

The Pennsylvania State University
The Graduate School

WORD SENSE DISAMBIGUATION

A Thesis in
Computer Science
by
Saket Kumar

© 2015 Saket Kumar

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

December 2015

The thesis of Saket Kumar was reviewed and approved* by the following:

Omar El Ariss
Assistant Professor of Computer Science
Thesis Advisor

Thang N. Bui
Associate Professor of Computer Science
Chair, Mathematics and Computer Science Programs

Linda M. Null
Associate Professor of Computer Science
Graduate Coordinator

Jeremy J. Blum
Associate Professor of Computer Science

Sukmoon Chang
Associate Professor of Computer Science

*Signatures are on file in the Graduate School.

Abstract

Humans can infer meaning through the use of not only the definition of a word, but also, where one word might have various conflicting definitions, based on their experiences and the text's context and domain. Word Sense Disambiguation (WSD) is the problem of finding the most appropriate meaning of a word in a particular context. The functional importance of WSD lies in processing the sequence of words by pinpointing their meaning without the need for human intervention. It is crucial for many applications such as translation, summarization, information retrieval, and many other natural language applications.

We introduce an unsupervised knowledge-source approach for word sense disambiguation using a bee colony optimization algorithm. We also present several variations to our bee colony approach that improve the overall performance of the algorithm. Our results are compared with recent unsupervised approaches such as ant colony optimization, genetic algorithms, most frequent sense, and simulated annealing.

Table of Contents

List of Figures	vii
List of Tables	viii
Chapter 1	
Introduction	1
Chapter 2	
Preliminaries	5
2.1 Problem Definition	5
2.2 WordNet	6
2.3 Previous Work	8
2.3.1 Supervised Approach	8
2.3.2 Unsupervised Knowledge-Source Approach	9
2.3.2.1 Dictionary-based Algorithms	10
2.3.2.2 Concept Hierarchy-based Algorithms	12
2.3.2.3 Heuristic-based Algorithms	13
Chapter 3	
Bee Colony Optimization	16
3.1 Bees in nature	16
3.2 BCO Algorithm	18
Chapter 4	
Word Sense Disambiguation - Bee Colony Optimization	21
4.1 Initialization	24

4.2	Forward Phase	27
4.3	Backward Phase	29
Chapter 5		
	Modifications	33
5.1	Document Level	33
5.2	Sentence Level	34
5.3	Voting	36
5.4	Hybrid	36
Chapter 6		
	Experimental Results	38
6.1	Data Set	38
6.2	Evaluating The System	40
6.2.1	Baseline	40
6.3	Results	42
6.3.1	WSD-BCO	42
6.3.2	BCO-D-NA/VB/J-TS	43
6.3.3	BCO-GS-NA/VJ-TS-I	43
6.3.4	VBCO-D-NA/VB/J-TS	43
6.3.5	VBCO-D-NA-TS-HY	44
6.4	Analysis	45
Chapter 7		
	Wordnet 3.0	47
7.1	BCO-S	48
7.2	BCO-D-NA/VB/J-TS	48
7.3	BCO-GS-NA/VJ-TS-I	49
7.4	VBCO-D-NA/VB/J-TS	49
7.5	VBCO-D-NA-TS	50
Chapter 8		
	Conclusion	51

<i>TABLE OF CONTENTS</i>	vi
Appendix	53
WSD-BCO Algorithm	53
References	60

List of Figures

2.1	Word Sense Disambiguation Problem	6
2.2	WordNet Example for Word “shut”	7
3.1	A recruiter bee performing the waggle dance	17
3.2	Waggle Dance Overview	18
3.3	Forward/backward phase	19
4.1	WSD-BCO Overview	24
4.2	Prepare Word Frequency	25
4.3	Prepare Extended Sense	26
4.4	Initiate Bees	27
4.5	Pickup Roulette Wheel	28
4.6	WSD-BCO Relatedness	28
4.7	Forward Phase Overview	29
4.8	Recruitment Roulette Wheel	31
4.9	Backward Phase Overview	32
5.1	The modified document splitting step from the initialization phase of Figure 4.1 for BCO–D–NA/VB/J–TS	34
5.2	The modified document splitting step from the initialization phase of Figure 4.1 for BCO–GS–NA/VJ–TS–I	35
5.3	Voting algorithm for BCO–D–NA/VB/J–TS	36
5.4	The modified document splitting step from the initialization phase of Figure 4.1 for VBCO–D–NA–TS–HY	37

List of Tables

4.1	BCO vs WSD-BCO	23
6.1	Target Words in Each Document	39
6.2	Most Frequent Sense (WordNet 2.1)	41
6.3	BCO-S (Rec=3, WordNet 2.1)	42
6.4	BCO-D-NA/VB/J-TS (Rec=3, TS=5, WordNet 2.1)	43
6.5	BCO-GS-NA/VJ-TS-I (GS=5, I=5, TS=5, Rec=3, WordNet 2.1)	44
6.6	VBCO-D-NA/VB/J-TS (V=200, Rec=3, WordNet 2.1)	44
6.7	VBCO-D-NA-TS-HY (V=200, Rec=3, WordNet 2.1)	45
6.8	Comparison of WSD Systems	46
7.1	Most Frequent Sense Result (WordNet3.0)	47
7.2	BCO-S (Rec=3, WordNet 3.0)	48
7.3	BCO-D-NA/VB/J-TS (Rec=3, TS=5, WordNet 3.0)	48
7.4	BCO-GS-NA/VJ-TS-I (GS=5, I=5, TS=5, Rec=3, WordNet 3.0)	49
7.5	VBCO-D-NA/VB/J-TS (V=200, Rec=3, WordNet 3.0)	49
7.6	VBCO-D-NA-TS (V=200, Rec=3, WordNet 3.0)	50

Acknowledgements

I highly appreciate and am thankful to my thesis advisor, Dr. Omar A El Ariss, for his valuable suggestions, insights and foremost having patience during the whole thesis process. I am thankful and grateful to the committee members for going through my work and defense and providing important feedbacks. Lastly, I thank my parents for providing me the opportunity to learn.

Chapter 1

Introduction

Semantics, or the meaning of a given sentence is an important feature in communication and knowledge acquisition. Humans are good at understanding the meaning of given text, but how do we do that? Is it simply looking up the definition of each word one at a time or is it more than that? Also, is it possible for a program to automate the process of language understanding? A word has one or more different senses attached to it. Each and every *sense* of a word is represented by a definition, a list of synonym, and an example of sense usage. The process of assigning the correct definition or the correct sense to the word that we are disambiguating is called Word Sense Disambiguation (WSD).

Algorithms for WSD fall into two main groups, supervised and unsupervised. Supervised approaches for WSD make use of machine learning algorithms such as classification and feature recognition. The resultant classification models are trained on an annotated corpus, which is a collection of data that has been tagged correctly by a linguist. The process of manual sense annotation of words to create a large corpus of examples is laborious and requires the presence of a linguist. On the other hand unsupervised approaches use information from an external knowledge source to determine the exact sense of these words. The knowledge source can be a machine readable dictionary, or an organized network of words or semantics [1]. In addition, the knowledge source should be periodically updated as there are domain changes, new senses are introduced, and new words are added. These changes impact the semantic networks and require them to be updated. WSD falls in the class of AI-complete problems, which means it is at least as hard as most

of the difficult problems in AI, and by parallel comparison, in complexity theory it is NP-complete [2].

There are many distinct reasons why WSD is a difficult problem [2]:

- It is difficult to define the senses of words and the level of detail represented by a particular sense with respect to sense usage.
- It is difficult to determine if the word should be disambiguated for a more generic sense or for a finer sense in a given context.
- It is difficult to determine how much context to use to achieve the most accurate disambiguation.

Consider a system developed to solve the WSD problem. The performance of a WSD system is contingent on the knowledge source. Furthermore, one can relate to the importance of this problem because there are many real world applications based on it. Solving the WSD problem in an efficient way will have a huge effect on the efficiency of systems such as document warehousing, response generation, machine translations, and document summarization.

Let us take an example that demonstrates the WSD problem. Consider the following sentence: “His parties fall on every Saturday night.” It is easy to see that the sense of the words “fall,” “parties,” and “Saturday” are clear from the current context. The WSD system interprets a suitable sense of the word “fall” by using a knowledge source. There are twelve distinct noun and thirty-two distinct verb senses for the word “fall” [3]. A few of them include:

- *fall1* - lose office or power
- *fall2* - be due
- *fall3* - slope downward
- *fall4* - occur at a specified time or place

The WSD system is unable to decide which sense to pick for the word “fall” without any additional information based on its context. It is to be noted here that a machine or a human cannot determine the exact sense of a particular word without knowing its context. We introduce the context by putting emphasis on the

surrounding words, to give some contextual perspective and determine the correct sense of “fall.” The WSD system tries to figure out the appropriate sense of the word “fall” in relation to the surrounding word “parties.” The system will make use of all the senses and definitions of “parties” to help with the disambiguation process. There are five noun senses and one verb sense for the word “parties” [3]. A few of them include:

- *party1* - an organization to gain political power
- *party2* - a group of people gathered together for pleasure
- *party3* - a band of people associated temporarily in some activity
- *party4* - a person involved in legal proceedings

In this limited context, the WSD system has a stronger claim for *fall1* being the appropriate sense of the word “fall.” It can relate to *fall1* where “losing office or power of an organization” is being referred to here since the word “power” occurs in the definition of *fall1* and *party1*. The system can only claim that *fall1* is the correct sense in the short context that was provided to it. It is evident that context plays a significant role in deciding the correct sense of the word “fall.” We increase the context by providing another surrounding word “Saturday” to the WSD system. The newly added word gives a deeper insight to the system for determining the correct sense of “fall.” There is only one noun sense of the word “Saturday” [3]:

- *Saturday1* - the seventh and last day of the week; observed as the Sabbath by Jews and some Christians

The term “Saturday” is a specified time and therefore can be related to *fall4*. The WSD system continues with this process by adding “night” to the context. Similar to “Saturday,” “night” is also a specified time and can be related to sense *fall4*. Now the WSD system’s claim that *fall1*, which was the previously selected sense, is weaker. This process of assigning the appropriate sense *fall4* to the word “fall” in a given context of words is called the WSD problem.

In this thesis, we introduce an unsupervised knowledge-source approach for WSD. The algorithm is based on Bee Colony Optimization (BCO) and is therefore

called WSD-BCO. The approach exploits the relationship between the word and the context in which it occurs. The algorithm uses a database of semantic networks called Wordnet 2.1 [3] as a knowledge source. WSD-BCO endeavors to solve the WSD as an optimization problem. We generate bees based on the word chosen to represent the hive. Bees travel to other words in the document and, based on their path, assign senses to other words. We present other variations of WSD-BCO that differ in the context supplied to WSD-BCO. The strength of the relationship between senses and surrounding words is determined by senses' parts of speech, definitions, entailments, synonyms, and attributes. The algorithm uses a dataset from SemEval 2007 [4], an international workshop on semantic evaluations. The test data is composed of five different documents ranging from general topics to more specific topics. The results are compared against an ant colony optimization (ACO) algorithm, a genetic algorithm (GA), a simulated annealing (SA), and the Most Frequent Sense (MFS) algorithm. MFS acts as a baseline for all the approaches.

The rest of the thesis is structured as follows. In Chapter 2, we discuss the problem in more detail. We also describe the previous work done on WSD. In Chapter 3, we discuss the general Bee Colony Optimization approach. In Chapter 4, we introduce basic Bee Colony Optimization with respect to WSD. The various phases of WSD-BCO include initialization, a forward phase, and a backward phase. In Chapter 5, we discuss the variations of our approach for improving the efficiency. In Chapter 6, we discuss the Semeval 2007 dataset and experimental results of our algorithm, and then we compare our results with other approaches such as ACO, GA, SA and MFS. In Chapter 7, we show the results of our various approaches using Wordnet 3.0. Appendix A contains the WSD-BCO algorithm.

Chapter 2

Preliminaries

2.1 Problem Definition

In the previous chapter, we saw an example of how the word “fall” was disambiguated and its proper sense was determined. We defined Word Sense Disambiguation (WSD) as a problem of figuring out the correct sense of a word in a given context. A more precise definition of WSD that we are going to use in this thesis is that, given text that is composed of a set of n target words, the task is to determine the correct sense of all n words. We call the word that we are currently disambiguating the *target word*, while the *context* is defined as the surrounding words which occur around the target word.

For example suppose we have the sentence, “This code may be a modification of existing source or something completely new” that we need to disambiguate. First we remove all the stop words from the sentence. *Stop words* are words that occur frequently in text irrespective of a particular domain. For example, “who” and “or” are stop words. The words that need to be disambiguated are “code,” “be,” “modification,” “existing,” “source,” “completely,” and “new.” Let us assume that the WSD system is at the stage of disambiguating the fourth word “existing.” This word is the target word. We are going to use the other surrounding words as context to help us find the correct sense for “existing.” Figure 2.1 shows that the target word “existing” is disambiguated with the help of the surrounding words “code,” “be,” “modification,” “source,” “completely,” and “new.” The *window length* represents the number of words on the right or left of the target word. The

window length defines the surrounding words in the frame of reference for the target word. In the example of Figure 2.1, the window length that is used is three.



Figure 2.1: Word Sense Disambiguation Problem

There are two main approaches to solve WSD: supervised and unsupervised [2]. In the supervised approach, the available data is divided, in an appropriate ratio, into training data and test data, and the WSD task becomes a classification problem. This approach requires a large corpus of data that is tagged with correct senses. Classifiers such as decision lists, decision trees, Naive Bayes, neural networks, and support vector machines are popular for this approach. It has been observed that supervised WSD methods provide better results than unsupervised approaches for WSD [2]. The main disadvantage of the supervised approach is the need for sense-annotated data.

An unsupervised approach has the advantage of not depending on hand-annotated data [2]. The general idea of this approach is based on the fact that the correct sense of a target word will have similar words when used in the same domain. For example, the target word “code” in the domain of computer science may have the words “computer” and “program” in the surrounding text. The WSD problem is then reduced into measuring the similarity of the context words with the sense in question. Instead of learning from a hand-annotated data, it makes use of a knowledge source such as a dictionary or a semantic network. In this thesis, we are going to focus on an unsupervised approach for determining the correct senses of the target words.

2.2 WordNet

WordNet [3] is the most popular and widely used knowledge source for unsupervised WSD approaches. WordNet is a lexical database whose structure allows it to be accessed using a variety of tools and programming languages. It is segregated

into four databases based upon the parts of speech: noun, verb, adverb, and adjective. Each database consists of a set of lemmas, where a *lemma* is the dictionary form of a word. For example, the words “sing,” “sang,” “sung,” and “singing” all have the same lemma, which is “sing.” Each lemma in the database has a set of senses. Each sense has a definition and a list of synonym words. In addition to the definition, a sense is accompanied by one or more sentences that demonstrate the sense usage. For example, Figure 2.2 shows an example of one sense for “shut,” where “close” is a synonym.

