets). The only difference was the time needed to reach that result. You can observe a time comparison made in Senseval 2 test set in Table 25. SAC was the fastest technique: it required half of the time less than the other optimizers.

Table 25 Comparison of the time needed by the Lesk Algorithm when using different optimizers.

Technique	Time per word in seconds
SAC	**0.24**
Differential Evolution	0.46
SaDE	0.45
PSO	0.79

7.3.3 Integrating Our Modifications with the Lesk Algorithm

We tested our modifications with the Lesk Algorithm in the Senseval 2 and 3 test sets. Table 26 shows the performance of the Lesk Algorithm when using our modifications. This table allows us to confirm that our modifications heavily increase precision of the Lesk algorithm. Also, we confirmed that the Simplified Lesk Algorithm has slightly more precision than the Lesk Algorithm.

Table 26 Lesk Algorithm with our modifications.

WSD method	Senseval-2			Senseval-3		
	P	**R**	**F1**	**P**	**R**	**F1**
Our SLA	**82.4**	40.4	**54.2**	**81.0**	32.7	**46.6**
Our Lesk	78.9	40.2	50.9	78.5	33.1	46.5
Lesk	48.1	**46.0**	47.0	38.4	**36.7**	37.8

Chapter 8. Conclusions

This work proposes and analyzes modifications for increasing the performance of the Simplified Lesk Algorithm and other dictionary based algorithms. We have reached the following conclusions

- We have shown that the commonly accepted technique of using a small closest context for selecting a window does not benefit the Simplified Lesk Algorithm. In fact, words that are necessary for making correct decisions are present in the document, but remain unused when using a small window size (like 4 words or the sentence). It is shown that the algorithm's performance can be significantly improved with rather simple modifications in the context window selection procedures. The proposed context window can be summarized as constructing a small context window containing just overlapping words without the target or duplicates.
- It was detected that words like *be*, *have* or *not* often produce overlaps and often lead algorithms to select the wrong senses. However, removing them is not the proper solution -we will try to create a better solution in our future work.
- WSD methods can attain high precision when solving domain words. These words are known to have one sense per discourse (OSD). Around half of the words in a document are domain words.
- Using co-occurring words increases performance of the Simplified Lesk Algorithm. Co-occurring words lead to a performance rivaling the use of corpus samples.
- Using all the proposed modifications lead to a 80% precision –almost as good as human annotators. That means that our algorithm has good precision when disambiguating domain words.

References

Agirre E, Rigau G (1996) Word Sense Disambiguation using Conceptual Density Proceedings of COLING'96, 16-22. Copenhagen (Denmark).

Agirre, E. and Edmonds, P. (2006). Word sense disambiguation. Springer.

Agirre E, Lopez de Lacalle O, Fellbaum C, Hsieh S, Tesconi M, Monachini M, Vossen P, Segers R (2010) SemEval-2010 task 17: All-words word sense disambiguation on a specific domain, Proceedings of the 5th International Workshop on Semantic Evaluation, p.75-80, July 15-16, Los Angeles, California

Auger, A., Kern, S. and Hansen, N. (2005) A Restart CMA Evolution Strategy with Increasing Population Size. CEC 2005 Special Session on Real-Parameter Obtimization. Nanyang Technol. Univ., Singaporem IIT Kanpur, India, 2005.

Banerjee S, Pedersen T (2002) An Adapted Lesk Algorithm for Word Sense Disambiguation Using WordNet, Lecture Notes In Computer Science; Vol. 2276, Pages: 136 - 145, 2002. ISBN 3540432191.

Budanitsky A, Hirst G (2001) Semantic distance in Word-Net: An experimental, application-oriented evaluation of five measures. In Proceedings of the NAACL Workshop on WordNet and Other Lexical Resources, Pittsburgh.

Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. Computational linguistics, 16(1). pp. 22--29.

Cimiano, P., Hotho, A. and Staab, S. (2005). Learning Concept Hierarchies from Text Corpora using Formal Concept Analysis. J. Artif. Intell. Res.(JAIR), 24. pp 305--339.

Cotton S, Edmonds P, Kilgarriff A, Palmer M (2001) “SENSEVAL-2.” Second International Workshop on Evaluating Word Sense Disambiguation Systems. SIGLEX Workshop, ACL03. Toulouse, France.

Decadt B., Hoste V., Daelemans W. and Van den Bosch A. (2004) GAMBL, Genetic Algorithm Optimization of Memory-Based WSD. In: Proc. of ACL/SIGLEX Senseval-3.

References

Edmonds P (2000) Designing a task for SENSEVAL-2. Tech. note. University of Brighton, Brighton. U.K.

Fuentes-Cabrera, J-C. and Coello-Coello, C-A (2007) Handling Constraints in Particle Swarm Optimization using a Small Population Size. LNCS. MICAI 2007: Advances in Artificial Intelligence. Springer-Verlag. vol 4827. pp. 41-51.

