Natural Language Processing with Signal/Collect

Bachelor of Science Thesis

by

Marc Tobler

Institute of Informatics
University of Zurich
Binzmuehlestrasse 14
8050 Zurich, Switzerland
Prof. Dr. Abraham Bernstein

Supervisor: Philip Stutz / Coralia-Mihaela Verman
Submission Date: 21.08.2012

Table of Contents

Abstract. Traditional Natural Language Processing (NLP) focuses on individual tasks, such as Tokenizing, Part of Speech tagging (POS) or Parsing. To acquire final results one would usually combine several of these steps in a sequence, thereby creating a pipeline. In this thesis we suggest a new approach to Natural Language Processing (NLP), using parallel combination instead.
We will illustrate our proposal with a Word Sense Disambiguation (WSD) and a Part of Speech (POS) tagger. We start by implementing the PageRank algorithm for WSD and the Viterbi algorithm as a POS-tagger on Signal/Collect - a framework for parallel graph processing. Then we continue by combining the two tasks in a pipeline, using the information gathered from the Part of Speech tagger to increase the performance of WSD. We proceed with our suggestion of a non-sequential combination of the algorithms, combining them into a single algorithm that handles POS tagging and WSD in parallel.
With our thesis, we want to contribute with the following two ideas. Firstly, we want to show that graph theory provides a suitable model for solving selected NLP problems. And we want to prove that modeling such graphs in Signal/Collect is a promising approach, due to the framework's good scaling and its potential for parallelization. Secondly, we want to suggest a different methodology in solving NLP tasks. We are showing a way how to get away from isolated studies of NLP problems and pipelining to a broadened approach.
We evaluate our algorithms on the Senseval 3 data, comparing the obtained results to a similar approach introduced by Agirre and Soroa in 2009.

Keywords: Word Sense Disambiguation, WSD, Part of Speech Tagging, POS, Senseval, Graph, Signal/Collect, NLP

Abstract. Traditionelle Sprachverarbeitung konzentriert sich oftmals auf gesonderte Problemstellungen, wie Tokenisierung, Wortartzuweisung oder Parsing wenn ein Text analysiert werden soll. Um dann an verwendbare Resultate zu gelangen kettet man die einzelnen Elemente aneinander. In dieser Arbeit schlagen wir einen neuen Ansatz für die automatische Sprachverarbeitung vor, indem wir die einzelnen Arbeitsschritte parallel statt sequentiell kombinieren.

Um unseren Vorschlag zu veranschaulichen, wenden wir uns zwei unterschiedlichen Aufgabenstellungen zu: Word Sense Disambiguation und Wortartzuweisung (Part of Speech Tagging). Zu Beginn werden wir den PageRank Algorithmus für die Word Sense Disambiguation (WSD) und einen Viterbi Algorithmus für implementieren, beide auf Basis von Signal/Collect - einem Framework zur parallelen Verarbeitung von Graphen. Wir fahren fort mit der Kombination der beiden Elemente in einer sequentiellen Anordnung, in der wir uns die Resulate des Part of Speech Taggings zu Nutze machen um die Leistung unseres PageRank Algorithmus zu verbessern. Zum Schluss präsentieren wir unseren neuartigen Ansatz zu einer parallelen Ausführung der Algorithmen, indem wir die beiden Algorithmen zu einem einzigen Prozess verschmelzen, der sowohl eine Word Sense Disambiguation als auch Part of Speech Tagging durchführt.

Mit dieser Arbeit möchten wir zwei Ideen in die Thematik einbringen. Erstens, möchten wir zeigen, dass Graphen ein passendes Model für die Lösung ausgewählter Probleme der Sprachverarbeitung sind. Weiter möchten wir darlegen, dass das Signal/Collect Framework dank seiner Skalierbarkeit und seines Potenzials zur Parallelisierung von Prozessen eine vielversprechende Möglichkeit zur Umsetzung genannter Lösung bietet. Zweitens, schlagen wir eine neue Methodik zur Anwendung in der Sprachverarbeitung vor. Wir zeigen hierfür einen Weg auf, der es ermöglichst Aufgaben in einem umfassendereren Ansatz anzugehen.

Wir evaluieren unsere Resulate mithilfe des Datensatzes aus dem Senseval 3 Wettbewerb, und vergleichen unsere Resultate mit den von Agirre & Soroa 2009 präsentierten Daten.

1 Introduction

Natural Language Processing (NLP) has made enormous progress in the last decades. Computer linguists and colleagues from related scientific fields have managed to continuously improve the performance of NLP algorithms and have thereby enabled the introduction of an ever-growing of NLP applications to everyday life. Speech Recognition Software can nowadays be found in almost every premium segment car, allowing the driver to control most of the vehicles functions with his voice. The same is true for cell-phones, especially smartphones. The owner of the phone can not only control most of the phone's programs, but he can also use his voice to send text messages. In 2011, the presentation of Siri (Speech Interpretation and Recognition Interface) for Apple's iPhone 4S brought the advances of NLP technology to a bigger audience than ever before. With the growing number of practical applications, the interest in creating even better language processing software is immense.

In our thesis we want to suggest a new methodology for NLP, where emphasize the advantages of combining individual NLP subproblems in a parallel way, opposed to the prevalent sequential combination. In order to merge multiple subproblems, we need a model that works well for most of them. We believe that graph theory can provide this model.

Graphs are a long established model that can be used in a wide area of applications, ranging from mathematics over biology to sociology. In the last two decades, graph theory has been of particular importance for the development and study of the internet. The gigantic amount of data that is nowadays accessible at any time has also created a demand for software that can proccess big chunks of data in an efficient manner. Graph algorithms have often been used to address this problem. In consequence, frameworks have been programmed to facilitate the implementation of graph algorithms. Signal/Collect is such a framework and we will be using it in this thesis. We will later on explain the motivation behind chosing this framework.

We believe that Graphs are suitable to solve NLP problems because of their potential to model relations between entities. In written text, the words, sentences, sections and chapters are always somehow connected to each other, thereby forming what we would call a context. When interpreting the text, we as humans, usually take notice of this context. It is therefore understandable that we would also want to teach a computer about this context when it is performing NLP. Graphs enable us to do exactly that. They can model the relationships between the nodes (words, sentences etc.) with connecting edges between them.

In the following chapters we will address two NLP subtasks: Word Sense Disambiguation (WSD) and Part of Speech (POS) tagging. We will explain how we solved those tasks using graphs and Signal/Collect. Once we have established our solutions for each of the individual tasks, we will go ahead and combine them. Firstly, we will take a look at the sequential combination of POS and WSD, which is the standard approach in NLP. Secondly, we are going to suggest a different method, using parallel, joint running of the algorithms. Our hope is that we can thereby beat the performance of the previous methods. We will evaluate all algorithms using the

Senseval 3 dataset, that has previously been used in the eponymous NLP contest[1]. We compare our results to the algorithm introduced by Agirre and Soroa in 2009 [2]. We will begin by exactly reproducing their results in Signal/Collect to create a baseline for our system.

But before going into the details of the implementation and the evaluation of the results, we will proceed to explain the theoretical foundation of the thesis, showing whose research we base our system on. We start out with an introduction to graphs and Signal/Collect.