- **S: (v) close, shut (move so that an opening or passage is obstructed; make shut) "Close the door"; "shut the window"**
 - [verb group](#)
 - [direct troponym](#) / [full troponym](#)
 - [cause](#)
 - [antonym](#)
 - [derivationally related form](#)
 - [phrasal verb](#)
 - [sentence frame](#)

Figure 2.2: WordNet Example for Word “shut”

Concepts in WordNet are represented using synonym sets (synsets). A *synset* is a set of near synonyms for one particular sense. There are 117,000 synsets and 155,000 words in WordNet. Synsets are interlinked together based on their conceptual relation. The advantage of using WordNet is that all of the entries are independent of the domain; that is the entries are not domain specific. The structural layout is mostly based on the *synonymy* of the words. Synonymy is defined when different words have similar meaning. For example, the words “big” and “large.” Noun synsets can be arranged in the form of a tree according to their noun relations. Noun relations are determined based on the concept of *hyperonymy* and *hyponymy*. Hyperonymy gives the more general concept of a particular word, whereas hyponymy gives the more specific concept of the word. For example, the word “car” is a hyponymy of “vehicle,” and “vehicle” is a hyperonymy of “car.” Verb relations are arranged on the concept of *troponyms*. Troponyms give the more succinct concept of a particular verb at the bottom of the tree. For example, the word “stroll” is a troponym of “walk.” Adjective relations are arranged based on the concept of *antonymy* and *pertainyms*. Antonymy arranges two opposite

concepts together, which in turn are linked to a synonym concepts to provide an indirect antonym. For example, the word “increase” is an antonym of “decrease.” Pertainyms give the classifying adjectives synset that refer to the noun synset from which it has been obtained. For example, the word “interaction” is a pertainym of “mutual” [5].

There are multiple versions of WordNet. Most of the research on WSD has been done using WordNet 2.1, although a new version WordNet 3.0 is also available. The difference between WordNet 3.0 and WordNet 2.1 is described in detail in Chapter 7. We have included the results of our approach on both WordNet 2.1 and WordNet 3.0.

2.3 Previous Work

We discuss in detail some of the supervised and unsupervised techniques used to solve WSD.

2.3.1 Supervised Approach

NUS- ML [16] is a supervised approach that uses a hierarchical three-level Bayesian model in combination with lexical, syntactic, and topic features of the words for the disambiguation process. The lexical features used are: parts of speech, positions with respect to other words, word forms, and lemmas of the target word. Syntactic features such as dependencies are also used by the model. Finally, topic features used include the general or fine context and a window length. This approach extracts the information from the global context with the help of latent Dirichlet allocation. Latent Dirichlet allocation is a probabilistic model for the collection of discrete data. It is used in the field of text classification and document modeling [2]. SemCor, which is a tagged corpus, is used as the training data. Apart from the probabilistic models (Naive Bayes algorithm), there are other models, such as k -nearest neighbor, that use previously tagged examples to help disambiguate the targeted text [2].

Structural semantic interconnections (SSI) [17] represent the word senses using graphs and determine the semantic similarity based on context-free grammars.

The basis of classification of a sense is the number and distinctness of the interconnections in the network. The graph representation is built using the relations in WordNet, the domain labels, the annotated corpus and the dictionary of collocations. The layout of the word senses as a graph demonstrates a strong claim during manual sense tagging. This algorithm performs well on domain specific concepts [17].

A corpus-based supervised approach is heavily dependent upon hand-annotated data. It requires a linguist to tag the text, which is costly and difficult to maintain over a period of time. Distributional and translational equivalence methods are used to reduce the dependency on hand annotation. The distributional method takes advantage of the fact that words have identical sense in similar context. The translational equivalence method uses parallel corpora that help map the sense from the initial language to the final language [18]. Multilingual word sense disambiguation uses Wikipedia to form a mono-sense classification. It exploits the interlingual links between the five languages: English, French, German, Spanish, and Italian. BabelNet, on the other hand, combines WordNet and Wikipedia sense inventories with the support of six languages [19].

2.3.2 Unsupervised Knowledge-Source Approach

An unsupervised knowledge source approach has the ability to disambiguate all the words in the text irrespective of the domain. This approach has two important methods: dictionary-based and thesauras-based methods. Local context can be used in both approaches, which refers to the syntactic relation between the words in the context. It is a knowledge-intensive method that does automatic sense tagging using a relatedness measure. The notion behind the relatedness measure is to find how close two words are in the same context and to reveal the correct meaning of the words. There are two distinct approaches in which relatedness can be measured [6]:

- Machine readable dictionaries such as WordNet, to exploit the definition of the words and examples
- Concept hierarchies, which exploit the layout of words in the dictionary

These approaches have been extensively used and are discussed below.

2.3.2.1 Dictionary-based Algorithms

Michael Lesk introduced a dictionary-based algorithm [9] that counts the number of *overlapping words* between the definition of the target word sense and all the senses of the surrounding words in context. An overlapping word is defined as the common word that occurs in two sets of words. This process is repeated for all the senses of the target word. Stop words are not considered as overlapping words.

In the original Lesk algorithm, when a target word is disambiguated with its surrounding words, the definition and examples of each sense of the target word are compared to the definition and examples of each sense of the surrounding words. The sense for the target word with the maximum overlapping of words is considered to be the correct sense [9]. For example, suppose the words “computer,” “code,” and “instruction” occur together in a sentence. Suppose further that the word “instruction” is the target word for which the sense needs to be labelled; the words “code” and “computer” are the surrounding words. For simplicity, we are going to consider the definitions without the example sentences. There are five distinct noun senses for the word “instruction” [3]:

- *instruction1* - a message describing how something is to be done
- *instruction2* - the activities of educating or instructing; activities that impart knowledge or skill
- *instruction3* - the profession of a teacher
- *instruction4* - (computer science) a line of code written as part of a computer program
- *instruction5* - a manual usually accompanying a technical device and explaining how to install or operate it

There are three distinct noun and two distinct verb senses for the word “code” [3]:

- *code1* - a set of rules or principles or laws (especially written ones)
- *code2* - a coding system used for transmitting messages requiring brevity or secrecy
- *code3* - (computer science) the symbolic arrangement of data or instructions in a computer program or the set of such instructions

- *code4* - attach a code to
- *code5* - convert ordinary language into code

There are two distinct noun senses for the word “computer” [3]:

- *computer1* - a machine for performing calculations automatically
- *computer2* - an expert at calculation (or at operating calculating machines)

The original Lesk algorithm compares the definition of *instruction1* with the definition of the senses *code1*, *code2*, *code3*, *code4*, *code5*, *computer1*, and *computer2* to find the overlapping words. Score here refers to the number of overlapping words. The algorithm adds up the score for each comparison to its previous score. A similar step is done for *instruction2*, *instruction3*, *instruction4*, and *instruction5*. *Instruction1*, *instruction2*, *instruction3*, and *instruction5* have score of zero. *Instruction4* has a score of six. *Instruction4* and *code1* have one overlapping word, which is “written”; *Instruction4* and *code3* have three overlapping words, which are “computer,” “science,” and “program”; *Instruction4* and *code4* have one overlapping word, which is “code”; *Instruction4* and *code5* have one overlapping word, which is “code.” All the other comparisons of *instruction4* have a score of zero. The sense *instruction4* has the largest number of overlapping words and is therefore chosen as the correct sense for the target word “instruction.” A drawback of this method is not considering the distance between the target word and the surrounding word in the given context while choosing the sense of the target word [6].

The simplified Lesk algorithm is a variation of the original Lesk algorithm [10]. The simplified Lesk algorithm disambiguates the target word by using its context. Context words are checked for their presence in the definition and examples of each sense of the target word. The simplified Lesk algorithm does not use the definitions and examples of context words. From the example above, the simplified Lesk algorithm checks for the existence of the surrounding words “code” and “computer” in the definition of *instruction1*. A similar step is done for *instruction2*, *instruction3*, *instruction4*, and *instruction5*. *Instruction1*, *instruction2*, *instruction3*, and *instruction5* have score of zero. *Instruction4* has a score of two since both the surrounding words “code” and “computer” appear in its definition. Similar to the original Lesk, simplified Lesk chooses *instruction4* as the correct sense.

Wilks et al. [11] indicated that the *glosses* of words, the definition and examples of a sense, are not sufficient enough to have a more specific distinction between word senses. They proposed the concept of expanding the gloss vectors. The vectors were built using the co-occurrence of words with each other. They used Longman's Dictionary of Contemporary English (LDOCE), which consists of 2,200 words of controlled vocabulary prepared by linguists. The expanding of gloss vectors increases the probability of finding overlapping words between word senses.

2.3.2.2 Concept Hierarchy-based Algorithms

Algorithms described in this section make use of the machine-readable dictionaries by laying out the content of the dictionary in the form of semantic networks for calculating relatedness measure.

In 1967, M. Ross Quillian introduced the use of the contents of machine readable dictionaries to determine the relation between the senses [7, 8]. His approach depicts the contents of a dictionary in the form of semantic networks. Content words are the words that define a sense in the dictionary. Their network is composed of two different types of nodes: a type node and a token node. A sense is represented by a unique node (type node). A content word is represented by a token node. Type nodes are linked to token nodes, based on their dictionary definition. The token nodes are then linked to their type nodes (senses) and those type nodes are again linked to the token nodes (content word) that occur in their definitions and so on. There is no direct link between two nodes of same type. This process is carried out for all the senses in consideration. This process forms a network of words, which can be used to determine the similarity between the definitions of two words. The number of common words found defines the relatedness between the senses [8]. Veronis and Ide [12] proposed a neural network approach that represents the senses of words in the form of semantic networks. Their network structure is similar to the Quillian's approach. Spreading activation is used to perform disambiguation. The words in the network, which are present in the context of the target word, are assigned senses of those type nodes that are located in the heavily activated part of the network.

The availability of the structural layout of the words in WordNet has encouraged various work that exploited the dictionary layout for WSD. Michael Sussna

[13] introduced a measure that assigns a sense to the noun with the help of the surrounding words by using the semantic relatedness. An edge can represent one of the following relationships: is-a, has-part, is-a-part-of, and antonyms between nouns. The edges are given weight based upon the relation. Since more general concepts in WordNet are at the top of the hierarchy and most specific concepts are at the bottom, the relatedness assigns different weights to edges that appear at various locations in the hierarchy.

Agirre and Rigan [14] introduced the notion of conceptual density in the relatedness measure. This measure is only applicable to nouns. This measure uses the is-a relationship of nouns of WordNet. The conceptual density is defined as the amount of space taken by an ambiguous noun in a particular context. The space here refers to the area (number of occurrences of a word) occupied by each of the context words inside the hierarchy of the target word sense over the total area occupied by the hierarchy of that sense. The sense with the maximum conceptual density is assigned to the target noun.

Banerjee and Pederson [15] introduced an adapted version of the original Lesk algorithm where they enriched the glosses by using relations from the semantic network. Instead of comparing just the glosses of the target word and the surrounding word, they also use the synset tree structure. In addition, they also introduced a new scoring scheme where consecutive words in two glosses have more weight than a single word match.

2.3.2.3 Heuristic-based Algorithms

Cowie [21] proposed a simulated annealing (SA) approach that uses Longman's Dictionary of Contemporary English (LDOCE) as the external knowledge source. This approach works on each sentence individually in a text. The relatedness measure in this approach depends upon the existence of the surrounding words in a particular gloss of a sense. Initially, a configuration of senses is prepared for a particular sequence of words. Then the configuration is changed by selecting a random word and its random sense. The probability of keeping the configuration of senses fixed depends upon the total score of all the senses relative to the previous configuration score.

Gelbukh [22] proposed a genetic algorithm (GA) approach that uses an original

Lesk algorithm as the similarity measure. It enriches the glosses by using the synonym relation between the words. It also takes into account the linear distance between two words in the context window period when disambiguating each other. This approach stresses the fact that a sense of a particular word is determined by the surrounding words in its context and aims to maximize the notion of coherence.

Schwab [23] proposed the use of Ant Colony Optimization to solve WSD, which we reference in this thesis as ACA-2011. This approach uses a variant of the original Lesk algorithm where glosses are expanded using the relations in WordNet. Each word is represented as a component in a vector. The algorithm starts with a graph. The graph is built based on the text structure where each word is linked to its predecessor sentence, and each sentence is linked to its predecessor paragraph. ACA-2011 performed better than the original Lesk algorithm (brute force approach) using a limited amount of time [18].

Schwab [24] proposed another use of Ant Colony Optimization, which we refer to in this thesis as ACA-2012. ACA-2012 was an adaptation of ACA-2011 with the inclusion of majority voting approach that performed better than ACA-2011 in terms of efficiency and correctness.

Nguyen [1] proposed another use of Ant Colony Optimization algorithm, which we refer to as TSP-ACO. This approach endeavors to solve the WSD problem by relating it to the Travelling Salesman Problem (TSP). It uses the original Lesk algorithm in a vector model space where each gloss is extended by the relations in WordNet and the glosses from the extended WordNet and Wikipedia. This approach also differs from other unsupervised approach by using a combination of knowledge sources instead of a single knowledge source.

All the other heuristic methods mostly use original Lesk algorithm which gives less importance to the context words while determining sense of the target word. In this thesis, we put more emphasis on the context and the part of speech of the words while determining their correct senses. We make use of simplified Lesk algorithm as this relatedness measure gives more importance to context of the target words than any other relatedness measure. We also extend the gloss, while using the simplified Lesk algorithm, by adding additional relation words from WordNet. The extension of glosses gives higher probability of context words when used in simplified Lesk algorithm. Finally, the main difference between our approach and

other heuristic approaches for WSD is that ours is constructive in nature. Previous heuristic approaches initially start with randomly generated solution and improve on it, while our algorithm disambiguates one word at a time.

Chapter 3

Bee Colony Optimization

Fish, birds, and packs of animals stick together in swarms due to their living needs. Discrete individuals have higher chances of surviving by living in a group than living alone, as they can protect and support each other more effectively. They respond to the path and speed of their counterparts. They communicate among themselves which allows them to flourish [25]. Bees exhibit similar kind of behaviour. There are a few vital components of a bee colony such as the hive, bees, and food sources [27]. At first, we explain the behavior of bees in nature. Then, we explain the computational Bee Colony Optimization (BCO) approach.