Gale W. A., Church K. W. and Yarowski D. (1991) One sense per discourse. In: Proc. of HLT. pp. 233--237. ACM.

Gelbukh A, Sidorov G, Han S (2005) On Some Optimization Heuristics for Lesk-Like WSD Algorithms. A. Montoyo et al. (Eds.) Natural Language Processing and Information Systems. 10th International Conference on Applications of Natural Languages to Information Systems, NLDB-2005, Alicante, Spain, 2005. Lecture Notes in Computer Science, N 3513, ISSN 0302-9743, Springer, 2005, pp. 402–405.

Hoste V., Kool A. and Daelemans W. (2001) Classifier optimization and combination in the English all words task. In: Proc. of ACL/SIGLEX Senseval-2.

Jiang J, Conrath D (1997) Semantic similarity based on corpus statistics and lexical taxonomy. In Proceedings of the International Conference on Research in Computational Linguistics, Taiwan.

Kilgarriff A (1997) I don't believe in word senses. Computers and the Humanities. 31(2), pp. 91–113.

Kilgarriff A, Rosenzweig J (2000) English Framework and Results Computers and the Humanities 34 (1-2), Special Issue on SENSEVAL.

Lesk M (1986) Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In Proc. of SIGDOC-86: 5th International Conference on Systems Documentation, Toronto, Canada.

Lin D (1998) An information-theoretic definition of similarity. In Proceedings of the 15th International Conference on Machine Learning, Madison, WI.

Magnini B., Strapparava C., Pezzulo G. and Gliozzo A. (2001) Using domain information for word sense disambiguation. In: Proc. of ACL/SIGLEX Senseval-2.

Martínez D. and Agirre E. (2000) One Sense per Collocation and Genre/Topic Variations. In: Proc. of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora.

Mezura-Montes, E., Coello Coello C.A. and Velazquez, R J (2006) A comparative study of differential evolution variants for global optimization. Proceedings of the 8th annual conference on Genetic and evolutionary computation. pp. 485-492.

Mihalcea R. and Moldovan D. (2001) Pattern learning and active feature selection for word sense disambiguation. In: Proc. of ACL/SIGLEX Senseval-2.

Mihalcea R, Edmons P (2004) Senseval-3 Third International Workshop on Evaluating of Systems for the Semantic Analysis of Text. Association for Computational Linguistics. ACL 04. Barcelona, Spain.

Mihalcea R (2006) Knowledge Based Methods for Word Sense Disambiguation, book chapter in Word Sense Disambiguation: Algorithms, Applications, and Trends, Editors Phil Edmonds and Eneko Agirre, Kluwer.

Mihalcea R (2007) Using Wikipedia for Automatic Word Sense Disambiguation. In Proc. of the North American Chapter of the Association for Computational Linguistics (NAACL 2007), Rochester.

Miller G., Chodorow M., Landes S., Leacock C. and Thomas R. G. (1994). "Sing a Semantic Concordance for Sense Identification. In: Proc. of ARPA Human Language Technology Workshop. pp.~240--243. ACM.

Miller G (1995) WordNet: A Lexical Database for English. Communications of the ACM Vol. 38, No. 11: 39-41.

Navigli R, Litkowski K, Hargraves O (2007) SemEval-2007 Task 07: Coarse-Grained English All-Words Task. Proc. of Semeval-2007 Workshop (SemEval), in the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007), Prague, Czech Republic.

Navigli R (2009) Word Sense Disambiguation: a Survey. ACM Computing Surveys, 41(2), ACM Press, pp. 1-69.

Pradhan S, Loper E, Dligach D, Palmer M (2007) SemEval-2007 Task 17: English lexical sample, SRL and all words. Proc. of Semeval-2007 Workshop (SEMEVAL), in the 45th Annual Meeting of the Association for Computational Linguistics (ACL 2007), Prague, Czech Republic.

Qin, K. A., Huang, V. L. and Suganthan, P. N. (2009) Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization. IEEE Transactions on Evol. Comput. Vol. 13, No. 2. IEEE Press. pp. 398-417.

Rada R, Mill H, Bicknell E, Blettner M (1989) Development and application of a metric on semantic nets, in IEEE Transactions on Systems, Man and Cybernetics, vol. 19, no. 1, pp 17-30.

Resnik P (1995) Using information content to evaluate semantic similarity. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada.

Schwab, D., Goulian, J., and Tchechmedjiev, A. (2013). Worst–case complexity and empirical evaluation of artificial intelligence methods for unsupervised word sense disambiguation. In International Journal of Web Engineering and Technology. 8(2). pp~124--153.