2 Importance of Large Graphs

As mentioned, graphs are one of the most important models in current computer science. Their ability to represent relations between objects can serve a wide area of applications in a world that gets more and more interconnected. One of the main driving forces for this development is the spread and growth of the internet. While the internet itself is a graph-like structure and thereby acts as foundation of many graph implementations (e.g. Page Rank [15]), its true importance for graph research lies in the connecting services it provides to businesses and individuals. Today's logistic systems, energy grids, public transportation, traffic controls and payment systems are all highly dependent on the internet. The same is true for the entire financial market. And in recent times the growth of social networks set grounds for an even wider research space for graph theory.

Graph theory, however, is not a new approach. The theory has been established in 1736 by Swiss mathematician Leonhard Euler in his famous paper about the "Seven Bridges of Koenigsberg". It has since then been used in many research topics ranging from biology (especially in neuroscience) over mathematics to sociology. The sheer infinite number of applications for graph algorithms demands for an efficient and fast framework to work with. Especially the importance of processing large graphs efficiently can not be underestimated. In many of the above listed modern research topics, such as social networks or financial systems, work is being done on graphs with millions or even billions of nodes. Let us take Facebook for example. According to their report for the 2nd quarter of 2012 [9], Facebook has now over 955 million monthly users. If one wants to run graph algorithms over such a large network, he will not only need a lot of hardware resources, but also a framework that enables performant execution of the algorithm. The keyword here is **parallelization**. Graphs are generally a good model for parallel processing. If one manages to build his algorithm so that the calculations can be independently done in nodes or edges, the potential for load distribution in enormous. Theoretically, one could buy a PC for each node in the network and let it process only the operations for that one single node.

A framework that tries to make use of parallelization is Signal-Collect (S/C) [18]. In the following chapter, S/C will be introduced, as it will further on be used to work on the NLP tasks.

[1] http://www.senseval.org/

3 Signal/Collect

Signal/Collect (S/C) is a programming model and framework for large graph processing developed at the University of Zurich by Stutz et al. In their 2010 paper [18], they argue that as the amount of data available has increased immensely, the need to process large scale databases is becoming very important. One such collection of databases is the so called Semanic Web [2]. The idea of the Semantic Web, which was introduced by internet pioneer Tim Berners-Lee [3], is to build a web of data, i.e. a network that connects existing data via semantic relations. Stutz et al. argue that the incredible growth of this Semantic Web graph creates a demand for frameworks that can handle graph structures of great size. Some frameworks have been used quite succesfully to handle large databases, e.g. MapReduce [7]. It is suitable for many tasks that require scaling, but it does not work just as well in the presence of iterative elements. As a consequence of this weakness, Stutz et al. proposed their Signal/Collect framework, which would allow for large scale iterative graph processing in both synchronous and asynchronous manner. Like MapReduce, Signal/Collect is designed for scaling. Its structure allows the distribution of graph vertices and edges over a cluster of hosts. The capability to run algorithms asynchronously further increases the performance, as the individual host will not have any downtime while waiting for others. This property is also one of the main advantages of S/C compared to the resembling software Google Pregel [13] - Google's response to MapReduce's weaknesses. Pregel does not allow for such an asynchronous computation. On top of that, S/C is able to handle different kinds of vertices in the same graph and assigns more importance to the edges of the network, as we will see in the following section.

3.1 The Concept of Signal/Collect

Signal/Collect is built upon the idea that many graph algorithms can be calculated locally on either vertices or edges. This idea is implemented in the form of two functions: signal(for the edges) and collect(for the vertices).

In Signal/Collect, vertices act as stateful processing units and they interact with each other only by signal, which they pass along the edges. After a vertex receives a signal it performs the collect function. In this collect function, the vertex "collects" all the information (signals) it has received recently. It then recalculates its state, using the gathered information and knowledge about its previous state. The updated state information can then be sent along edges to trigger the collect functions in adjacent vertices. In between, the signal can be modified and information can be added in the signal functions of the edges.

Let us look at a simple example to illustrate the functionality of the signal and collect methods. Assume we want to calculate the shortest path from a source vertex to every other vertex in the graph. The functions would look like this:[3]

[2] http://www.w3.org/2001/sw/

[3] Code from : http://code.google.com/p/signal-collect/source/browse/trunk/core/src/test/scala/com/signalcollect/examples/Sssp.scala

```
 def signal(sourceVertex: Location) = {
  sourceVertex.state map (_ + weight.toInt)
}

def collect(oldState: State, mostRecentSignals: Iterable[Int]): Option[Int] = {
 val currentShortestPath = oldState.getOrElse(Int.MaxValue)
  Some(mostRecentSignals.foldLeft(currentShortestPath)(math.min(_, _)))
}
```

The above Scala codes basically performs the following operations. In their signal function, edges read the state of the signal-emmitting vertex and add their own weight to that number, before sending it along. In this scenario, the current state of the emmitting vertex contains the information about the shortest currently known path from the source vertex to said vertex. The edges takes this information and adds its own weight, as the weight represents the pathcost along the edge. So what will eventually arrive at the target destination is an appropriate measure of the currently known shortest path.

The nodes collect this shortest path information from all incoming edges. Among those pathlenghts, and the information about the shortest path which is stored in the nodes' state, they select the minimum value, update their state and forward the information to the outgoing edges.

On the webpage of the Signal/Collect project you can watch a video that explains the framework using a different example.[4]

The architecture of Signal/Collect enables scaling and parallelization, as all the calculations are performed on the vertices and edges. Therefore one can just distribute the vertices onto different hosts in order to achieve load distribution. This makes it possible to work with enormous amounts of nodes, as long as there are enough hosts to assign them to.

In this chapter we have given a short introduction into Signal/Collect. We explained the motivation for the framework and we looked into a simple example. We have seen that Signal/Collect, due to its architecture, along with its scalability, its potential for parallelization and asynchronous runnings is a powerful tool for the processing of large graphs.

4 Word Sense Disambiguation

Now that we have established a basic knowledge about graphs and the Signal/Collect framework, we shift our focus onto the challenges at hand. In this chapter, we present the an introduction into Word Sense Disambiguation. Word Sense Disambiguation(WSD) is the NLP task of assigning the most fitting sense of a word to a specific occurence of said word in a context. While some words have very clear interpretation, others are highly context dependent in their meaning and therefore called ambiguous. Examples for ambiguous words are manifold, but to illustrate the idea let us look at the many meanings of the word "play"[5]:

[4] http://code.google.com/p/signal-collect/

[5] From the American Heritage Dictionary of the English Language, Fourth Edition

verb:

1. To occupy oneself in amusement, sport, or other recreation: children playing with toys.
2. a. To take part in a game: No minors are eligible to play.
   b. To participate in betting; gamble.
3. To act in jest or sport: They're not arguing in earnest, they're just playing.
4. To deal or behave carelessly or indifferently; toy. See Synonyms at flirt.

[. . .] noun:

1. a. A literary work written for performance on the stage; a drama.
   b. The performance of such a work.
2. Activity engaged in for enjoyment or recreation.
3. Fun or jesting: It was all done in play.

[. . .]