3.1 Bees in nature

Initially bees pick a hive that is close to most of the food sources. A hive is the home of the bees. It has a storage where bees deposit their nectar. The vertical dance floor, where all the bees communicate about the food sources and their quality, is an integral part of the hive. Generally the quality of the hive depends upon its location with respect to the food sources in the vicinity. A hive generates a certain number of bees to search for food sources. The number of bees depends on its quality. These bees are called scout bees. Scout bees go in different directions in the surroundings for food. When they find a food source, they check the quality and quantity and bring a sample back to the hive. The quality of the food source depends on how much nectar (information) can be extracted from it, the distance from the hive, and the bees' ability to harvest it. They can inspect one or more

food sources at a time.

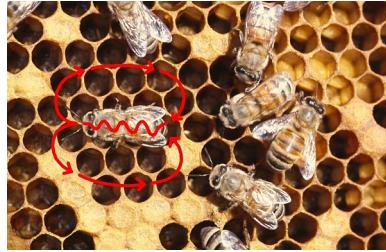


Figure 3.1: A recruiter bee performing the waggle dance

The scout bees that bring the information about the best food sources in the region are called recruiter bees [27, 28]. They get the privilege to advertise their food sources and the paths they have taken to reach that food sources. The scout bees that were not chosen as the recruiter bees check for their loyalty to their own path based upon the requirements and characteristics of the hive. If they are loyal, they follow their own path in hopes of getting a better food source in their path or else they follow one of the recruiter bees' paths and abandon their paths [29]. The bees who abandon their paths are called uncommitted bees. The communication between recruiter bees and uncommitted bees is done by performing a waggle dance. Figure 3.1 shows the waggle dance [35]. This waggle dance takes place on the dance floor of the hive. The duration and direction of the waggle dance demonstrate the distance and quality of the food source advertised by the recruiter [27]. Waggle dances happens inside the hive where there is no access of light and on the vertical dance floor. On the vertical dance floor, gravity act as a reference. Since there is no light inside the hive, the direction against the gravity depict the direction of the sun. The angle between the food source and recruiter bee as observed from the hive is replicated on the vertical dance floor [36]. It is to be noted that vertical dance floor is the inside surface of the hive. If the recruiter bee is dancing in the direction of the sun, that depicts availability of the good food source in that direction. A similar depiction is done by dancing on the left or right of the sun. The uncommitted bees have a chance to see the number of dances and pick one of the recruiter bees to follow. Figure 3.2 shows the connection between the waggle dance and the food source location. The entrance of the hive is always facing the sun. In the next section we are going to describe BCO algorithm.

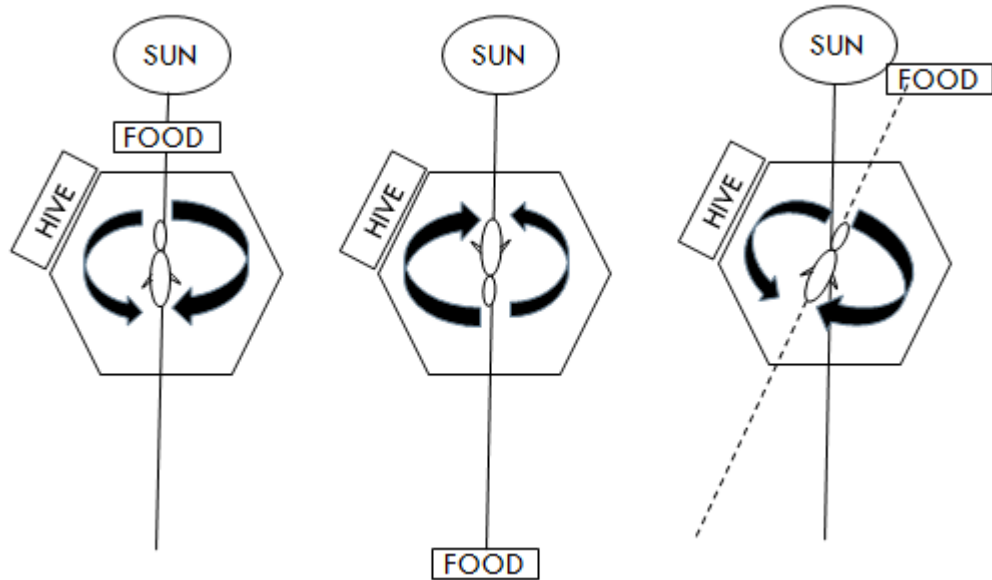


Figure 3.2: Waggle Dance Overview

3.2 BCO Algorithm

The subfield of artificial intelligence that is based on the research of the behavior of common individuals in different decentralized systems is known as swarm intelligence [25]. Insects in nature have a very subtle way of adjusting to the environment. They make use of their large population to build a decentralized system. They fetch, share, and make collective decisions for developing their colonies. There are two vital properties of swarm intelligence: self organization and division of work. These properties help in solving complex distributed problems. Self organization can be characterized in terms of positive information, negative information, fluctuation, and multiple interactions. Delegation and the division of work are based on the skills of the insect at the local level. This adaptation to the search space, based on low-level communications helps in building a global solution [26]. Bee Colony Optimization (BCO) is a meta-heuristic method introduced by Teodorovic [25]. BCO is derived from the notion of joint intelligence between the bees in nature. It is a bottom-up approach in which multiple bees are created based on the problem, and the swarm intelligence of the bees is used to constructively solve the problem. The bees collect and share information to segregate a good food source from a bad one so their colonies can flourish.

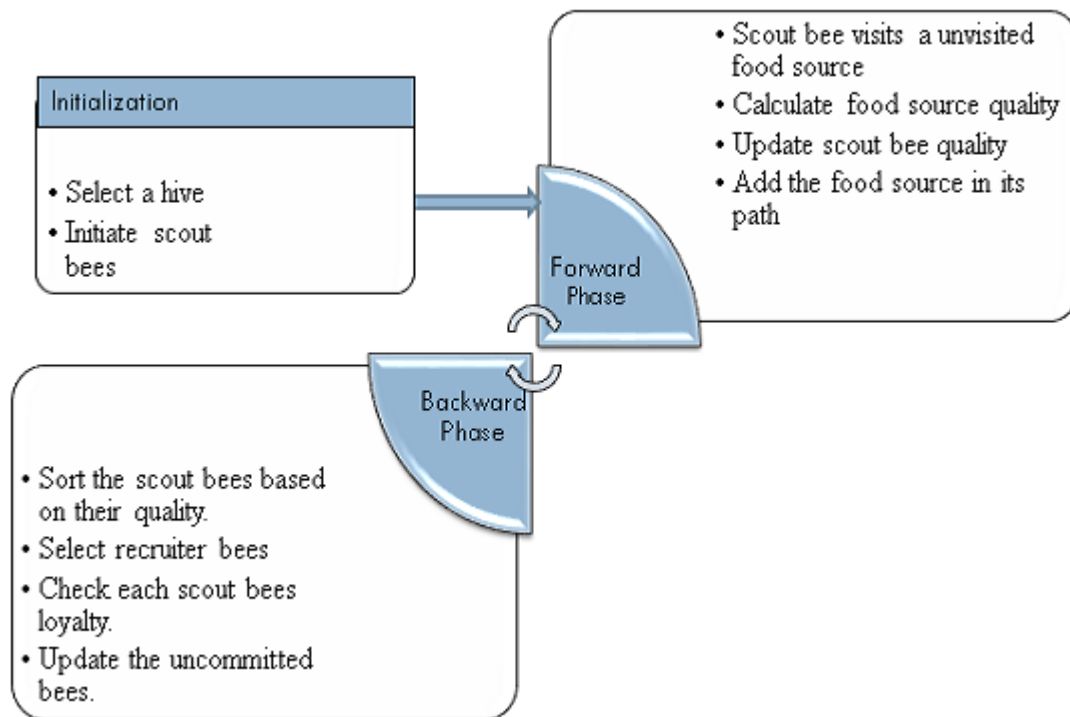


Figure 3.3: Forward/backward phase

In computational BCO, the location of a food sources depicts a feasible partial solution to the optimization problem. The number of scout bees depicts the number of solutions in the search space that we are constructing simultaneously. The quantity of the nectar represents the quality (fitness value) of the corresponding scout bee's path (sequence of partial solution traversed by the bee). At the first step, the BCO starts with scout bees. In the forward phase, each scout bee visits an unvisited food source, determines the quality (fitness value) of the food source, and adds the food source to its path (scout bee's path). The number of sources to be searched by a particular scout bee is fixed. The next step is to sort the scout bees based on their quality. The backward phase consists of picking the recruiter bees using a selection function. Selection of the recruiter bees can be one of the many approaches, including use of a roulette wheel, tournament selection, or elitism. Now for every scout bee initially in the hive, the probability of the bee being loyal to its path is calculated, based on the fitness value. If the scout bee is loyal it continues with its own path. On the other hand, if it is not loyal

it picks up the path of one of the recruiter bees. The process described above is iterated by the bees in two steps: a forward and a backward phase alternating one after another. The process is repeated until there is no improvement in the solution or a predefined maximum value of bees' path is attained. Figure 3.3 shows the alternating forward and backward phase. The BCO algorithm is shown in Algorithm 1.

In this chapter we have explained the general behavior of the bees and the BCO algorithm. In the next chapter, we explain the BCO algorithm with respect to the Word Sense Disambiguation problem.

```
Input: Food sources  
initialization()  
while stopping condition do  
    forwardPhase()  
    sortBees()  
    backwardPhase()
```

Algorithm 1: BCO

Chapter 4

Word Sense Disambiguation - Bee Colony Optimization

In this chapter, we describe Word Sense Disambiguation (WSD) as an optimization problem. We also define our Bee Colony Optimization (BCO) algorithm for WSD, which we call WSD-BCO.

The process of maximizing the relatedness between context words and the senses of the target word is defined as an optimization problem as shown in equation 4.1. Given a context C that is composed of a sequence of words $\{w_1, w_2, w_3, \dots, w_n\}$ of context length n , we assume a target word to be w_t , where $1 \leq t \leq n$. Let s_{ti} be m possible senses of w_t , where $1 \leq i \leq m$. The goal is to establish a sense s_{ti} that has the maximum relatedness measure between the sense s_{ti} and all the other words (i.e., context words of the text). We extend the sense s_{ti} to an extended sense es_{ti} by adding to it other relevant information that we describe below. We use the simplified Lesk algorithm [10] as the relatedness measure (rel) between a context word and an extended sense es_{ti} .

$$\text{WSD}_{\text{OPT}} = \text{argmax}_{i=1}^m \left\{ \sum_{j=1, j \neq t}^n \text{rel}(w_j, es_{ti}) \right\} \quad (4.1)$$

The equation shows that the optimization function (WSD_{OPT}) computes a score for each extended sense es_{ti} of the target word w_t . The output of the optimization function is the score of the extended sense index of the target word w_t that is best related to the context words.

The simplified Lesk algorithm, as described in the Chapter 2, determines the sense of a target word with respect to its context. The words in the definition and examples of a sense are called a gloss. The sense with the most appropriate meaning tends to have the maximum number of context words in its gloss. Here, the glosses of the senses are further extended with additional information based on the part of speech (POS) of the target word:

- When the POS of the target word is a noun - We include the glosses and examples for: 1) the adjectives that are pertainyms of that particular noun sense, and 2) all the directly related hypernyms of the noun sense; these terms represent the more general meaning of the noun sense.
- When the POS of the target word is a verb - We include the glosses and examples for: 1) the verb entailments that are implied by a particular verb sense, and 2) all the directly related hypernyms of the verb sense which represent the more general meaning of the verb sense.
- When the POS of the target word is an adjective - We include the glosses and examples for the nouns for which the pertainym is that particular adjective sense.

We refer to the above extension of a sense s_{wi} to es_{wi} as an *extended sense*.

Input: Sequence of words $\{w_1, w_2, w_3, \dots, w_n\}$ and target word w_t

Output: Highest extended sense score index

for each extended sense es_{ti} of target word w_t **do**

```

    for each context word  $w_j$  do
        /* Continue to next context word if  $t == j$  */
        set senseScores [ $i$ ]  $\leftarrow$  senseScores [ $i$ ] + rel ( $w_j, es_{ti}$ )
    return max{senseScores [ $k$ ]}  $1 \leq k \leq n$ 

```

Algorithm 2: WSD_{OPT}

Our optimization algorithm (WSD_{OPT}) is shown in Algorithm 2. For every extended sense es_{ti} of the target word w_t , the algorithm computes the relatedness measure between a context word and an extended sense es_{ti} and adds to its previous score for every context word. Then it returns the highest score of all the

extended senses. The extended sense index for which the highest score is returned is considered as the most appropriate extended sense of the target word w_t .

In the WSD-BCO algorithm, one word among all the words to be disambiguated is randomly selected as the hive. The number of senses of this selected word, which is now the hive, determines the number of scout bees. The path of a bee represents the words that were disambiguated. Initially this path contains the hive word as its first word. When a bee visits a new target word to disambiguate, it determines the extended sense of the new target word with the help of the words in the path it traversed. The bee uses the WSD_{OPT} defined earlier to determine the extended sense of that target word. This target word, with its disambiguated sense, is added to the bee’s path. The bee returns back to the hive and the forward/backward phase repeats again. While each bee moves to and fro between the hive and the target words, its builds an appropriate sequence of context words as path. This helps in disambiguating the next target word visited by the bee with a wider context (bee’s path).

The WSD-BCO algorithm has three main phases: initialization, forward, and backward. During initialization, we determine the frequency of the target words in the gloss, example, hypernyms, entailments, and attributes of all the target words that is supplied as input to WSD-BCO. In the forward phase, the bee travels to different target words picking the most appropriate extended sense of the target word based upon the path it traversed. In the backward phase, the path quality of all the bees is calculated and then used to decide the effective path to be explored. The forward and backward phases are alternated until all target words are assigned an extended sense. The main activities of WSD-BCO and their interactions are shown in Figure 4.1. A one-to-one correspondence between the general BCO algorithm and our WSD-BCO is shown in Table 4.1.