Sinha R, Mihalcea R (2007) Unsupervised Graph-based Word Sense Disambiguation Using Measures of Word Semantic Similarity, in Proceedings of the IEEE International Conference on Semantic Computing (ICSC 2007), Irvine, CA.

Storn, R. and Price, K. (1997) Differential Evolution - a simple and efficient heuristic for global optimization. Journal of Global Optimization, Volume 11, Number 4, Springer Netherlands. pp. 341-359.

Strapparava C., Gliozzo A. and Giuliano C. (2004) Pattern Abstraction and Term Similarity for Word Sense Disambiguation: IRST at Senseval-3. In: Proc. of ACL/SIGLEX Senseval-3

Toutanova K, Manning C D (2000) Enriching the Knowledge Sources Used in a Maximum Entropy Part-of-Speech Tagger. In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP/VLC-2000), pp. 63-70.

Vasilescu F, Langlais P, Lapalme G (2004) Evaluating Variants of the Lesk Approach for Disambiguating Words. LREC, Portugal.

Viveros Jiménez, F., Mezura Montes, E., & Gelbukh, A. (2012). Empirical analysis of a micro-evolutionary algorithm for numerical optimization. Int. J. Phys. Sci, 7, pp. 1235-1258.

Winston, P.H. (1992) Artificial Intelligence. Addison-Wesley Publishing Company, Reading, MA, third edition.

Wu Z, Palmer M (1994) Verb semantics and lexical selection. In Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, Las Cruces, New Mexico.

Yarowsky D (1992) Word-sense disambiguation using statistical models of Roget's categories trained on large corpora. In Proc. of the 14th conference on Computational linguistics (COLING).

Yarowsky D. (1995) Unsupervised Word Sense Disambiguation Rivaling Supervised Methods". In: Proc. of ACL 2007. pp. 189-–196.

Appendix

1 Usage of the API

Our implementation in Java is available at http://fviveros.gelbukh.com as WSD Java API project. It also contains all necessary data from Senseval-2 and Senseval-3 English-All-Words test sets, as well as the sense inventory from WordNet. It is an open source project. It is very easy to use and gives the possibility to change all parameters mentioned in this paper. It also contains the implementation of other dictionary-based methods mentioned before.

More specifically, it has the following characteristics of the user interface:

- Easy setup of many parameters such as window size, number of senses retrieved from the dictionary, back-off strategy, tie solving method and conditions for retrieving window words.
- Easy configuration in a single XML file.
- Output is generated in a simple XLS file by using JExcelApi.

The API is licensed under the GNU General Public License. Source code is included.

Using the library for your experiments only requires adjusting **config.xml** file and then typing **java -jar cicwsd.jar** in your command line.

Configuration instructions are inside the **config.xml** file.

The sample configuration file represented in Table 27 corresponds to comparison of the graph indegree, the conceptual density, the traditional Simplified Lesk Algorithm and the proposed modifications of the Simplified Lesk Algorithm over Senseval-2 test set. It means that all these tests will be conducted with the given parameters. The results will be stored in the results.xls file.

Table 27 Sample configuration file corresponding to the comparison of graph indegree, conceptual density, Simplified Lesk and our proposed modification of Simplified Lesk over Senseval-2 test set. Results will be stored in results.xls file.

```
<run>
  <dict sources="WNGlosses;WNSamples"/>
  <testbed senses="All">
    <docs src="Resources/senseval2" prefix=""/>
  </testbed>
  <xls src="results.xls" detail="false">
    <test>
      <algorithm disambiguation="GraphInDegree" backoff="none"
              windowSize="4" tie="MFS"/>
      <condition type="none"/>
    </test>
    <test>
      <algorithm disambiguation="ConceptualDensity" backoff="none"
              windowSize="4" tie="MFS"/>
      <condition type="none"/>
    </test>
    <test>
      <algorithm disambiguation="SimplifiedLesk" backoff="none"
              windowSize="4" tie="MFS"/>
      <condition type="none"/>
    </test>
    <test>
      <algorithm disambiguation="SimplifiedLesk" backoff="none"
              windowSize="4" tie="MFS"/>
      <condition type="NoTarget"/>
      <condition type="NoDuplicates"/>
      <condition type="IsUseful:SimplifiedLesk"/>
    </test>
  </xls>
</run>
```

2 Structure of the API

The API is classified in the following packages:

- **wordnet:** Allows a user to retrieve information from WordNet 3.0 database files.
- **wordnet.pairs:** Auxiliary classes for storing WordNet memory mappings.
- **wsd.disambiguation:** Classes for creating disambiguation algorithms.
- **wsd.semcor:** Classes for loading SEMCOR format files.
- **wsd.semcor.pruning:** Classes for creating lexicon filters.
- **wsd.testing:** Classes for testing Word Sense Disambiguation algorithms over SEMCOR test files.
- **wsd.windowing:** Classes for adding window selection conditions.