As you can see, the word "play" has many meanings stemming from diverse domains and even being of different parts of speech. Ng and Lee provided a statistical analysis of word ambiguity in their 1996 paper. They showed that for the 121 most frequently used nouns in English the average number of senses is 7.8, whereas for the 70 most frequent verbs it is even 12.0 [14]. So we would expect, that finding the "correct" meaning of such an ambigous language should be very hard to do. And still most human beings accomplish this on the fly and with ease. The reason for this probably lies in the evolution of language itself. As a mean of communication, language was never observed in an isolated manner, but instead embedded in an environent. Humans, as social beings, are usually well aware of their environment. They know about the social status of the speaker or writer, they can judge the situation in which the text was expressed and they normally also know about the domain of the text. For computers however, such tasks are traditionally hard to acquire as they lack the intrinsic image of their environment. Nonetheless, automating the disambiguation process is of high importance, as its results are basic for many problems in computational linguistics. Once a sentence is disambiguated, the computer will have some sort of knowledge about its meaning. This information may further on be improved to acquire a semantic interpretation of a sentence. This interpretation is fundamental, e.g. in speech recognition, in order to understand the intentions of a user. The information gained by WSD can be used for a wide variety of other NLP tasks as well such as text mining, parsing or information retrieval. Another important application, and one of the driving forces behind WSD research is machine translation [20]. When trying to translate text containing ambiguity one encounters a complication: Ambiguous terms in the source language do often not match an equally ambiguous term in the target language. Let us construct the following example to illustrate this concept. Assume one wants to translate multiple occurences of the word "play" from English to German:

1. John plays tennis with Anna.
2. What is your favorite Shakespeare play?
3. A brilliant play by the Denver Broncos resulted in a touchdown.

Looking at the different meanings in English, it is however impossible to match all of them to a single word in German. For example the following mappings from English to German would be appropriate for the above sentences:

1. to occupy oneself in (a sport or diversion); amuse oneself in (a game) - spielen
2. (Performing Arts / Theatre) a dramatic composition written for performance by actors on a stage, on television, etc.; drama - (Theater-)Stück
3. (General Sporting Terms) US and Canadian a move or manoeuvre in a game a brilliant play - Spielzug

So to reach a suiting translation, running word sense disambiguation in a pre-processing step is essential.

In this section, we have introduced Word Sense Disambiguation as a research topic in Natural Language Processing. We explained why disambiguation is easy for humans, whereas computers are usually struggling. And we depicted the importance of WSD as a foundation for several applications in computational linguistics. We will now give a brief history of WSD research and show the current state of the art.

4.1 A History of WSD

The task of Word Sense Disambiguation has first been described and tackled by Warren Weaver in his ground-breaking thoughts about translation in 1949:

> "First, let us think of a way in which the problem of multiple meaning can, in principle at least, be solved. If one examines the words in a book, one at a time as through an opaque mask with a hole in it one word wide, then it is obviously impossible to determine, one at a time, the meaning of the words. "Fast" may mean "rapid"; or it may mean "motionless"; and there is no way of telling which. But if one lengthens the slit in the opaque mask, until one can see not only the central word in question, but also say N words on either side, then if N is large enough one can unambiguously decide the meaning of the central word." [20]

He understood that for a disambiguation the word itself would never be sufficient, but an environment would also be mandatory. More specifically, he suggested using adjacent words as additional information.

He also noted the importance of the domain of the text in question. He realized, that knowledge about the domain of a certain text, especially in science, would facilitate selecting the correct sense, as the spectrum of possible values could be narrowed:

> "Clearly N varies with the type of writing in question. It may be zero for an article known to be about a specific mathematical subject. It may be very low for chemistry, physics, engineering etc."

As a consequence of this observation, the following decades of machine translation research (and therefore WSD research) were dedicated to the creation of so called

"micro-glossaries" i.e. specific dictionaries that would only contain word senses for a specific domain [12].

As Agirre and Soroa pointed out, some early approaches that introduced methods still applied today were suggested by Masterman (1957) and Madhu & Lytle(1965). Masterman tackled the need for a logical structure in language, as postulated by Weaver, by studying semantic networks. A modern representation of such a semantic network is WordNet, that we will introduce in detail later on. Agirre and Edmonds emphasize the importance of Madhu & Lytle's contribution, who measured sense frequencies of words, on which they applied the Bayesian Formula to get the most probable sense for a word in the context [1].

A big boost to WSD research was the introduction of large scale electronic dictionaries and corpora such as the Oxford Advanced Learners Dictionary of Current English (OALD). The sudden availabity of these resources replaced the by then present handcrafting of dictionaries and corpora with automatical knowledge extraction. This shift in methods not only facilitated the introduction of new WSD approaches, but also their comparability, which resulted in the Senseval contest starting in the 1990's. Agirre and Edmonds argue that it also showed the close connection between WSD and lexicography, which kicked off the research field of "Dictionary-based WSD".

The ground for modern WSD was set in the 1990's, with the introduction of WordNet, the application of statistical and machine learning methods to WSD and the first realization of the Senseval contest.

Today's best performing approaches in the Senseval contest are machine learning algorithms that are trained on manually sense-tagged corpora and then applied to a test setting. Only recent methods, such as Sinha & Mihalcea [16] or Agirre and Soroa [2] - whose method reproduced in this paper - show an attempt to abandon supervised methods, and work with unsupervised approaches - i.e. approaches that do not rely on manually constructed training sets.

4.2 WordNet

An important part of every word sense disambiguation is a dictionary-like structure that contains the senses that can be assigned. These dictionaries may come in various forms, ranging from micro-glossaries over ontologies to thesauri. Current state of the art algorithms, including the ones presented in the Senseval contests, are working with WordNet, a project developed at Princeton[6]. WordNet is a thesaurus that consists of so called synsets, which are sets of synonyms. Those synsets are connected to each other with semantical relations. Relations covered are among others: hypernym - hyponym[7], holonym - meronym[8] or relations based on similarity. Synsets are built for four different word types: Nouns, Verbs, Adjectives and Adverbs.

[6] http://wordnet.princeton.edu

[7] A word is a hyponym of another word, if it is part of the class described by the second word. I.e. "dog" is a hyponym of "animal". "Animal" would then be the hypernym of "dog".

[8] A word is a meronym of another word, if it is a part of the second word. I.e. "bumper" is a meronym of "car". "Car" would then be the holonym of "bumper".

Important to note here is that the part-of-speech of a synset is actually stored, which will be relevant for some of the methods used in this thesis. The entire WordNet can be represented as a graph, where synsets are formed into vertices, and the semantic relations among them build the edges. This property of WordNet makes it perfectly suitable for our approach, which is why we will use it as our target of sense mapping. For a more detailed description of WordNet, you can visit the site http://wordnet.princeton.edu/.

5 Page Rank

PageRank (PR) is an algorithm developed by Larry Page and Sergei Brin at Stanford University [15]. It is an algorithm that can be used on graph-like structures to calculate an objective measure about the importance of nodes within the network. It was specifically designed to rate the relative importance of Web Pages. This information can be used for web search and user navigation. Page & Brin used their invention to build Google, a company with a current market capitalization of more than 220 Billion Dollars.[9] The PageRank algorithm has hence been widely used in research, including applications in computational forensics.

The idea of PageRank is to use information about the link structure of the studied network to mechanically calculate the node importance. The algorithm can be described by this recursive formula:

$$PR(p_i) = \frac{1-d}{N} + d \sum_{p_j \in M(p_i)} \frac{PR(p_j)}{L(p_j)} \tag{1}$$

Where $PR(p_i)$ is the rank (or weight) of page *i*, *d* is the so-called damping factor, *N* is the total number of nodes in the network, $M(p_i)$ is the set of pages that link to *i* and $L(p_j)$ is the number of outgoing edges of *i*. The formula can be explained with three basic ideas.