BCO	WSD-BCO
Food sources	All the target words except hive
Hive	One target word that is randomly selected
Bee	Senses of target word selected as hive
Bee’s quality	WSD_{OPT}
Bee’s path	Sequence of target words

Table 4.1: BCO vs WSD-BCO

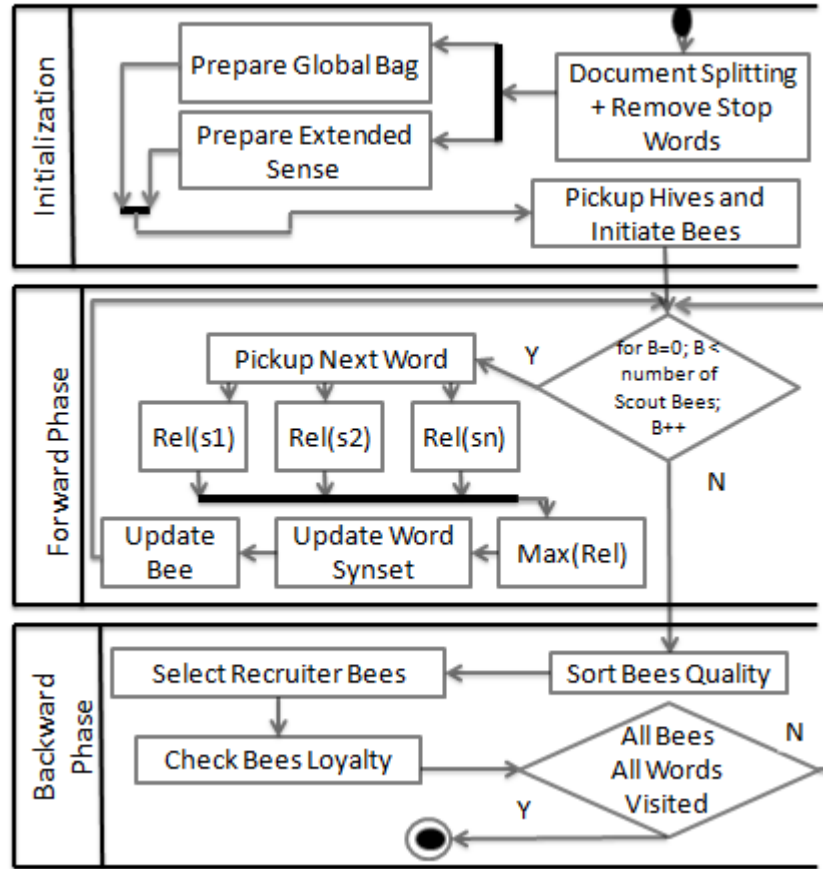


Figure 4.1: WSD-BCO Overview

In order to demonstrate how the algorithm behaves and to explain the different phases in Figure 4.1 we use the following sentence: “Every model of computer would be likely to need different instructions to do the same task.” as a running example.

4.1 Initialization

The first step in the initialization phase is the document splitting process; we split the document based on a single sentence. As the context plays an important role in deciding the meaning, we disambiguate the target words by working on one sentence at a time in order to have a stronger claim when picking the sense of the target word. The second step in the initialization process is to remove the

stop words from the input sentence. The sentence now contains only words that needs to be disambiguated. These target words can have multiple senses for the same parts of speech and can vary for different parts of speech. If the sense of a particular target word for the tagged parts of speech does not exist, then we retrieve the cluster sense of that particular target word.

We are provided with each target word’s lemma and POS. We prepare a global bag of words using all of the senses of the target words. This process is repeated for each and every target word. The next step is to count the frequency of all the target word lemmas in the global bag. Figure 4.2 shows an example where the senses for the target word “computer” are fetched from WordNet. First, we prepare a global bag of words using all the sense of the word “computer.” The global bag is filled with words from the gloss, hypernyms, entailments, and attributes of the particular senses. We then calculate how many times “computer” appears in the global bag and save this value in its frequency feature. This frequency value will be helpful when a bee in the forward phase picks a word. We can imagine that the frequency denotes the closeness of the target word to the hive. The higher the frequency, the closer the target word is to the hive.

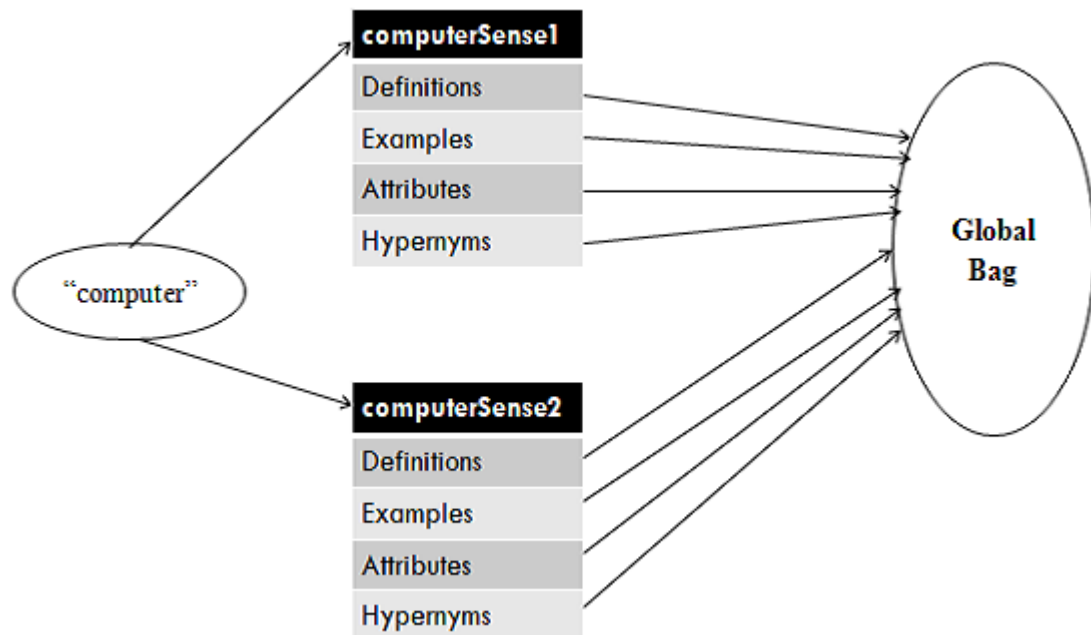


Figure 4.2: Prepare Word Frequency

The next step is to prepare the extended information for all senses of the target words. All the stop words are removed from glosses, hypernyms, entailments, and attributes of the particular sense while preparing the extended sense. The extension of a sense is prepared initially because fetching information from the WordNet database is a costly process and we can avoid repeated transactions for the same sense. Figure 4.3 shows an example where the extended information is prepared for the word “computer.” The extended senses of “computer” are `computerExtendedSense1` and `computerExtendedSense2`, with hypernyms and attributes in addition to their definitions and examples.

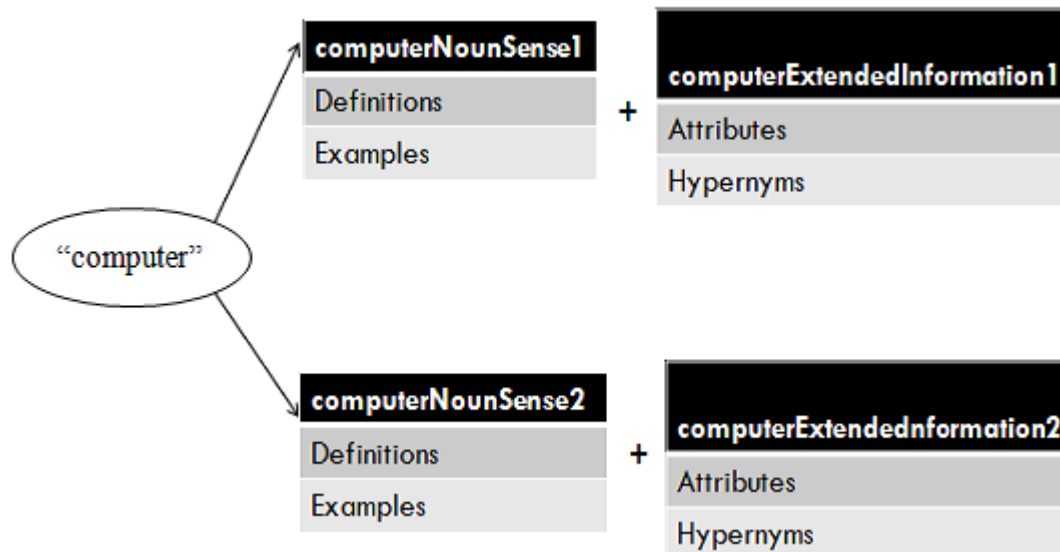


Figure 4.3: Prepare Extended Sense

The last step in the initialization phase is to randomly select a target word from the input set of target words to act as the hive. We retrieve the senses of the hive based on its POS. Scout bees corresponding to each sense of the hive are generated. Each bee has its own properties such as the bee’s path and its path quality. Figure 4.4 shows the initial status of the hive. Let us assume that “model” was picked up as the hive. There are three senses for “model.” BEE1 contains `modelSense1`, path, and path quality. The path quality is initialized to zero. The starting point of each bee’s path is the hive. Similar settings are done for BEE2 and BEE3.

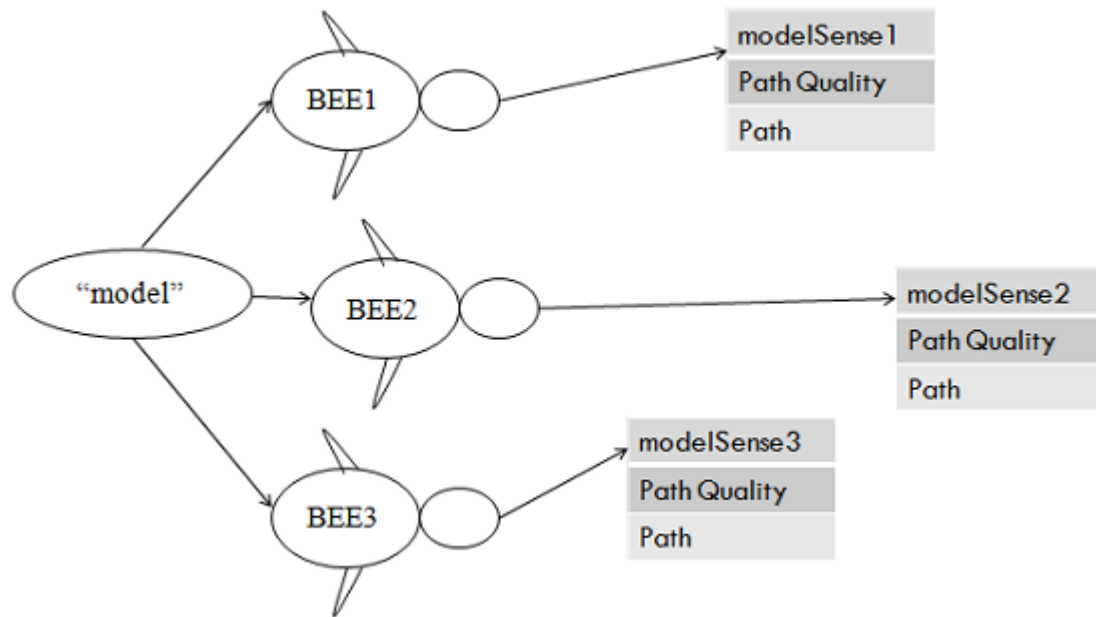


Figure 4.4: Initiate Bees

4.2 Forward Phase

The first step for a bee in the forward phase is to choose a new word to add to its path. Each bee prepares a roulette wheel based on the frequency of the target words calculated in the initialization phase for all the unvisited target words. If the target word has a frequency of 0, then we add one to its frequency so that this word has a chance, even if it is a slight one, to be picked. Each bee picks a target word from the roulette wheel to visit randomly, as shown in Figure 4.5. BEE1 picks the word “computer” to visit. After picking the word, the bee uses the optimization function in Equation 4.1 to assign an appropriate extended sense to the picked target word, in this case “computer.” The bee adds the new disambiguated word “computer” to its path. Using a roulette wheel gives a greater probability to the target words with high-frequency value.

The words in the bee’s path act as the context words and are used in the disambiguated process. The scout bee checks the words in its path for their existence in the extended senses of the target word. The extended sense with the maximum overlap of words with the path words is assigned as the correct sense to the target word. The process can be seen in Figure 4.6, where we check for the existence

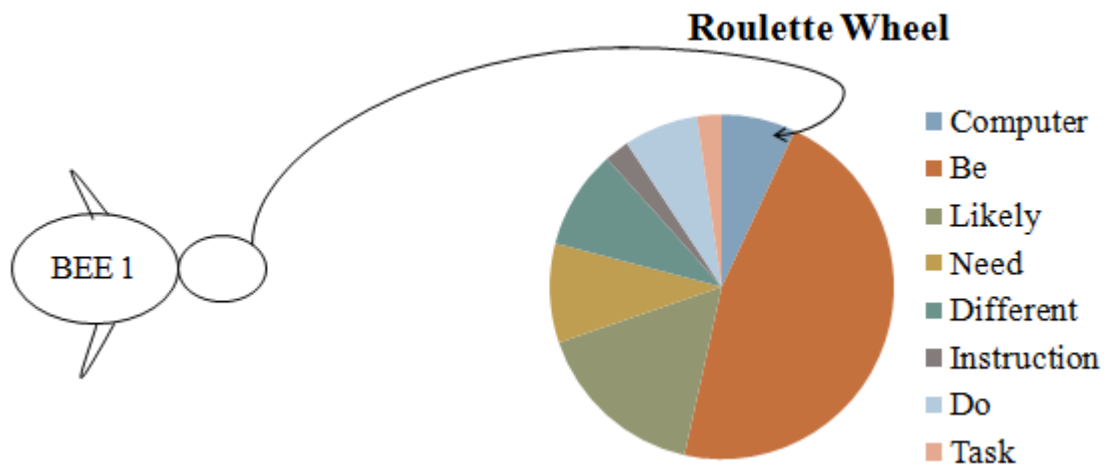


Figure 4.5: Pickup Roulette Wheel

of the lemma of the path words “model” and “instructions” in the two extended senses of the target word “computer.” The extended sense with the maximum score is assigned to “computer.” If the scores of both extended senses are equal or both are zero, then we pick the most frequent extended sense in that case. The most frequent sense of a target word is maintained by WordNet. After the extended sense for the target word is determined, the next step is to add this target word to the bee’s path. The quality of the path is also updated with the chosen extended sense score to reflect the quality of the bee’s path.

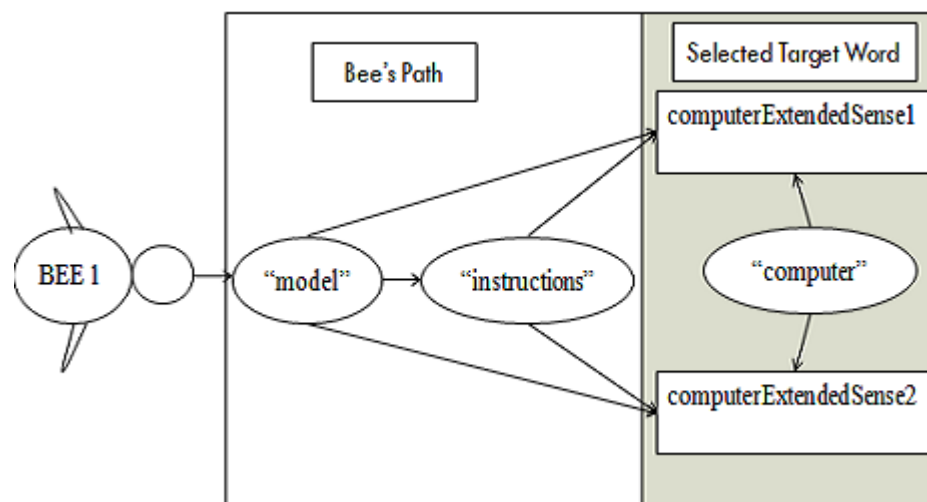


Figure 4.6: WSD-BCO Relatedness

Each bee that is initiated from the hive goes in a different direction by picking up different words in the vicinity, where the words with higher frequencies are closer to the hive. Each bee, upon reaching a target word, finds the correct sense of that target word based on the path it has pursued. The overview of the forward phase can be seen in Figure 4.7. In this example the word “model” represents the hive, while each bee represents one of the three senses of “model.” There are eight words to disambiguate (excluding the hive word). Each of the three bees selects the next word to disambiguate using the roulette wheel. In this case, BEE1, BEE2, and BEE3 select “computer,” “task,” and “need,” respectively, and disambiguate them. The circular ring denotes the single-forward phase. The words on the ring denote the target words to be disambiguated in that particular phase. Each bee eventually visits all the target words in the search space.

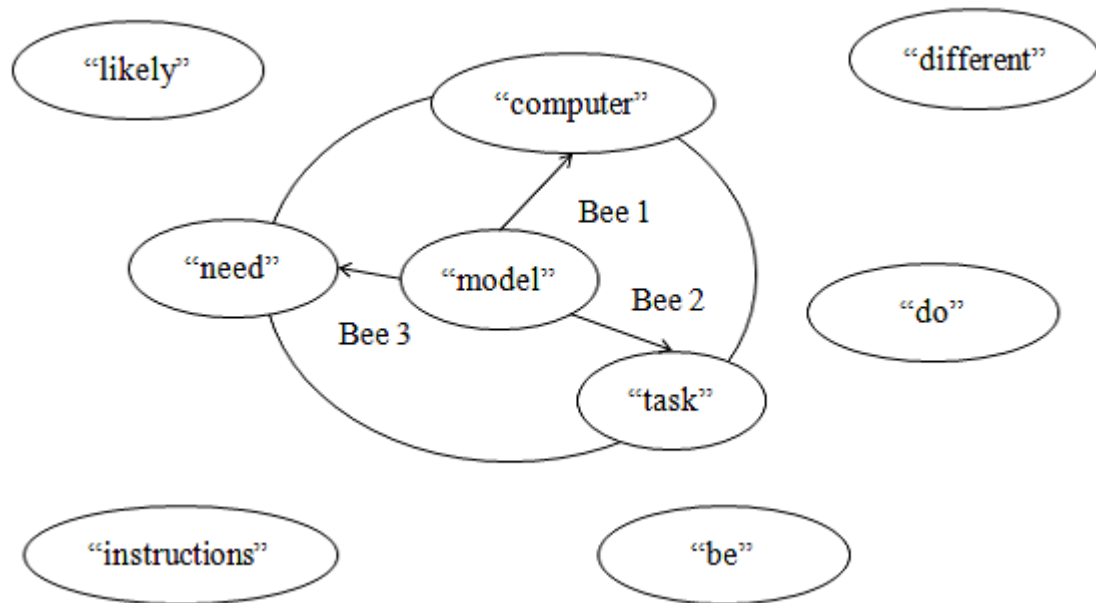


Figure 4.7: Forward Phase Overview

4.3 Backward Phase

In this phase we sort the bees based on their path qualities, where the best bee is placed at the beginning. Then we determine whether the bees will:

- become recruiters

- abandon their path and follow one of the recruiter’s path
- be loyal to their path.

We define the Recruitment Size (Rec) based on the number of bees in the hive. For example, for a hive of seven bees, we define Rec as three as we want to explore combination of target words which gives maximum WSD_{OPT} . We pick the bees with the highest path quality as recruiters. Each bee then decides whether to stay loyal to its path or not. Equation 4.2 is used to calculate the loyalty probability of the b^{th} bee.

$$\text{ployalty}_b^{u+1} = e^{-\left(\frac{o_{max}-o_b}{x}\right)} \quad (4.2)$$

$$o_b = \frac{qual_b - qual_{min}}{qual_{max} - qual_{min}}, x = \sqrt[4]{u}$$

In this equation u denotes the number of iterations of the forward phase, x defines the fourth root of u , $qual_{max}$ defines the maximum overall value of the partial solution among all bees, a partial solution here refers to the target words disambiguated in previous phases, $qual_{min}$ defines the minimum overall value of the partial solution, $qual_b$ defines the path quality of the b^{th} bee with the partial solution, o_b is the normalized value of the partial solution generated by the b^{th} bee, and o_{max} is the maximum overall normalized value of the partial solution generated among all bees [31]. As the number of forward phases increases, the probability for a bee to change its path decreases. We have modified the equation for the loyalty probability that was proposed by the authors of the BCO algorithm to fit the problem of word sense disambiguation [30]. We introduce x , which is the fourth root of u , while calculating the loyalty probability since we are disambiguating a single target word in one forward phase and the number of target words can grow as much as five hundred. Higher values of u result in very few bees changing their path; therefore, we make the value of u smaller.

$$\text{precruitment}_b = \frac{o_b}{\sum_{k=1}^{Rec} o_k} \quad (4.3)$$

The next step is for uncommitted bees to pick which of the recruiters bees to follow. Equation 4.3 is used for calculating the recruitment probability of the b^{th} uncommitted bee [25], where o_b is the normalized value of the partial solution generated by the b^{th} scout bee, and o_k denotes the normalized value of the k^{th} recruiter bee partial solution. Using a roulette wheel, the uncommitted bees can pick any one of the recruiter bees to follow as shown in the Figure 4.8. Recruitment probability can be related to the waggle dance of the bees as described in the previous chapter.

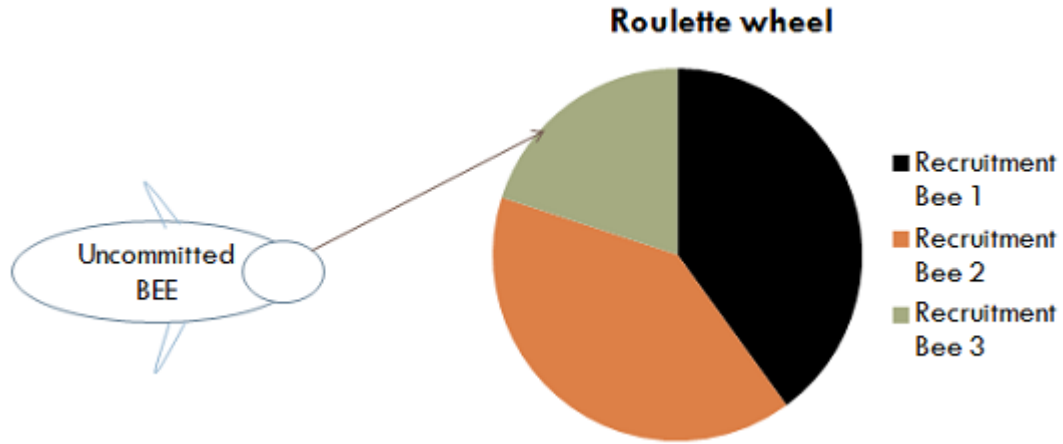


Figure 4.8: Recruitment Roulette Wheel

The overview of the forward/backward phase can be seen in Figure 4.9. BEE1 decided to stay loyal to its path and moved to the next target word “likely.” BEE2 decided to stay loyal to its path and moved to the next target word “be.” BEE3 decided to abandon its path (Bee3PreviousPath) and changes its path to that of BEE1. It then moved to the next target word “different.”

In this chapter, we have described our basic WSD problem in relation to BCO. We presented the extension of the simplified Lesk algorithm and the way it fits into BCO. We described WSD-BCO using a sentence as an example. In Chapter 5, we increase the context in consideration, describe the document splitting section in detail, and explain variations made to our algorithm.

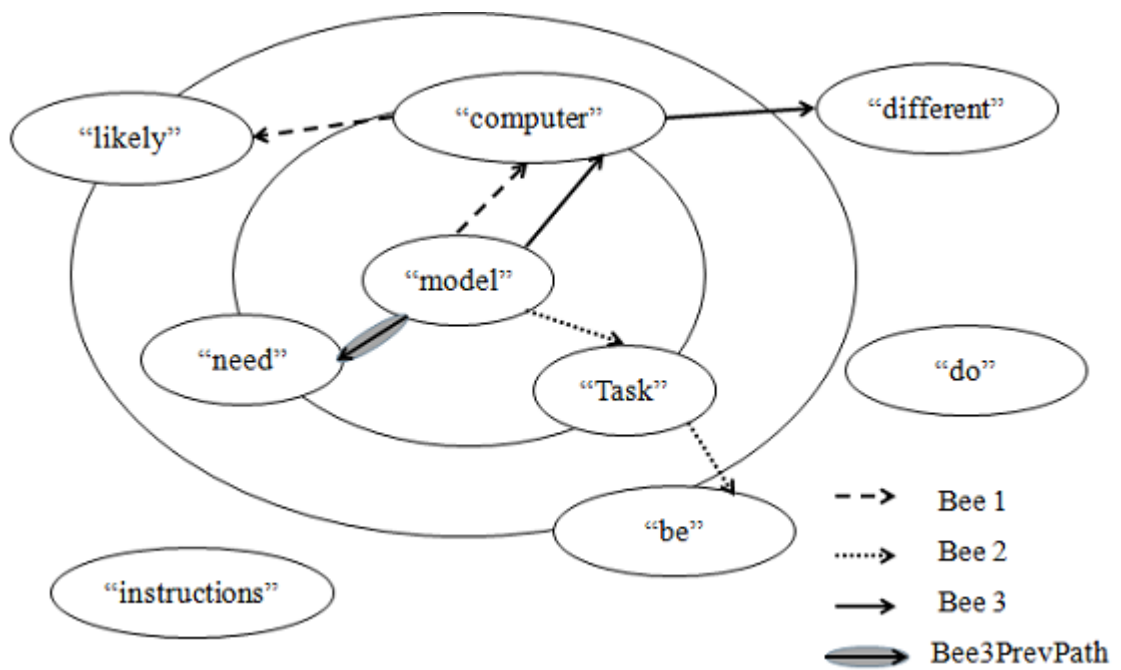


Figure 4.9: Backward Phase Overview

Chapter 5

Modifications

In this chapter we describe the modification done on WSD-BCO. We introduce four modifications where the main difference lies in the initialization phase (Figure 4.1), mainly in how we split a document. In the document splitting section of Figure 4.1 we segregate the input data between the Document level (D) and the sentence level (S). The Document level represents all the words that are related to one single document whereas the sentence level represents the words at each sentence in a particular document. As we go along we provide short notations to the approaches where they differ on the basis of how input and parameters were supplied to WSD-BCO.

5.1 Document Level

In this algorithm we disambiguate the words by working on one document at a time. For each document, we split the target words into three groups:

- all the target words that are nouns or adjectives
- all the target words that are verbs
- all the target words that are adverbs.

The document splitting process is shown in Figure 5.1. The next step is to run the WSD-BCO algorithm on all the groups separately. Extended senses with scores less than a certain threshold are not considered, and the most frequent sense is used instead. We call this algorithm BCO-D-NA/VB/J-TS. The D represents all the

target words at the document level, NA represents the noun-adjectives group, VB represents the verb group, J represents the adverb group, and TS indicates that a threshold score is being used.

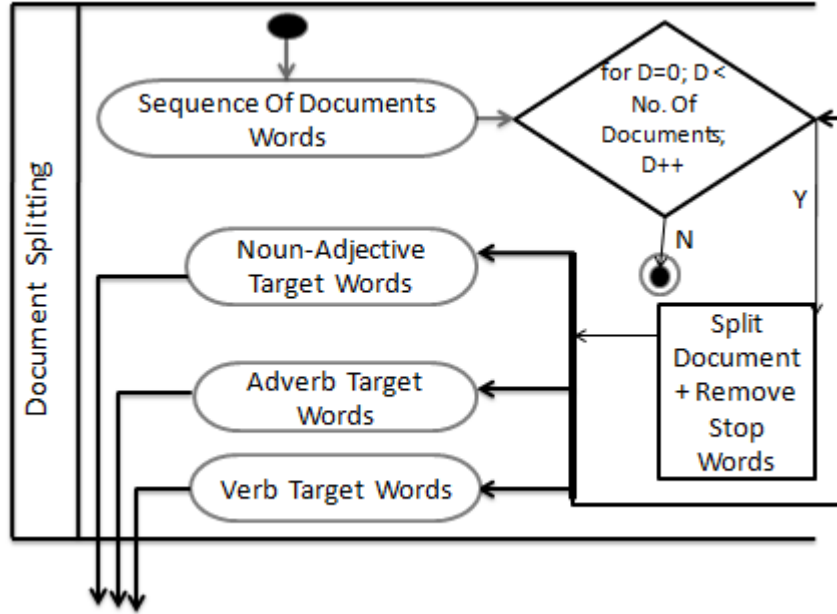


Figure 5.1: The modified document splitting step from the initialization phase of Figure 4.1 for BCO-D-NA/VB/J-TS

5.2 Sentence Level

In this algorithm the test data is processed by breaking each document into groups of sentences (GS). Initially when we try to disambiguate one sentence at a time very few senses yield a score other than zero due to the limited context that is provided by a single sentence. Therefore, we increase the number of sentences in consideration. Next, we split the target words into two subgroups:

- all the words that are nouns or adjectives
- all the words that are verbs and adverbs.

We run the WSD-BCO algorithm on both subgroups separately. Extended senses with scores less than a certain threshold are not considered. We choose to iterate over a single instance more than once as on each iteration WSD-BCO

algorithm starts with a new hive, and the more random combination of the bee's path will result in more senses yielding an acceptable score. The next iteration runs on the result of the previous iteration. The correct sense of a target word is replaced in the next iteration only if the score of the next iteration is higher than the previous iteration. We call this algorithm BCO-GS-NA/VJ-TS-I, where GS represents a group of sentences, NA represents the noun-adjectives subgroup, VB represents the verb subgroup, J represents the adverb subgroup, I represents the use of multiple iterations, and TS indicates the threshold score being used. This version of the document splitting is shown in the Figure 5.2.