First,we assume that nodes that are referred (linked) to very frequently - i.e. nodes with many incoming edges - must have some importance among the referring nodes. This idea is represented by the sum built in the formula.

Second, nodes that have a high weight themselves are a better indicator of the importance of other nodes. This means that outgoing links from such nodes will have higher influence on the weight of the target nodes than others. This concept is represented by the PageRank term $PR(p_j)$.

Third, edges that stem from nodes with a high number of outgoing links have a lower influence on the weight of the target node. This concept is represented by the term $L(p_j)$.

The traditional PageRank formula can be modified to allow the assignment of higher weight or importance to a certain group of nodes. This modified concept is often referred to as Personalized PageRank (PPR) or Topic Sensitive PageRank [11].

[9] As of 21st August 2012

To show the necessary modifications, let us rewrite the above formula in the form used by Agirre and Soroa [2]:

$$Pr = dMPr + (1 - d)\, v \tag{2}$$

We redefined the formula, using matrices and vectors. The object of interest for PPR is the vector v. Using the Equation (1), we can deduce that it should be a vector of size N, which has the constant value $\frac{1}{N}$ for every element. The idea of this vector is that when people are surfing on the internet, they do not just follow links between pages. Sometimes a user would also perform a "random jump" to any node in the network. For traditional PageRank, this random jump factor is constant for every node as we can see. Personalized PageRank ,in contrast, uses this vector to model user preferences. It assigns different random jump values for different nodes in the network. This has the effect of creating an area of higher weight around nodes with high random jump factor values. The goal of these modifications is to a create a more specialized and personalized page ranking.

6 Word Sense Disambiguation with Personalized Page Rank

In the following sections we will look into the program for Word Sense Disambiguation. We will start off with a general overview of the approach, like it was first introduced by Agirre and Soroa in 2009 [2]. We will provide an intuition why Personalized PageRank is suitable for WSD and we will show how the method works. In a second section, we will describe how we implemented PPR in Signal/Collect. Then we continue by explaining some of our design decisions. This section will also cover some problems we encountered during the reimplementation and illustrate how we solved them. Finally, we will look at some preliminary steps that had to be undertaken, and how they affected our results.

6.1 Functionality of the PPR algorithm

As mentioned before, the Personalized PageRank algorithm that we are using for our thesis has been suggested by Agirre & Soroa. However, they implemented the algorithm in C++ in a very problem specific manner. Combinining their algorithm with our POS tagger would therefore be almost impossible. This is the reason why we decided to reproduce their algorithm in Signal/Collect. We studied their algorithm in much detail and were eventually able to identically reproduce their results, given the same input. But before we describe our version, we should discuss why PPR is a suitable algorithm for WSD.

What PPR does is spread weight into a graph. And as it is the Personalized version of PageRank, it actually spreads it from well-defined roots. We can compare this spreading of weight to the spreading of rumours. Imagine a group of people who know about some rumours, we will call them rumour-keepers. Each of the members of this group knows about a different rumour. Aside from this group, there are a

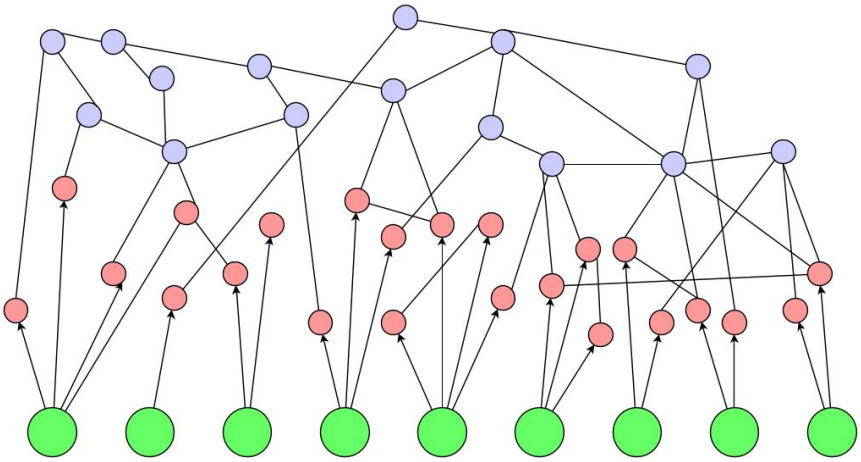
lot of other people like friends and familiy, who initially know nothing about any rumour. The rumour-keepers will then start telling their friends and families about the rumour, but always keep a part to themselves. So the friends and families now also know about the rumours, but only parts of them. In a next iteration of tattling they also start spreading the rumour as well, telling it to their friends and family. This process repeats itself over and over. In the end, every person knows something about some rumours but none of them knows the complete rumour, except for the rumour-keepers. But there is also a difference between the amount of the rumour that each person knows. Some of them might have heard it first-hand, some heard it from a friend of a friend of a friend. There is also the possibility that certain people know about more than one rumour, as they are somehow connected to more than one rumours keeper. So we ended up with an uneven distribution of rumour knowledge across the social network. Personalized Page Rank works in exactly that way. We will have now see how this can be used for Word Sense Disambiguation.

We start by defining the the group of rumour-keepers. These are replaced by the words in our context. The nodes that model these words have an increased randum jump factor and will therefore continuously acquire weight. Then we find the friends and family of the rumour-keepers. These are replaced WordNet Nodes that are directly connected to the word context. So if the word "play" would be part of the sentence, this group would contain an entry for every sense that we found in the dictionary earlier. Then we define the remaining people. These are replaced by the remaining nodes in WordNet. They are not connected to the rumour-keepers directly, but indirectly through the first group of WordNet Vertices. So now the spreading starts but instead of rumours we are spreading weight. And just as described above, after a few iterations we end up with an uneven distribution of weight. Some nodes have more weight because they are closely connected to an input word, some have more weight because they are connected to more than just one input word. We will now look upon how to use this information for WSD. Studying the first group of synsets, thats contains all the possible senses for the input words. We have previously seen two potential explanations for weight differences among nodes. This group however is a little different. First of all, they are all equally strong connected to an input word. They have a direct connection. In consequence, the only remaining explanation for weight differences is, that they are somehow indirectly connected to some other input words as well. So what we found is an implication of importance for all possible word senses of a word, because we know about adjacent words. This is exactly the connection Weaver pointed out, when he first talked about Word Sense Disambiguation.

So we have seen why and how PPR can be used for Word Sense Disambiguation. In the next section we will describe in detail how we implemented PPR in Signal/Collect.

6.2 The Implementation

Building upon the intuition we gave in the previous section, we will now explain how we implemented the defined roles in Signal/Collect. The following diagram illustrates the graph we build.



In the picture, green nodes are Input Vertices.
Pink nodes are WordNet Vertices that are directly connected to Input Vertices.
Grey nodes are the remaining WordNet Vertices.