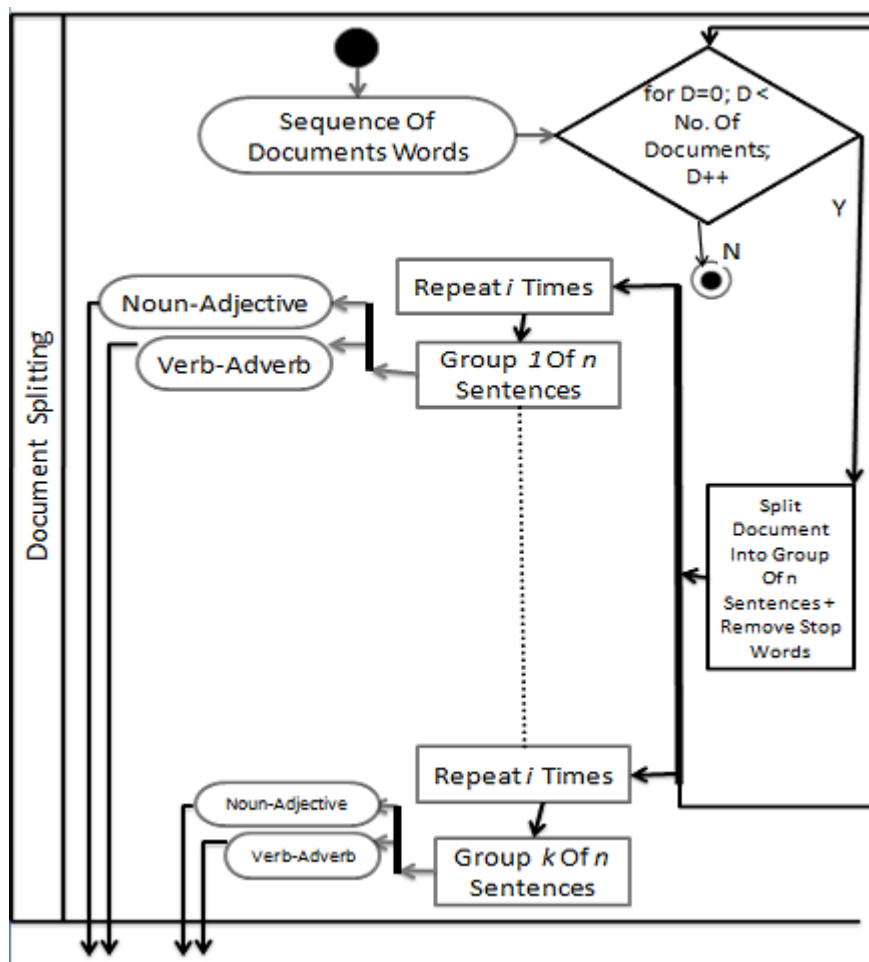


Figure 5.2: The modified document splitting step from the initialization phase of Figure 4.1 for BCO-GS-NA/VJ-TS-I

5.3 Voting

The previous approach, BCO–D–NA/VB/J–TS, does not always provide the same results when run multiple times. The reason for this inconsistency in the results is because we use a roulette wheel when bees pick up words and when scout bees choose recruiter bees to follow. This randomness in the algorithm assigns different senses on every run. Therefore, we apply the voting concept to the previous approach in order to improve the disambiguation process. Voting approach reduces the inconsistency in the results to a small range. Voting results have a stronger claim as the extended sense were chosen after multiple runs. We run the algorithm on separate instances of the input for a fixed number of times. Then we perform voting on the results to determine the extended senses with the highest agreement. We call this algorithm VBCO–D–NA/VB/J–TS. V at the beginning of an approach name represents that voting is applied to that approach. Figure 5.3 depicts the entire VBCO–D–NA/VB/J–TS algorithm with voting.

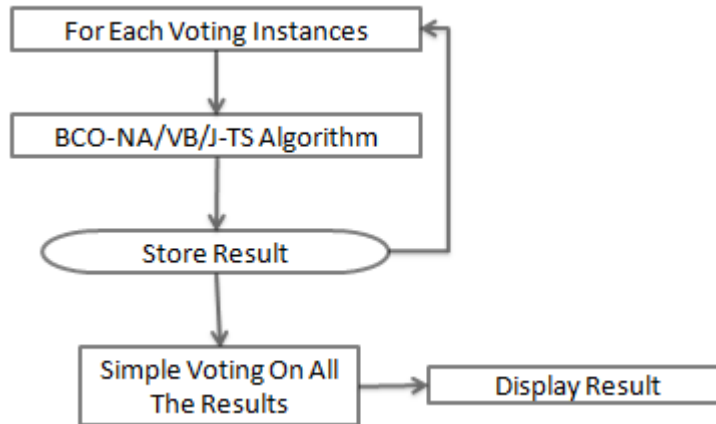


Figure 5.3: Voting algorithm for BCO–D–NA/VB/J–TS

5.4 Hybrid

In this algorithm we disambiguate the words by working on one document at a time. For each document, we split the target words into two groups:

- all the target words that are nouns or adjectives

- all the target words that are verbs or adverb

We run our algorithm only on the first group, which consists of all the nouns and adjectives. Extended senses with scores less than a certain threshold are not considered. When the relatedness measure is less than the threshold score then we use the most frequent sense of the target word. The verbs and adverbs are automatically set to their most frequent sense. We perform voting on the results to determine the extended senses with the highest agreement. We call this algorithm VBCO-D-NA-TS-HY. The D here stands for all the target words in different documents that are disambiguated separately, NA represents the noun-adjectives group, HY represents usage of the Most Frequent Sense heuristic and TS indicates the threshold score being used. This process is explained in the Figure 5.4.

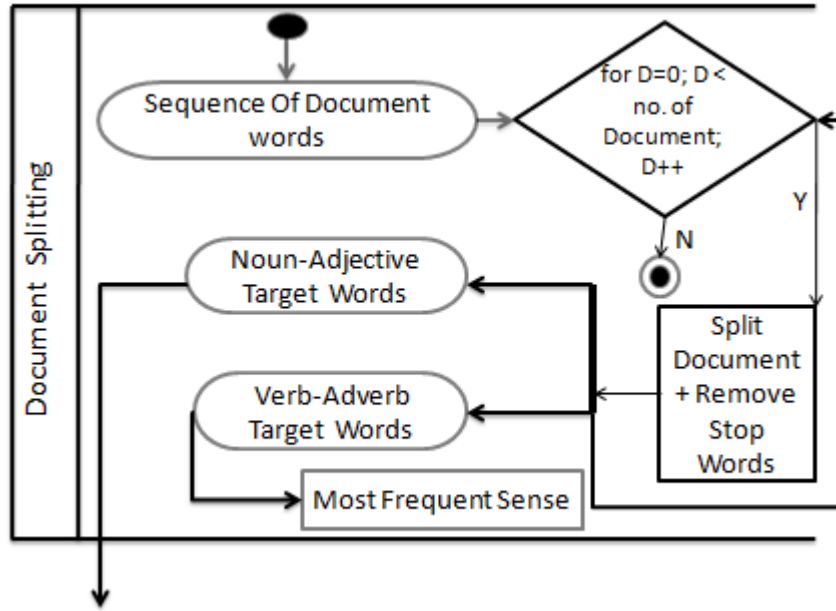


Figure 5.4: The modified document splitting step from the initialization phase of Figure 4.1 for VBCO-D-NA-TS-HY

In this chapter we have explained the modification done on the WSD-BCO algorithm based on the context. In the next chapter, we explain the test data set and results on WordNet 2.1.

Chapter 6

Experimental Results

In this chapter we report the results of our WSD-BCO algorithm and its variations on WordNet 2.1.

6.1 Data Set

We use SemEval-2007 Task 07: Coarse-Grained English All-Words Task as our test data. This data set is comprised of five documents related to fields such as journalism, book review, travel, computer science, and biography. The first three documents were obtained from the Wall Street Journal corpus [32]. The computer science document is obtained from Wikipedia, and the biography document is an extract from Amy Steedman’s *Knights of the Art*, a biography of Italian painters. Documents three and four consist of 51.87 percent of the total target words [4].

In this Semantic Evaluation (SemEval) task, linguists annotated the sense of each of the target words by picking the most appropriate sense from the coarse grained sense inventory of WordNet. Each of the five documents is an annotated set of target words. The target word is tagged with its respective part of speech (noun, verb, adverb or adjective). The lemma of the target word is also tagged along with original content. Table 6.1 gives an idea of the number of target words in each document.

WordNet is initially a fine-grained dictionary. Since word sense disambiguation is very difficult even for humans, there was a need to cluster the WordNet senses into coarse-grained ones. The coarse-grained sense inventory was created by auto-

Article	Domain	TargetWords
d001	Journalism	368
d002	Book Review	379
d003	Travel	500
d004	Computer Science	677
d005	Biography	345
Total		2269

Table 6.1: Target Words in Each Document

matically mapping the senses from WordNet to that of the Oxford Dictionary of English (ODE), which contains more general senses of the words. The difference between coarse-grained and fine-grained inventories can be seen below. Let us consider the following example for the word “pen” under fine and coarse grained senses [34]:

Fine-grained senses for “pen”

- pen1: pen (a writing implement with a point from which ink flows)
- pen2: pen (an enclosure for confining livestock)
- pen3: playpen, pen (a portable enclosure in which babies be left to play)
- pen4: penitentiary, pen (a correctional institution for those convicted of major crimes)
- pen5: pen (female swan)

Coarse-grained senses for “pen”

- pen1: pen (a writing implement with a point from which ink flows)
- pen2: pen (an enclosure: this contains the fine senses for livestock and babies)
- pen3: penitentiary, pen (a correctional institution for those convicted of major crimes)
- pen4: pen (female swan)

It can be clearly seen that in the coarse grained representation the pen2 sense does not distinguish between “an enclosure for confining livestock” and “a portable enclosure in which babies be left to play.” Therefore, the task of WSD is simpler at a coarse-grained sense inventory as compared to a fine-grained sense inventory.

6.2 Evaluating The System

In order to evaluate the effectiveness of our approach to compare it with other unsupervised WSD methods we use different evaluation measures such as precision, recall, and an F-measure of the results.

The number of target words we attempt to disambiguate can be different from the number of total target words to disambiguate as some words might not be present in WordNet. Therefore, the words which are not present in WordNet are not disambiguated. Precision, as shown in Equation 6.1, is defined as the percentage of the number of target words that are correctly disambiguated in the number of target words we attempt to disambiguate. The better the Precision, the better the system. Recall, as shown in Equation 6.2, is defined as the percentage of the number of target words that the WSD system correctly disambiguates to the number of total target words to disambiguate. The better the Recall, the better the system. F-Measure, as shown in Equation 6.3, is the harmonic mean of the precision and recall. An F-measure is used to calculate the effectiveness of the system [5]. The better the F-measure, the better the system. We used the sense inventory and scorer provided by the SemEval-2007 Task 07 organizers [4].

$$Precision = \frac{NumberOfTargetWordsCorrectlyDisambiguated}{NumberOfTargetWordsAttemptedToDisambiguate} * 100 \quad (6.1)$$

$$Recall = \frac{NumberOfTargetWordsCorrectlyDisambiguated}{TotalNumberOfTargetWordsToDisambiguate} * 100 \quad (6.2)$$

$$F\text{-measure} = \frac{2 * precision * recall}{precision + recall} \quad (6.3)$$

6.2.1 Baseline

Our baseline for the WSD problem is the Most Frequent Sense (MFS) algorithm. The MFS is calculated using Equation 6.4 [4].

$$BL_{MFS} = \frac{1}{|N|} \sum_{t=1}^{|N|} \psi(t, s_{ti}) \quad (6.4)$$

where N denotes the total number of target words. $\psi(t, s_{ii})$ equals to 1 when there is a match between the chosen sense and the cluster senses decided by our WSD system for the word t ; otherwise 0 is assigned. The calculation of the MFS is based on the SemCor corpus [33]. Table 6.2 shows the MFS for WordNet 2.1 calculated by us.

Document	Attempted	Precision	Recall	F-Measure
d001	100	85.598	85.598	85.598
d002	100	84.169	84.169	84.169
d003	100	77.800	77.800	77.800
d004	100	74.298	74.298	74.298
d005	100	75.652	75.652	75.652
Total	100	78.757	78.757	78.757

Table 6.2: Most Frequent Sense (WordNet 2.1)

It can be seen that MFS works very well for documents d001 and d002 as they are related to more general topics. Document d003 contains words that are used in both general and specific topics, so the results are not as good as the general topic documents. Documents d004 and d005 focus on domain specific topics, which is why the MFS achieved low results. The MFS baseline used by SemEval-2007 Task 07 [4] is 78.89 while ours is 78.76. The difference between the organizers F-Measure and our F-Measure lies in the fact that over a period of time many changes have been made on WordNet 2.1. Therefore, the F-Measures differs slightly. Hence, we use an F-Measure of 78.89 as the baseline to compare our results with other approaches.

As a whole, the MFS baseline gives good results and most unsupervised approaches give either lower results or similar to that of MFS; Although MFS has very good results, it cannot be a successful WSD system because the most frequent sense of a word depends highly on the domain in which the word is used. Let us see the limitations of MFS using the example below.

Consider the target word “brush” used as a noun. There are nine distinct noun senses in WordNet 2.1; some of them are as follows:

- Brush1 - a dense growth of bushes
- Brush2 - an implement that has hairs or bristles firmly set into a handle

- Brush3 - momentary contact
- Brush4 - a bushy tail or part of a bushy tail (especially of the fox)

Brush1 is the most frequent sense of the target word “Brush,” but if “Brush” were used in a specific domain of topics related to painting, then Brush2 should have been the correct sense of the target word “Brush.” Therefore, the most frequent sense does not always achieve good results in different domains. Hence, the difference between the performances of MFS can be observed in documents d004 and d005 as they belong to a more specialized domain [4].

6.3 Results

In this section we use WordNet 2.1 as the dictionary for our basic approach and all the other proposed modifications.

6.3.1 WSD–BCO

The context of the words played an important role in deciding the sense. Therefore, we refer to our basic approach BCO-S where S represents a single sentence context. This is done so that we will have a clear picture of context while comparing with the other approaches. The results of this algorithm are shown in Table 6.3. The final result was an F-measure of 74.614. It can be noted that the results are good for general concepts (d001, d002) but not that good for domain specific concepts (d004, d005).

Document	Attempted	F-Measure
d001	100	80.435
d002	100	79.683
d003	100	75.000
d004	100	72.969
d005	100	65.507
Total	100	74.614

Table 6.3: BCO–S (Rec=3, WordNet 2.1)

6.3.2 BCO–D–NA/VB/J–TS

The results obtained from this algorithm are shown in Table 6.4. When we extended the senses, we weren't able to determine the relation between verbs and adverbs in the WordNet, so it is better to disambiguate them separately. Therefore we disambiguate the noun-adjective group, verb group, and the adverb group separately. Since we are considering all the words, in the path of a bee, we expect the selected extended sense selected to have a higher score than the threshold score (TS) of five. The final result was an F-measure of 74.482. Document d001 performs well; None of the other documents perform well.

Document	Attempted	F-Measure
d001	100	83.152
d002	100	77.045
d003	100	72.800
d004	100	72.083
d005	100	69.565
Total	100	74.482

Table 6.4: BCO–D–NA/VB/J–TS (Rec=3, TS=5, WordNet 2.1)

6.3.3 BCO–GS–NA/VJ–TS–I

The results of this algorithm are shown in Table 6.5. We use a threshold score (TS) of five and iterate (I) five times over the initial data set to improve the performance. We consider five sentences as the context. The final result is an F-measure of 78.669. One important observation to conclude is the importance of disambiguation of target words based on their parts of speech separately. This approach shows very good results that are comparable to the baseline, but suffers from a problem of the low number of extended senses yielding score other than zero.

6.3.4 VBCO–D–NA/VB/J–TS

The results of this algorithm are shown in Table 6.6. BCO–D–NA/VB/J–TS does not always provide the same result when it is run multiple times. We ran this

Document	Attempted	F-Measure
d001	100	85.598
d002	100	84.169
d003	100	77.800
d004	100	74.446
d005	100	74.783
Total	100	78.669

Table 6.5: BCO-GS-NA/VJ-TS-I (GS=5, I=5, TS=5, Rec=3, WordNet 2.1)

algorithm 200 times on separate instances of the input. Then we performed voting on the results to determine the extended senses with the highest agreement. The final result is an F-measure of 79.066 while our previous result without voting for this approach was 74.262. We can see a two to eight percent improvement on all the documents. We also have a huge number of extended senses yielding scores. Document d004 (78.139) performed better than the most frequent sense (74.298). However document d005 performed poorly compared to the most frequent sense. Therefore we can strongly claim the results of this approach where we got good results for both general and specific topic documents.

Document	Attempted	F-Measure
d001	100	84.511
d002	100	84.169
d003	100	77.200
d004	100	78.139
d005	100	72.174
Total	100	79.066

Table 6.6: VBCO-D-NA/VB/J-TS (V=200, Rec=3, WordNet 2.1)

6.3.5 VBCO-D-NA-TS-HY

The results of this algorithm are shown in Table 6.7. It is interesting to note that we got the best results among all the other BCO modifications using this approach. Sixty five percent of the total words in all the documents were nouns and adjective;, the rest were verbs and adverbs. Setting the verbs and adverbs to their most frequent senses helped the VBCO-D-NA/VB/J-TS algorithm to

increase its efficiency significantly. We ran this algorithm 200 times on separate instances of the input. Then we performed voting on all the results to determine the extended senses with the highest agreement. The final result was an F-measure of 81.446. Looking at documents d004 and d005, we can see a four to six percent improvement to the MFS results. On the other hand, document d003 does not show improvement while there is a decrease in performance of document d002.

Document	Attempted	F-Measure
d001	100	87.500
d002	100	84.169
d003	100	77.800
d004	100	80.502
d005	100	79.130
Total	100	81.446

Table 6.7: VBCO-D-NA-TS-HY (V=200, Rec=3, WordNet 2.1)

6.4 Analysis

In this section we compare all the results with some of the WSD systems such as: Ant Colony Optimization (ACA) [1, 23, 24], Genetic Algorithm (GA) [22], Most Frequent Sense (MFS), and the Simulated Annealing (SA) [21]. The results of our algorithms and the related work are shown in Table 6.8, and are all based on WordNet 2.1. The WSD systems are arranged in order of high to low result. BCO-D-NA/VB/J-TS showed disambiguating the target words in noun-adjective group, verb group, and adverb group separately with a certain degree of threshold performed better than GA, SA and ACA-2011. BCO-S, our basic approach, performed better than GA, SA and ACA-2011. BCO-GS-NA/VJ-TS-I performed better than TSP-ACO, ACA-2012, ACA-2011, SA, and GA.

The VBCO-D-NA/VB/J-TS and VBCO-D-NA-TS-HY approaches outperformed other WSD systems. VBCO-D-NA/VB/J-TS showed the importance of voting on multiple runs to improve the efficiency. It also showed us the benefits of exploiting WSD problem using segregation of parts of speech and a certain degree of threshold in a global context. None of the previous approaches have segregated words on parts of speech. Finally, the VBCO-D-NA-TS-HY

System	Attempted	Precision	Recall	F-Measure
VBCO-D-NA-TS-HY	100	81.446	81.446	81.446
VBCO-D-NA/VB/J-TS	100	79.066	79.066	79.066
MFS	100	78.89	78.89	78.89
ACA-2012 voting	100	78.76	78.76	78.76
BCO-GS-NA/VJ-TS-I	100	78.669	78.669	78.669
TSP-ACO	99.80	78.50	78.10	78.30
ACA-2012	100	77.64	77.64	77.64
BCO-S	100	74.614	74.614	74.614
BCO-D-NA/VB/J-TS	100	74.482	74.482	74.482
ACA-2011	100	74.35	74.35	74.35
SA	100	74.23	74.23	74.23
GA	100	73.98	73.98	73.98

Table 6.8: Comparison of WSD Systems

approach produced excellent results when disambiguating noun-adjectives groups of target words with the most frequent sense assigned to verbs and adverbs. VBCO-D-NA-TS-HY performs better than all unsupervised approaches.

In this chapter we have explained the test data set. We discussed the results of our approaches and compared it against other WSD systems. In the next chapter, we discuss the results on WordNet 3.0.

Chapter 7

Wordnet 3.0

In this chapter, we use WordNet 3.0 instead of WordNet 2.1 as the dictionary for all our proposed approach. We also show a sharp contrast between the results of WordNet 2.1 and WordNet 3.0. We are unable to compare our results of WordNet 3.0 with other approaches because the WSD problem is contingent upon the knowledge source, as over a period of time some new senses have been introduced and some senses have been removed in WordNet 3.0. We also have similar arguments for all the approaches, which we report in the results of WordNet 2.1. There are four target words that are found in WordNet 2.1 but missing from WordNet 2.1; they are *in_addition*, *known_as*, *familiar_with*, and *else*. Table 7.1 shows the results of the most frequent sense for WordNet 3.0.

Document	Attempted	Precision	Recall	F-Measure
d001	99.7	85.559	85.326	85.442
d002	100	85.752	85.752	85.752
d003	100	79.200	79.200	79.200
d004	99.7	74.332	74.003	74.167
d005	99.7	76.163	75.942	76.052
Total	99.8	79.757	79.242	79.329

Table 7.1: Most Frequent Sense Result (WordNet3.0)

7.1 BCO–S

The results of this algorithm are shown in Table 7.2. Documents d001 and d004 show decrease in performance compared to that of WordNet 2.1. Documents d002 and d005 show improvement in performance compared to that of WordNet 2.1. Document d003 stayed the same. The F–Measure of 74.592 on WordNet 3.0 is worse than 74.614 of WordNet 2.1.

Document	Attempted		Precision		Recall		F-Measure	
	2.1	3.0	2.1	3.0	2.1	3.0	2.1	3.0
d001	100	99.7	80.435	79.837	80.435	79.620	80.435	79.728
d002	100	100	79.683	81.003	79.683	81.003	79.683	81.003
d003	100	100	75.000	75.000	75.000	75.000	75.000	75.000
d004	100	99.7	72.969	71.407	72.969	71.196	72.969	71.302
d005	100	99.7	65.507	68.023	65.507	67.826	65.507	67.925
Total	100	99.8	74.614	74.658	74.614	74.526	74.614	74.592

Table 7.2: BCO–S (Rec=3, WordNet 3.0)

7.2 BCO–D–NA/VB/J–TS

The results of this algorithm are shown in Table 7.3. Documents d001, d004, and d005 show worse in performance compared to that of WordNet 2.1. Documents d002 and d003 show improvement in performance compared to that of WordNet 2.1. The F–Measure of 73.974 on WordNet 3.0 is worse than 74.482 of WordNet 2.1.

Document	Attempted		Precision		Recall		F-Measure	
	2.1	3.0	2.1	3.0	2.1	3.0	2.1	3.0
d001	100	99.7	83.152	82.289	83.152	82.065	83.152	82.177
d002	100	100	77.045	77.836	77.045	77.836	77.045	77.836
d003	100	100	72.800	74.000	72.800	74.000	72.800	74.000
d004	100	99.7	72.083	71.259	72.083	71.049	72.083	71.154
d005	100	99.7	69.565	66.570	69.565	66.377	69.565	66.473
Total	100	99.8	74.482	74.040	74.482	73.909	74.482	73.974

Table 7.3: BCO–D–NA/VB/J–TS (Rec=3, TS=5, WordNet 3.0)

7.3 BCO–GS–NA/VJ–TS–I

The results of this algorithm are shown in Table 7.4. Documents d001 and d004 show lower performance compared to that of WordNet 2.1. Documents d002, d003 and d005 show better performance compared to that of WordNet 2.1. The F–Measure of 79.180 on WordNet 3.0 is better than 78.669 of WordNet 2.1.

Document	Attempted		Precision		Recall		F-Measure	
	2.1	3.0	2.1	3.0	2.1	3.0	2.1	3.0
d001	100	99.7	85.598	85.559	85.598	85.326	85.598	85.442
d002	100	100	84.169	85.752	84.169	85.752	84.169	85.752
d003	100	100	77.800	79.200	77.800	79.200	77.800	79.200
d004	100	99.7	74.446	74.370	74.446	74.151	74.446	74.260
d005	100	99.7	74.783	75.000	74.783	74.783	74.783	74.891
Total	100	99.8	78.669	79.249	78.669	79.110	78.669	79.180

Table 7.4: BCO–GS–NA/VJ–TS–I (GS=5, I=5, TS=5, Rec=3, WordNet 3.0)

7.4 VBCO–D–NA/VB/J–TS

The results of this algorithm are shown in Table 7.5. Document d004 shows a decrease in performance compared to that of WordNet 2.1. Documents d001, d002, d003, and d005 show improvement in performance compared to that of WordNet 2.1. The F–Measure of 79.532 on WordNet 3.0 is better than 79.066 of WordNet 2.1.

Document	Attempted		Precision		Recall		F-Measure	
	2.1	3.0	2.1	3.0	2.1	3.0	2.1	3.0
d001	100	99.7	84.511	84.741	84.511	84.511	84.511	84.626
d002	100	100	84.169	85.448	84.169	85.488	84.169	85.488
d003	100	100	77.200	78.400	77.200	78.400	77.200	78.400
d004	100	99.7	78.139	77.778	78.139	77.548	78.139	77.663
d005	100	99.7	72.174	72.965	72.174	72.754	72.174	72.859
Total	100	99.8	79.066	79.603	79.066	79.462	79.066	79.532

Table 7.5: VBCO–D–NA/VB/J–TS (V=200, Rec=3, WordNet 3.0)

7.5 VBCO–D–NA–TS

The results of this algorithm are shown in Table 7.6. Documents d001, and d004 show a decrease in performance compared to that of WordNet 2.1. Documents d002, d003, and d005 show improvement in performance compared to that of WordNet 2.1. The F–Measure of 81.870 on WordNet 3.0 is better than 81.446 of WordNet 2.1.

Document	Attempted		Precision		Recall		F-Measure	
	2.1	3.0	2.1	3.0	2.1	3.0	2.1	3.0
d001	100	99.7	87.500	87.193	87.500	86.957	87.500	87.075
d002	100	100	84.169	85.752	84.169	85.752	84.169	85.752
d003	100	100	77.800	79.200	77.800	79.200	77.800	79.200
d004	100	99.7	80.502	80.148	80.502	79.911	80.502	80.030
d005	100	99.7	79.130	79.651	79.130	79.420	79.130	79.536
Total	100	99.8	81.446	81.943	81.446	81.798	81.446	81.870

Table 7.6: VBCO–D–NA–TS (V=200, Rec=3, WordNet 3.0)

In this chapter we have discussed the results on WordNet 3.0 and we can clearly see some of our approaches perform better on WordNet 3.0 than on WordNet 2.1. In the next chapter, we explain the conclusion and future works.

Chapter 8

Conclusion

In this thesis we studied the problem of disambiguating words in a given text. We proposed bee colony optimization to solve this problem. Virtual bees helped in disambiguating the words based on the paths the bees traveled. Using words in the bees' path helps in relating the words to their context. We compared our ensemble of algorithms to six other approaches and one baseline. We also reported the result of WSD-BCO on the upgraded version of WordNet and compared its results with the previous version. Voting variations of our algorithm depicted the competitiveness of our algorithm with other approaches.

WSD-BCO uses a modification of the relatedness measure used by other approaches in order to give more importance to the context to the disambiguation process. It also achieved on par results using a single knowledge source compared to that of ACA-2012, which used a combination of knowledge sources. Disambiguating the words based on noun-adjective group, verb group, and adverb group on their own showed greater efficiency. It is interesting to see that the approach of splitting the data into domains and sentences has an important role in WSD. A hybrid version of our algorithm showed a steep rise in efficiency that was better than the most frequent sense algorithm and all the other unsupervised approaches.

For future work, we intend to use a combination of knowledge sources. This will allow us to exploit the relationship between verbs and adverbs based on their occurrence with each other and will help us improve the relatedness measure. In addition, we plan to investigate the use of different relatedness measures and their combinations in the optimization function. We also plan to exploit the hierarchy

relation of the senses in WordNet, which were not used by us in the current extension of the senses.