The Input Nodes represent a word in a certain sentence or context. One of these nodes will be added for each word in a given text. The WordNet Nodes represent the synsets of WordNet. In the graph, there exist two types of edges. Edges that connect one WordNet node to another are called WordNet Edges. They are bidirectional connections between synsets. They control the flow of weights within WordNet. The edges that connect Input Nodes to their corresponding WordNet Vertices are called Input Edges. These unidirectional connections control the initial flow of weights into the WordNet, and remain inactive thereafter. The following code shows the signal function of both edge types:

```
def signal(sourceVertex: VP_WNVertex) = {
  sourceVertex.state / sourceVertex.sumOfOutWeights
}
```

In this signal function, the rank of the sourceVertex will be divided among its neighbours, who all get a part of that rank. Inside the following collect function, the PageRank formula as described in the PageRank chapter is being calculated.

```
def collect(oldState: Double, mostRecentSignals: Iterable[Double]): Double = {
  dampingFactor * mostRecentSignals.sum + (1 - dampingFactor) * randomJumpFactor
}
```

6.3 Design Decisions & Problems

While implementing PPR on Signal/Collect we stumbled upon a few differences between our initial results and the results obtained by Agirre & Soroa's algorithm (further on called UKB). In our effort to exactly reproduce their results, we often based our design decisions on their in order to have it as a baseline for our combined approaches. However, some of our initial ideas may be useful in improving the results.

The design decision however leave room for future research. We are confident that the performance of the isolated WSD with PPR could be improved. The subsequent list contains a few of the design decisions we made according to UKB and possiblities of improvement:

- The algorithm runs for exactly 30 iterations. Agirre & Soroa showed in their paper that the results do not improve significantly after this threshold anymore. This might however not be the case for our S/C adaptation
- The algorithm runs in synchronous mode. Some of our early tests showed that synchronous and asynchronous mode do not necessarily reach the same results. Studying the differences and trying to explain them might be a topic for future papers.
- The initial distribution of weight in the graph is even for our approach and for UKB. This means that even synsets that don't have any connection to the input context (sentence) start with some weight. The small amount of weight will later on be countered by the higher amounts of weight that flow into the graph from the Input Vertices. But the question arises, whether this initial even distribution of weight can create local hotspots, like in traditional PageRank, that are not strongly connected to the context. Those hotspots might then influence the end results.

In addition to these design decisions, we also ran into problems when trying to reproduce the numbers in the UKB paper. Even though we have been able to reproduce the UKB algorithm to the extent that, given and identical input context, the results would be identical as well (not only the selected senses, but also the weight distribution within the graph), we kept slightly underperforming compared to the published data. After some research and getting in contact with the authors, we were able to locate the differences at two sources:

- The UKB version used for the 2009 paper actually used a different parameter, which slightly corrupted the results.
- In a preliminary step, they created a slightly more complex input context than just the original sentence in the text. We tried to reproduce this preliminary step to the best of our knowledge (as described in the next section), but there might still be slight differences.

In this section we discussed some of our design decisions and some problems we stumbled upon. While all of the mentioned points potentially influence the overall performance of our algorithms (including the pipelining and joint running), they do not influence the relative comparability of the approaches, as they are consistent within our evaluated algorithms. We will now take a look at some preliminary steps that were necessary in order to reproduce the UKB approach.

6.4 Preliminary Steps

As we have seen in the previous sections, the PPR graph consists of two the distinct parts. One part that models the WordNet, and one that model the input context. For

the creation of the WordNet part, UKB and therefore also our algorithms rely on two types of input files. One of them is a dictionary file, which contains all words that have synsets in WordNet with their corresponding synset id. The other type of files are relation files that contain information about all the relations between synsets in WordNet. Those two filetypes are available online for a number of different WordNet versions. In our evaluation, we will focus on the most recent versions, namely versions 1.70 and 3.0.

In addition to the WordNet related files, one needs a file that contains the input context. We decided to use the syntax defined in UKB also for our input files. To create such an input context, one would start with the text that should be evaluated. In a first step, this text should be lemmatized, such that the resulting lemma can be found in WordNet. In a second step, Agirre & Soroa created contexts of at least 20 words. They would do this by starting with the sentence that should be disambiguated. Then they would add adjacent sentences to the context, until it has reached a total size of at least 20 terms. As mentioned before, we tried to reproduce this preliminary step, but our results potentially differ from UKB.

7 Part Of Speech Tagging

In the previous chapters we introduced the WSD as a first NLP problem and we illustrated how we tried to solve it. In the following sections, we will proceed with the second NLP algorith, which is Part Of Speech tagging.

Part of speech (POS) tagging in linguistics is the process of mapping a word in a text to its corresponding part of speech. Just like Word Sense Disambiguation, POS tagging is a context-sensitive task. When looking at one word isolated one cannot always tell what part of speech it has. We have previosly seen that for the word "play". It can act as a verb - as in "I play football." - or as a noun - as in "My favourite Shakespeare play is Hamlet.". This shows that POS tagging generally needs more than just a list of words and their part of speech.

Part of Speech Tagging evolved over time. What started out as a purely manual work, soon found approaches of automation. In the creation of the tagged Brown Corpus, the corpus we will use in this paper as well, Greene and Rubin invented TAGGIT [10].

TAGGIT is a rule-based method that was responsible for 77% of the tags in the 1 million word Brown Corpus, the rest of which was done manually. Rule-based methods like this build on a set of rules that specify what combinations are allowed. From this set of constraints, the best solution for a certain text is deduced.

A special case of the rule-based category of POS taggers are transformation-based taggers, such as the one proposed by Brill in 1992 [4]. Unlike usual rule-based algorithms, Brill's does not use a predefined set of rules, but instead builds them ad hoc. Ommitting the details, Brill's algorithm basically assigns its most frequent POS tag to each word (derived from a first training corpus). This initial tagging is used on a second training set, and results are compared with the correct solution. For every mistake the tagger initially made, it will try to find a so called "patch". To change tags, one of the examples the authors give is:

Change tag a to tag b when:
1. The preceding (following word is tagged z.)

Another type of POS taggers are the stochastic or probabilistic methods. They base their tagging upon information about probabilities, derived from a training set.

8 Hidden Markov Models and the Viterbi Algorithm

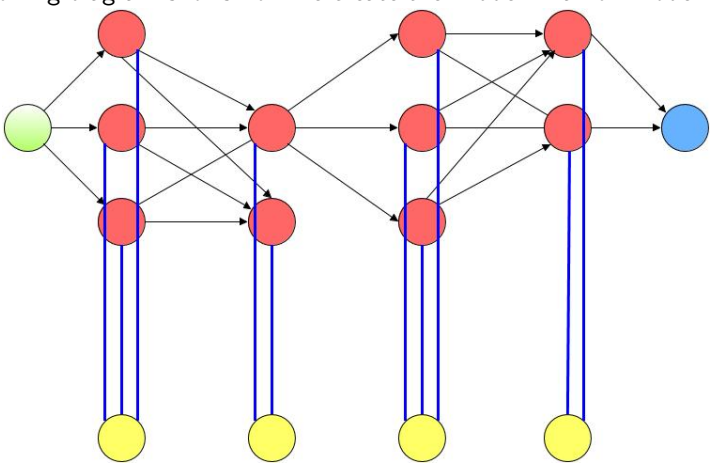
Markov models are a statistical models that use Bayesian probabilities to make assumptions about the "hidden states". Hidden state are properties that can not be observed. Therefore one tries to deduce information about the hidden states from the information about observable events and their correlation with the hidden states.

The Viterbi algorithm, invented by Andrew J. Viterbi [19], can be applied to a HMM. It then calculates the most probable sequence of states, given the observations, emission probabilites and transition probabilites between the hidden states.

Inspired by the method presented in [5] we decided to implement our Part of Speech tagger with the Viterbi Algorithm.

9 Part of Speech Tagging using the Viterbi Algorithm

In the previous chapter, the Viterbi algorithm was introduced. We explained that its purpose is to find the most probable path or sequence of hidden states in a HMM given a series of observations. In our case, Viterbi will calculate the most probable sequence of POS tags (hidden states) given a sequence of words (observations). The following diagram shows how we create the Hidden Markov Model:



The yellow nodes are Observation Vertices. They model the words from the input sentence.

The red nodes are Hidden State Vertices. They represent part of speech tags. Every node in a column is one possible assignment of a POS to the word below.

The green node is the Start Vertex, the blue one is the Evaluation Vertex.

The thin black arrows represent state transitions from one Hidden Vertex to the next one. It is important to note that these edges are directed, so the information is only transported in one direction, from the start of the sentence to the end of the sentence.

The thick blue lines represent emissions. As we have seen in the chapter about HMMs, for every Hidden Vertex there is a table that contains emission probabilites for each possible observation. In our case, the emission probabilites are defined as follows: "What is the probability of emitting the word x, given the hidden state with part of speech y."

10 Combining WSD and POS Tagging

We just introduced two individual NLP problems and possible graph-based algorithms to solve them. But our goal for this paper is to combine multiple approaches in order to increase the performance. Before we go into the details of how this combination can be built, we want to show our motivation for this thesis: how would we want to join the approaches, and why do we think our results could benefit from such a combined method.

10.1 Motivation for the Combination of Approaches

Word Sense Disambiguation and Part of Speech Tagging are well studied research topics. Many solutions have been suggested and a lot of them are much more complex than the ones presented in the previous chapters. A lot of them are also very domain specific. For example in medicine, POS taggers can reach excellent results as can be seen in Smith et al. [17] Their POS tagger reached 97% accuracy in their target domain. Smith et al. also emphasize the importance of domain specialization, as taggers for general text do not perform well when applied to the database they used. Their results, and those of others, are very promising. However, they also noted that part-of-speech taggers are only the starting point of computer comprehension of language. Only when combined with other methods, they can provide useful real-life applications such as improved access to literature (information retrieval), creating knowledge databases (information extraction) and automated reasoning (knowledge discovery). [17] This highlights the importance of an intelligent combination of NLP algorithms. The traditional way to do this is pipelining. A popular software to build such a pipeline is StanfordCoreNLP[10]. The framework integrates several NLP tools, including the part-of-speech (POS) tagger, the named entity recognizer (NER), the parser, and the coreference resolution system, and provides model files for analysis of English. These NLP tools can be combined in a sequential manner, thus forming the pipeline. Even though some elements in this pipeline could be swapped, they usually work in a specific order. Looking at the NLP tasks described in this paper, POS tagging usually serves as an input to Word Sense Disambiguation. Examples where this is the case are Dang et al. [6] or Wilks & Stephenson [21]. In this paper

[10] http://nlp.stanford.edu/software/corenlp.shtml

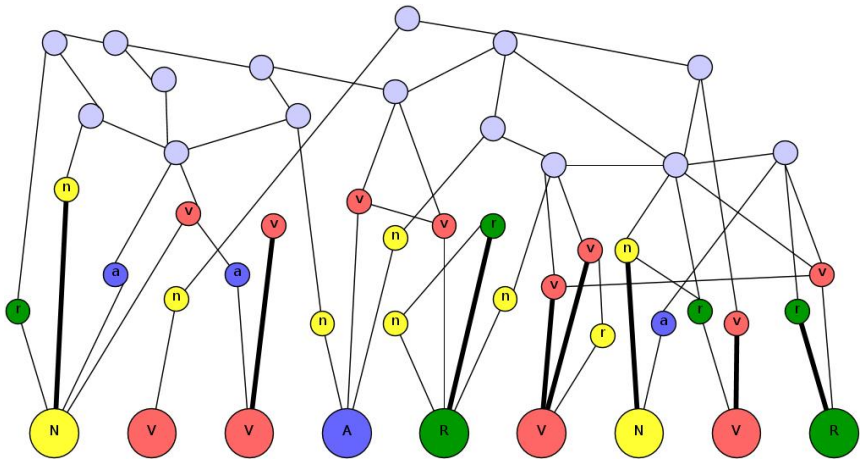
we want to suggest another approach. Instead of using a one directional pipeline, we want to use parallel execution, i.e. a joint running of multiple algorithms. We hope that we can thereby increase the overall performance compared to the pipeline. The idea of parallel running of two different algorithms is not entirely new. It was thought of before e.g in Buyko et al. [8] who implemented an ontology for NLP pipelines that would also allow for parallel integration of NLP tools. In this thesis, we want to focus on showing that joint running in the area of graph-based algorithms can be beneficial. We will therefore now go ahead to introduce the specific ways of combining WSD with POS tagging we used. First we will show how we combined the approaches in a pipeline, using POS tagging as input to WSD. The thereby created results will serve as a baseline for the joint approach. Then we will show how we made use of the WSD results in the POS tagging. And then we will show how we created a joint approach that combines the algorithms in a mutually beneficial way.

10.2 How to Combine the Approaches

In this section, we will look at how we can combine the WSD and POS tagging algorithms. In the first section we will look at the pipeline POS-WSD, in the second section we describe the pipeline WSD-POS, and in the last section we will take a look at the joint approach.

10.3 Pipeline: POS to WSD

The first approach we are going to look at is a pipeline, i.e. a sequential run of the algorithms. We start by running the Part of Speech tagger. Then we use the results gained as an additional input to the WSD. The following illustration shows how the part of speech tags will be fed into the Personalized PageRank that we used as the algorithm for WSD.



As you can see, the information is fed into the Input Nodes of the PageRank graph. The Input Vertices and their counterparts in the WordNet Graph are now colored, implying their part of speech. At this point, it is important to remember

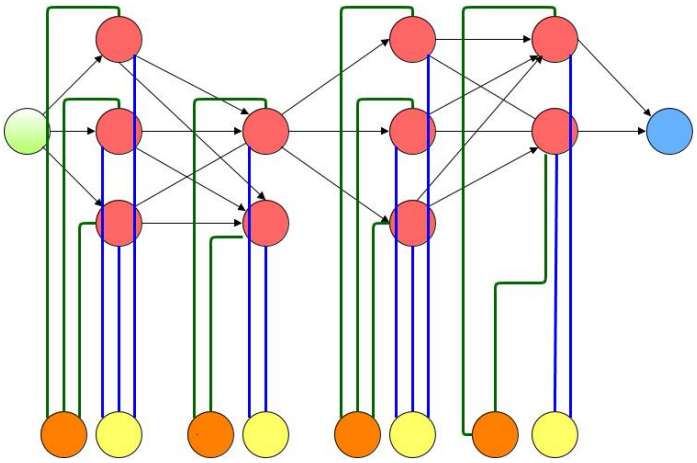
that WordNet synsets, the entities that are incorporated in WordNet nodes in our PageRank graph, also have Part of Speech tags assigned. We can use this information for comparison with the Input Vertices. The weights are now modified when passing from the context into WordNet. In relations where the POS of the Vertices fit, more weight is being transported, which is indicated by thick lines. The target WordNet Vertices will therefore recieve more weight in the PageRank algorithm. What we hope to achieve with this method are a word sense choice that generally fits better, and especially an improvement in the breaking of ties. In the isolated WSD algorithm, some WordNet nodes that belong to input nodes might actually end up with the same weight. In this case, the input nodes would not know how to decide and would randomly pick one of the top solutions. [11] We hope that by adding information about the Part of Speech, we can reduce the number of ties and thus the randomness factor of our solution.