Appendix

WSD-BCO Algorithm

Input: Sequence of target words with extended senses and pick up frequency defined

Output: Sequence of senses of the target words

numberAlternatingPhases \leftarrow 0

initialization()

while *each targetWords is visited by each bee do*

 forwardPhase()

 numberAlternatingPhases ++

 sortBees()

 backwardPhase()

Algorithm 3: WSD-BCO

Input: Sequence of target words

Output: Sequence of target words with pick up frequency and their extended senses

targetWordsPickup \leftarrow prepareGlobalBag(targetWords)

targetWordsPickupExtendedSense \leftarrow

prepareExtendedSense(targetWordsPickup)

hive \leftarrow pick up any random word from targetWordsPickupExtendedSense

initiateBees(hive)

Algorithm 4: Initialization

Input: Sequence of target words

Output: Pickup frequency for each target words

globalBag \leftarrow null

```

for each targetWord in targetWordsSeq do
  targetWordSenses  $\leftarrow$  fetchSenses(targetWord,
  targetWordPartsOfSpeech)
  if targetWordSenses is null then
    targetWordSenses  $\leftarrow$  fetchSenses(targetWord, Adjective satellite
    reference)
  for each sense in targetWordSenses do
    globalBag.add(senseDefinition)
    globalBag.add(senseExample)
    if targetWordSense pos is NOUN then
      senseHypernym  $\leftarrow$  fetchSenseHypernym(targetWordSense) for
      each sense in senseHypernym do
        globalBag.add(senseDefinition)
        globalBag.add(senseExample)
      senseAttributes  $\leftarrow$  fetchSenseAttributes(targetWordSense)
      for each sense in senseAttributes do
        globalBag.add(senseDefinition)
        globalBag.add(senseExample)
    if targetWordSense pos is VERB then
      senseHypernym  $\leftarrow$  fetchSenseHypernym(targetWordSense) for
      each sense in senseHypernym do
        globalBag.add(senseDefinition)
        globalBag.add(senseExample)
      senseEntailments  $\leftarrow$  fetchSenseEntailments(targetWordSense)
      for each sense in synsetEntailments do
        globalBag.add(senseDefinition)
        globalBag.add(senseExample)
    if targetWordSense pos is ADJECTIVE then
      senseAttributes  $\leftarrow$  fetchSenseAttributes(targetWordSense)
      for each sense in senseAttributes do
        globalBag.add(senseDefinition)
        globalBag.add(senseExample)
  populatePickUpFrequency( globalBag)

```

Algorithm 5: PrepareGlobalBag

Input: Global Bag of words

Output: Pickup frequency for each target words

```
for each targetWord in targetWordsSeq do
  pickupFrequency ← 0
  if targetWord lemma in globalBag then
    pickupFrequency ++
  targetWordsPickUpFrequency ← pickupFrequency
```

Algorithm 6: PopulatePickupFrequency

Input: Sequence of target words

Output: Extended sense for each sense of the target words

```

for each targetWord in targetWordsSeq do
  targetWordSenses ← fetchSenses(targetWord,
  targetWordPartsOfSpeech)
  if targetWordSenses is null then
    | targetWordSenses ← fetchSenses(targetWord, Adjective satellite
    | reference)
  for each sense in targetWordSenses do
    extendedSense ← null
    extendedSense.add(senseDefinition)
    extendedSense.add(senseExample)
    if targetWordSense pos is NOUN then
      | senseHypernym ← fetchSenseHypernym(targetWordSense)
      | for each sense in senseHypernym do
      | | extendedSense.add(senseDefinition)
      | | extendedSense.add(senseExample)
      | senseAttributes ← fetchSenseAttributes(targetWordSense)
      | for each sense in senseAttributes do
      | | extendedSense.add(senseDefinition)
      | | extendedSense.add(senseExample)
    if targetWordSense pos is VERB then
      | senseHypernym ← fetchSenseHypernym(targetWordSense)
      | for each sense in senseHypernym do
      | | extendedSense.add(senseDefinition)
      | | extendedSense.add(senseExample)
      | senseEntailments ← fetchSenseEntailments(targetWordSense)
      | for each sense in synsetEntailments do
      | | extendedSense.add(senseDefinition)
      | | extendedSense.add(senseExample)
    if targetWordSense pos is ADJECTIVE then
      | senseAttributes ← fetchSenseAttributes(targetWordSense)
      | for each sense in senseAttributes do
      | | extendedSense.add(senseDefinition)
      | | extendedSense.add(senseExample)
    targetExtendedSense ← extendedSense

```

Algorithm 7: PrepareExtendedSense

Input: Hive

Output: Bees

bees \leftarrow null

hiveSenses \leftarrow fetchSenses(hive, hivePartsOfSpeech)

if hiveSenses *is null* **then**

└ hiveSenses \leftarrow fetchSenses(hive, *Adjective satellite reference*)

for *each* senses *in* hiveSenses **do**

└ bee \leftarrow new bee is created

└ bees. add(bee)

Algorithm 8: InitiateBees

Input: Sequence of target words with extended senses and pick up frequency defined and Bees

Output: partial Sequence of synset of some of the target words

PickedUpWord \leftarrow null

for *each* bee *in* bees **do**

└ pickedUpWord \leftarrow pickUpNextWord(targetWords, bee)

└ */* Randomly select a word from the roulette wheel prepared based on the pickup frequency of the unvisited targetWords of that bee */*

└ senseScore \leftarrow pickUpSense(pickedUpWord)

└ beeQuality \leftarrow beeQuality + senseScore

└ beePath. add(pickedUpWord)

Algorithm 9: ForwardPhase

Input: Picked up Word and Bee
Output: Score of sense with maximum score
pickedUpWordSenses \leftarrow fetchSenses(pickUpNextWord,
PickUpNextWordPartsOfSpeech)
if pickedUpWordSenses *is null* **then**
 | pickedUpWordSenses \leftarrow fetchSenses(pickUpNextWord, *Adjective*
 | *satellite reference*)
pickedUpWordSensesScores \leftarrow 0
for *each* extendedSense *in* pickedUpWordSenses **do**
 | pickedUpWordSensesScores [extendedSense] \leftarrow WSDOPT(extendedSense,
 | bee)
return max(pickedUpWordSensesScores)
Algorithm 10: PickUpSense

Input: Extended sense and Bee
Output: Extended sense score
pathLemmas \leftarrow null
score \leftarrow 0
while *each* pathWordLemma *in* beePath **do**
 | pathLemmas.add(pathWordLemma)
while *each* lemma *in* pathLemmas **do**
 | **if** lemma *exist in* extendedSense **then**
 | | score ++
return score
Algorithm 11: WSD-BCO Optimization function

Input: Bees

Output: Updated Bees

```

sortBees(bees)
/* Sort the Bees based on their path quality */
recruiterBees ← Top bees obtained after sorting
/* The number of recruitment bees depend on the original Size
   of the Bees in the Hive */
maxBeeQuality ← max(beesQuality)
minBeeQuality ← min(beesQuality)
while each bee in bees do
  beeNormalizedQuality ← (beeQuality – minBeeQuality)/(maxBeeQuality
  – minBeeQuality )
/* prepare recruitment roulette wheel based on the
   beeNormalizedQuality for the recruiterBees */
while each bee in bees do
  beesPathLoyalty ←
   $e^{-(\max(\text{beesNormalizedQuality}) - \text{beeNormalizedQuality}) / \sqrt[4]{\text{numberAlternatingPhases}}}$ 
  if not checkLoyalty(beesPathLoyalty) then
    /* Pick any one of the recruiter bees based on the
       roulette wheel and update the path and quality of the
       bee to that of the recruitment bee */

```

Algorithm 12: BackwardPhase

References

- [1] Nguyen, Kiem-Hieu and Ock, Cheol-Young. Word sense disambiguation as a traveling salesman problem. *Artificial Intelligence Review*, 40(4), pp. 405-427. Springer. 2013.
- [2] Navigli, Roberto. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, 41(2), pp. 10. ACM. 2009.
- [3] Princeton University "About WordNet." WordNet. Princeton University. 2010, <<http://wordnet.princeton.edu>>.
- [4] Navigli, Roberto and Litkowski, Kenneth C and Hargraves, Orin. Semeval-2007 task 07: Coarse-grained english all-words task. *Proceedings of the 4th International Workshop on Semantic Evaluations*, pp. 30-35. Association for Computational Linguistics, 2007.
- [5] Jurafsky, Dan and Martin, James H. Speech and language processing. pp. 587-599. Pearson. 2007.
- [6] Pedersen, Ted and Banerjee, Satanjeev and Patwardhan, Siddharth. Maximizing semantic relatedness to perform word sense disambiguation. *University of Minnesota supercomputing institute research report UMSI*, 25, pp. 2005. 2005.
- [7] Minsky, Marvin Lee and Minsky, Marvin. Semantic information processing. *MIT press Cambridge, MA*, 15, pp. 227-270. 1968.
- [8] Quillian, M Ross. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral science*, 12(5), pp. 410-415. Wiley Online Library. 1967.

- [9] Lesk, Michael. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. *Proceedings of the 5th annual international conference on Systems documentation*, pp. 24-26. ACM.1986.
- [10] Mihalcea, Rada. Knowledge-based methods for WSD. *Word Sense Disambiguation: Algorithms and Applications*, pp. 107-131. Springer. 2006.
- [11] Wilks, Yorick and Fass, Dan and Guo, Cheng-Ming and McDonald, James E and Plate, Tony and Slator, Brian M. Providing machine tractable dictionary tools. *Machine translation*, 5(2), pp. 99-154. Springer. 1990.
- [12] Veronis, Jean and Ide, Nancy M. Word sense disambiguation with very large neural networks extracted from machine readable dictionaries. *Proceedings of the 13th conference on Computational linguistics-Volume 2*, 2, pp. 389-394. Association for Computational Linguistics.1990.
- [13] Sussna, Michael. Word sense disambiguation for free-text indexing using a massive semantic network. *Proceedings of the second international conference on Information and knowledge management*, pp. 67-94. ACM. 1993.
- [14] Agirre, Eneko and Rigau, German. Word sense disambiguation using conceptual density. *Proceedings of the 16th conference on Computational linguistics-Volume 1*, 1, pp. 16-22. Association for Computational Linguistics. 1996.
- [15] Banerjee, Satanjeev and Pedersen, Ted. An adapted Lesk algorithm for word sense disambiguation using WordNet. *Computational linguistics and intelligent text processing*, pp. 136-145. Springer. 2002.
- [16] Cai, Jun Fu and Lee, Wee Sun and Teh, Yee Whye. NUS-ML: Improving word sense disambiguation using topic features. *Proceedings of the 4th International Workshop on Semantic Evaluations*, pp. 249-252. Association for Computational Linguistics, 2007.
- [17] Navigli, Roberto and Velardi, Paola. Structural semantic interconnections: a knowledge-based approach to word sense disambiguation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(7), pp. 1075-1086. IEEE, 2005.

- [18] Dandala, Bharath and Mihalcea, Rada and Bunescu, Razvan. Multilingual Word Sense Disambiguation Using Wikipedia.
- [19] Navigli, Roberto and Jurgens, David and Vannella, Daniele. Semeval-2013 task 12: Multilingual word sense disambiguation. *Second Joint Conference on Lexical and Computational Semantics (* SEM)*, 2, pp. 222–231. 2013.
- [20] Agirre, Eneko and Edmonds, Philip Glenly. Word sense disambiguation: Algorithms and applications. *Springer Science & Business Media*, 33, pp. 107-127. Springer. 2007.
- [21] Cowie, Jim and Guthrie, Joe and Guthrie, Louise. Lexical disambiguation using simulated annealing. *Proceedings of the 14th conference on Computational linguistics- Volume 1*, pp. 359-365. Association for Computational Linguistics, 1992.
- [22] Gelbukh, Alexander and Sidorov, Grigori and Han, Sang Yong. Evolutionary approach to natural language word sense disambiguation through global coherence optimization. *WSEAS Transactions on Computers*, 2(1), pp. 257-265, 2003.
- [23] Schwab, Didier and Guillaume, Nathan. A global ant colony algorithm for word sense disambiguation based on semantic relatedness. *Highlights in Practical Applications of Agents and Multiagent Systems*, pp. 257-264. Springer. 2011.
- [24] Schwab, Didier and Goulian, Jérôme and Tchechmedjiev, Andon and Blanchon, Hervé. Ant Colony Algorithm for the Unsupervised Word Sense Disambiguation of Texts: Comparison and Evaluation. *COLING*, pp. 2389-2404. 2012.
- [25] Teodorović, Dušan. Bee colony optimization (BCO). *Innovations in swarm intelligence*, pp. 39–60. Springer. 2009.
- [26] Bonabeau, Eric and Dorigo, Marco and Theraulaz, Guy. Swarm intelligence: from natural to artificial systems. , 1, Oxford university press, 1999.

- [27] Karaboga, Dervis. An idea based on honey bee swarm for numerical optimization. *Technical report-tr06, Erciyes university, engineering faculty, computer engineering department*. 2005.
- [28] Camazine, Scott and Sneyd, James. A model of collective nectar source selection by honey bees: self-organization through simple rules. *Journal of theoretical Biology*, 149(4), pp. 547-571. Elsevier. 1991.
- [29] Baykasoglu, Adil and Ozbakir, Lale and Tapkan, Pinar. Artificial bee colony algorithm and its application to generalized assignment problem. *Swarm Intelligence: Focus on Ant and particle swarm optimization*, pp. 113-144. Itech Education and Publishing, Vienna, Austria. 2007.
- [30] Marković, Goran Z and Teodorović, Dušan B and Aćimović-Raspopović, Vladanka S. Routing and wavelength assignment in all-optical networks based on the bee colony optimization. *AI Communications*, 20(4), pp. 273-285. IOS Press. 2007.
- [31] Teodorovic, Dusan and Lucic, Panta and Markovic, Goran and Dell'Orco, Mauro. Bee colony optimization: principles and applications. *Neural Network Applications in Electrical Engineering, 2006. NEUREL 2006. 8th Seminar on*, pp. 151-156. IEEE. 2006.
- [32] Charniak, Eugene and Blaheta, Don and Ge, Niyu and Hall, Keith and Hale, John and Johnson, Mark. Bllip 1987-89 wsj corpus release 1. *Linguistic Data Consortium, Philadelphia*, 36. 2000.
- [33] Miller, George A and Leacock, Claudia and Tengi, Randee and Bunker, Ross T. A semantic concordance. *Proceedings of the workshop on Human Language Technology*, pp. 303-308. Association for Computational Linguistics. 2013.
- [34] Havasi, Catherine and Speer, Robert and Pustejovsky, James. Coarse Word-Sense Disambiguation Using Common Sense. *AAAI Fall Symposium: Commonsense Knowledge*, 2010.
- [35] Understanding the Dance of Honey Bee. Rajkumar R <<http://www.elixirofknowledge.com/2011/06/understanding-dance-of-honey-bee.html>>, 2011.

- [36] Esch, Harald. Foraging honey bees: How foragers determine and transmit information about feeding site locations. *Honeybee Neurobiology and Behavior*, pp. 53-64. Springer. 2012.