10.4 Pipeline: WSD to POS

What we have seen in the last section is a straightforward approach on how to integrate a Part of Speech tagger and Word Sense Disambiguation. Using POS tags as additional input to WSD is the standard method how to combine the two algorithms. In order to benefit from a parallel running of the algorithm, this one directional way to look at the integration is not sufficient. To improve the overall performance of the Word Sense Disambiguation, we also need to take a look at how the Part of Speech taggers can in return profit from the results (or intermediate results) of the WSD algorithm. This is essential to create a mutually beneficial connection between the algorithms, that will in consequence lead to an iterative, incremental improvement of the results. In this section, we will therefore look at the other possible pipelining sequence - WSD to POS tagging.

The following image revisits the Viterbi model that we used for POS tagging. In addition, it shows how we can make use of the results of word sense disambiguation, namely WordNet synsets, to improve the performance of the Hidden Markov Model.

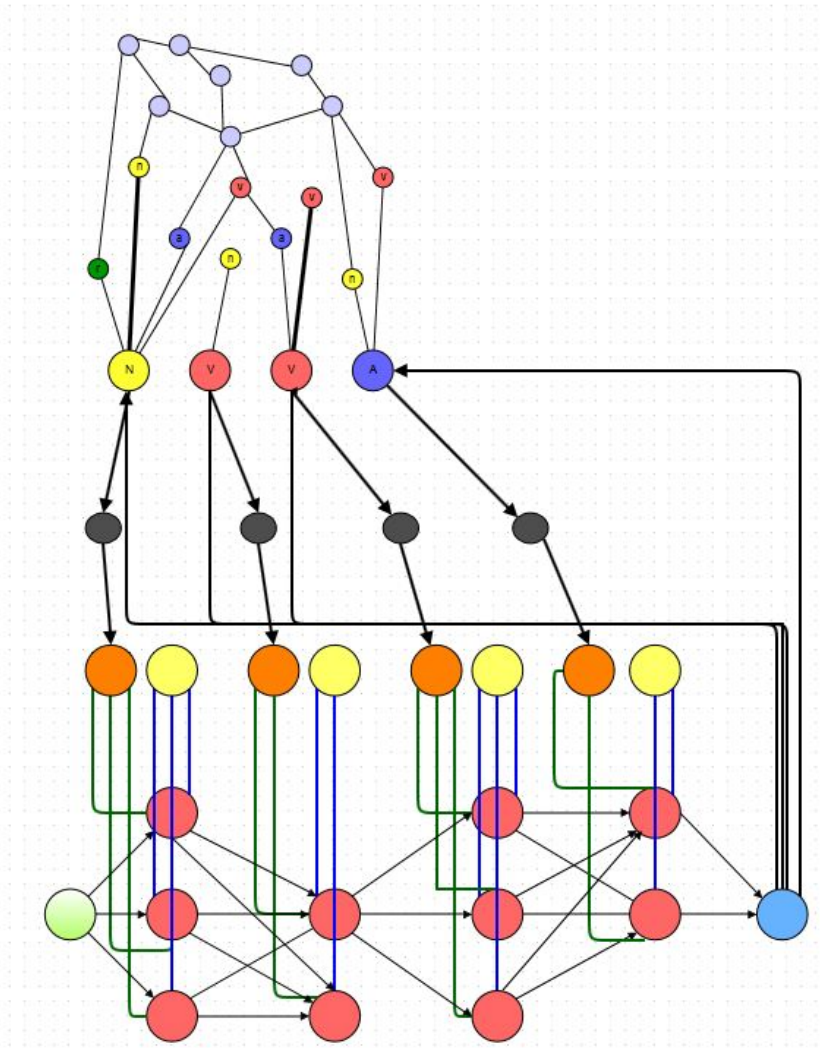
[11] Note that the Senseval evaluation algorithm would break ties also by selecting randomly among the possible solutions. We just make the selection in advance.



Once again, we need to recall that WordNet synsets also possess a POS tag. We will make use of this circumstance to integrate the WordNet POS tag as an additional observation into our Hidden Markov Model. This will be used alongside the observation of words, that we used before. The new observation vertices are the orange ones at the bottom. They have an emission connection (green line) to the same Hidden Nodes as the word they are assigned to. The two emssion signals are combined by multiplying the respective probabilites with each other. But just like for the emission probabilites of words, we need a training set to calculate the correct emission probabilities for the new nodes. That means we must calculate the probability of getting a certain part of speech as a result of the WSD process tag, given the hidden state. We can learn the probabilities by running WSD over the Brown corpus, which is already manually tagged with POS. Once the PPR algorithm has finished we know the part of speech of the resulting synsets for certain, as well as its real part of speech. We then use this information to calculate conditional probabilites for the Viterbi algorithm.

10.5 Parallel Execution

We have now seen possibilites how to integrate the solutions of one NLP task into the execution of the other. Our goal will now be to use this information to create an approach that merges the two algorithms into one process. We will use the interfaces introduced in the pipeline approaches to push information from one of the algorithms to the other. In order to do so, new types of vertices and edges are necessary to connect the interface with the nodes that contain the required information. The following illustration shows the structure of the merged graph:



The dark grey nodes handle transportation from the WSD Input Nodes to the corresponding Observation Vertices in the Viterbi part of the graph. Whenever an Input Node is sufficiently confident that a certain synset fits for him, and that synset has a different POS than the Input Vertex itself, it updates its own POS state and pushes the new information through the grey vertices into the Viterbi algorithm. The Viterbi algorithm, on the other hand, transports all its knowledge to the Evaluation Vertex as usual. The Evaluation Vertex will then forward this information to the WSD part of the graph. The Input Vertices will once again update their POS if necessary and push this information into the PPR algorithm. The graph creates an environment that allows for iterative improvement of results.

11 Experiments

In the following section we will present the results of our research. We will compare the three algorithms:

- Isolated run of PPR for WSD (pr old, pr new)
- Pipeling of Viterbi for POS to PPR for WSD (ppl 1.1,ppl 2.0,ppl 5.0)
- Combined, parallel running of both algorithms (combined)

We evaluated the algorithms using the scoring software that was provided with the Senseval contest data and freely available mappings between WordNet versions[12]. We used two different versions of WordNet, 1.70 and 3.0.

	Performance	
Algorithm	WordNet 1.70	WordNet 3.0
pr old	0.519	0.456
pr new	0.533	0.431
ppl 1.1	0.542	0.441
ppl 2.0	0.553	0.476
ppl 5.0	0.550	0.492
combined	0.479	0.356

The "pr new" algorithm makes some small changes to the original UKB algorithm, that was reproduced in "pr old". In the old version, the weights of the Input Nodes will be decreased to a lower level after the first iteration where they feed their weight into the WordNet part of the graph. In the newer version, Input Vertices keep their initial high weight and continuosly pass it into the WordNet. With this change, we were hoping to mitigate the influence of local hotspots as we explained in the section about our design decisions for PPR. Whether this change was really beneficial cannot be confirmed nor denied, as the outcome differs depending on the WordNet version used. All further algorithms build upon the new version of PPR.

The different versions of pipelining differ in the value of the POS multiplier. In the section about the Pipeline from POS to WSD, we described how we assign additional weight to edges that lead to WordNet nodes with POS that match the POS gathered from the Viterbi algorithm. The weight is calculated by multiplying the base value with the POS multiplier. We decided to try out different values for the multiplier, which are indicated in the name of the algorithm. What we can see in the experiments is that a higher multiplier seems to be beneficial, altough for WordNet 1.70, the multiplier 2.0 lead to slightly better results than 5.0. The combined approach also needs a multiplier, and we chose 2.0 over 5.0, because we wanted to balance the interaction between POS tagging and WSD, instead of giving a high value to the output of the POS tagging.

At this point we have to mention that the combined approach was only run on 120 of 304 contexts in total, due to time constraints. only run

[12] www.lsi.upc.es/ nlp/tools/download-map.php

12 Discussion

In this chapter, we want to discuss the results presented in our experiments. We will try to interpret the data gathered.

Firstly, we want to discuss the discrepance of permances of all algorithems between the different WordNet versions. We cannot fully understand how this difference exists, but it might be related to the fact that the initial Senseval task, including solutions was designed on WordNet 1.7. The scoring program that we use expects WordNet 1.71 ids as input, which requires in intermediate mapping from WordNet 3.0 to WordNet 1.71 ids. Altough we believe that the mapping is quite accuratele, we would suspect that the changes in the WordNet structurebetween the different versions, especially the relations among synsets, might lead to the observed performance loss. In any case, it has to be noted that same effect can also be found in the data published by Agirre & Soroa.

Secondly, we want to compare the performance of the pipelines with different multipliers. We can observe the trend that a higher multiplier leads to better performance. This is not further astonishing. The higher the multiplier the bigger the influence of the POS tags in the WSD algorithm. As POS tags are known to be beneficial in WSD, the measured values are well within expectation. There is however a value that conflicts with the observed trend. For WordNet 1.70, a multiplier of 5.0 actually leads to worse results than a multiplier of 2.0. This result might indicate diminishing returns of higher influence of POS tagging in the Word Sense Disambiguation. If the multiplier is set to strong the overwhelming importance of POS tags could hinder a proper execution of the WSD algorithm. At this point, we can only speculate wether this is the case, but the effect of this parameter definitely provides material for further research.

Finally, we want to take a look at the results obtained by the combined approach. Unfortunately our joint algorithm was not able to outperform the pipeline approach. Altough we were not yet able to beat the baseline with the joint approach, we believe that future research in this area could yield better results. The concept of joint running is still new and needs more exploration. Further studies could investigate the information transfer between the algorithms in more detail. We have shown that the information flow from the Part of Speech tagger into the Word Sense Disambiguation is beneficial. We are also confident that the the information flow from the WSD into the POS is generally beneficial. Therefore we think a good starting point for future research is the study of influence balancing between the algorithms. We have stated earlier, that a to strong influence from one algorithm on the other might hinder the overall performance. While we were initially hoping to create a circle of continuous perfomance improval, we might as well see the opposite. As soon as one algorithm has a negative influence on the performance of the other, we might actually create a vicious circle that lowers the performance. Therefore we believe that it would be a good idea for future research to study the balance between multiple algorithms.

13 Conclusion and Future Topics

In this section we will summarize our findings. Additionally we will bring up some topics for possible future research in the area.

In our thesis we have implemented NLP algorithms on the Signal/Collect framework. We were able to implement a Personalized Page Rank algorithm for Word Sense Disambiguation based on the methods of Agirre and Soroa and a Part of Of Speech Tagger based on a Viterbi algorithm. We went on to combine the two tasks with each other. First, we used a sequential order, where we used the results from the POS tagger to improve the performance of the WSD algorithm. This approach lead to a significant increase in performance. We then combined the algorithms in a parallel manner instead. We were hoping that we could improve the performance even more with this approach. Unfortunately, our joint approach did not succeed in outperforming the pipelining. The reasons for this might be manifold and could be investigated in future research. We also found other potential topics to study and highlighted them throughout the entire thesis. Some of the most promising ideas include asynchronous execution of the algorithms and the question how strong the influence of Part of Speech tags for the Word Sense Disambiguation process should be, in order to maximize the performance.

In our thesis we were able to show that graph-based algorithms for Natural Language Processing can reach promising performances which lets us look optimistic into the future of this research field. We believe that there is still much room for improvement.

References

1. Eneke Agirre and Edmonds. Draft of: Word sense disambiguation: Algorithms and applications. Note.
2. Eneko Agirre and Aitor Soroa. Personalizing PageRank for word sense disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, page 3341, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics.
3. Lassila Berners-Lee, Hendler. The semantic web. *Scientific American*, 2001.
4. Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the third conference on Applied natural language processing*, ANLC '92, page 152155, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.
5. Alexander Clark, Chris Fox, and Shalom Lappin, editors. *The Handbook of Computational Linguistics and Natural Language Processing*. Blackwell Handbooks in Linguistics. John Wiley & Sons, 2010.
6. Hoa Trang Dang and Martha Palmer. The role of semantic roles in disambiguating verb senses. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL '05, page 4249, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics.
7. Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107113, January 2008.
8. Ekaterina Euyko, Christian Chiarcos, and Antonio Pareja Lora. Ontology-based interface specifications for an nlp pipeline architecture. In *Proc. 6th LREC*, 2008.
9. Inc. Facebook. Facebook reports second quarter 2012 results. Technical report, Facebook, 2012.
10. Barbara B. Greene and Gerald M. Rubin. Automatic grammatical tagging of English. Department of Linguistics, Brown University, Providence, Rhode Island, 1971.
11. Taher H. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the 11th international conference on World Wide Web*, WWW '02, page 517526, New York, NY, USA, 2002. ACM.
12. Nancy Ide and Jean Vronis. Word sense disambiguation: The state of the art. *Computational Linguistics*, 24:140, 1998.
13. Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, page 135146, New York, NY, USA, 2010. ACM.
14. Hwee Tou Ng and Hian Beng Lee. Integrating multiple knowledge sources to disambiguate word sense: an exemplar-based approach. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, page 4047, Stroudsburg, PA, USA, 1996. Association for Computational Linguistics.
15. Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. *The PageRank Citation Ranking: Bringing Order to the Web*. 1999.
16. Ravi Sinha and Rada Mihalcea. Unsupervised Graph-basedWord sense disambiguation using measures of word semantic similarity. In *Proceedings of the International Conference on Semantic Computing*, ICSC '07, page 363369, Washington, DC, USA, 2007. IEEE Computer Society.
17. L Smith, T Rindflesch, and WJ Wilbur. MedPost: a part-of-speech tagger for bioMedical text. *Bioinformatics (Oxford, England)*, 20(14):2320–2321, 2004.
18. Philip Stutz, Abraham Bernstein, and William Cohen. Signal/collect: graph algorithms for the (semantic) web. In *Proceedings of the 9th international semantic web conference*

on The semantic web - Volume Part I, ISWC'10, page 764780, Berlin, Heidelberg, 2010. Springer-Verlag.
19. A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260 –269, April 1967.
20. Weaver. Translation. *Repr. in: Locke, W.N. and Booth, A.D. (eds.) Machine translation of languages: fourteen essays (Cambridge, Mass.: Technology Press of the Massachusetts Institute of Technology, 1955)*, 1949.
21. Yorick Wilks and Mark Stevenson. The grammar of sense: Using part-of-speech tags as a first step in semantic disambiguation. *Nat. Lang. Eng.*, 4(2):135143, June 1998.