# Supervised Lexical Chaining

*A THESIS*

*submitted by*

**ABHISHEK GHOSE**

*for the award of the degree*

*of*

**MASTER OF SCIENCE**
(by Research)


**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.**

**July 2011**

# THESIS CERTIFICATE

This is to certify that the thesis entitled **Supervised Lexical Chaining**, submitted by **Abhishek Ghose**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr. B. Ravindran**
Research Guide
Associate Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

# ACKNOWLEDGEMENTS

This research is a product of many helping hands. I would like to begin acknowledging these contributions by expressing my heartfelt gratitude to my advisor, Dr. B. Ravindran. I do not think it would have been possible for this work to assume its present form without his unrelenting encouragement. Successes in my research were few and far in between; for other times, comprised of long periods of disappointing results, failed approaches, personal lapses and delays, I am indebted to him for his guidance and moral support. Indeed, in most instances, his help has been more in the capacity of an excellent friend than an advisor.

I also wish to thank Dr. Deepak Khemani, Dr. Sutanu Chakraborti, Dr. Shankar Balachandran and Dr. Ashish Tendulkar who graciously accomodated me in their busy schedules when I needed their guidance.

The fact that computational infrastructure was never a concern was made possible by the staff and my friends at the Reconfigurable and Intelligent Systems Engineering (RISE) laboratory at IIT Madras. I owe my gratitude to Dr. V. Kamakoti for providing us with such a wonderful facility. I would like to thank Shivashankar Subramanian, Yousuf Sait, Arpit Joshi and Girish Rao for promptly making resources in RISE available for my use, whenever I requested them.

I would like to especially thank my friend Yousuf Sait; my worries with errant systems-related issues usually ended with requesting him for help, which was readily extended, often at the expense of his own time.

I am deeply obliged to my friends Raghunandan, Balaji Lakshman, Bala Sanjeevi, Preethi Chandur, Vijay Ram Chandrasekaran, D. R. Lakshminarasimhaiah, Sonal Oswal, Esha Ghosh, Yara Pavan, Pratik Gupte, Anil Patelia and Vineet Rajani,

who often served as sounding boards for my ideas and never hesitated in expressing their critical opinion of them. I thank Balaji Lakshman also for inspiring me with his zest for trying out new things.

The delay in finishing my thesis had the unintended consequence of having to continue working on it after I had taken up a job. The much needed encouragement to attend to my thesis, after office hours, was provided by Shyam Rajagopalan. Without his support, finishing my research and thesis would have taken much longer. I am also grateful to Harendra Mishra and Srinibas Swain for their support during this period. I wish to thank my colleague and friend Renzil D'Souza whose help during office hours often left me with ample time to dedicate to my thesis.

For proofreading my thesis, and pointing out some glaring mistakes in it, I would like to thank Bala Sanjeevi and Preethi Chandur. Their help was especially invaluable on account of the fact that I had requested their review at an extremely short notice and was willingly provided with it.

It is probably redundant to mention how much I rely on the love and support provided by my parents and my brother. With my research, as with any other enterprise I have undertaken, they have been exceedingly patient and supportive. Without their continued faith in me, not much would be possible.

# ABSTRACT

KEYWORDS: Word Sense Disambiguation, Lexical Chains, Supervised Learning, Hidden Markov Models, Segmentation

Lexical chaining is a method of grouping semantically related words in a document. Groups thus obtained are known as lexical chains. Lexical chains provide a rich representation of text, and have been used in various tasks like discourse analysis, summarization, corrections of malapropisms, amongst many others, with reasonable success. However, despite the general applicability of the method, its use is limited by the fact that chaining algorithms often group weakly related or unrelated words together. The large amount of time required for mining chains from a document also make it unsuitable for certain tasks. In our research, we look at applying supervised learning methods to address these drawbacks.

We propose two supervised algorithms as viable alternatives to classical algorithms. These algorithms rely on certain probabilistic properties of usage of words in text. We empirically establish the relevance of these properties to rapid construction of high quality lexical chains through experiments we have performed. Using lexical chains formed over sense tagged documents as training data, along with a knowledge of these properties, our algorithms are shown to be capable of reliably constructing chains on a test set of documents.

Although, we believe that exploring supervised learning for chaining is a worthy investigation in its own right, we provide certain encouraging results in defence of the approach. We compare our algorithms to a classical chaining algorithm and report a 44% improvement in quality and 55 times improvement in speed. Our experiments were performed on the SemCor 3.0 dataset.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **DP** | Dynamic Programming |
| **HMM** | Hidden Markov Model |
| **NN** | Noun, singular or mass |
| **NNS** | Noun, plural |
| **NP** | Proper noun, singular |
| **NPS** | Proper noun, plural |
| **OPTICS** | Ordering Points To Identify the Clustering Structure |
| **POS** | Part Of Speech |
| **STLC** | Sense Tagged Lexical Chaining |
| **WSD** | Word Sense Disambiguation |

# CHAPTER 1

# Introduction

## 1.1 Overview

In a typical text, semantically related words are used to collectively convey an idea. As examples, consider the following text snippets [1]:

> The *apple*, formally known as Malus Domestica, grows on a tree that is small and deciduous. The *fruit* matures in autumn and is typically 5 to 9 centimetres (2.0 to 3.5 in) in diameter. Five carpels arranged in a five-point star characterize the *Malus Domestica*.
>
> Some *wolf* pairs have been reported to prey on dogs by having one wolf lure the dog out into heavy brush where the second *animal* waits in ambush.

In the first example, the words *apple*, *Malus Domestica* and *fruit* are semantically related: *Malus Domestica* is a synonym of *apple*, while *fruit* is the general class that *apple* is a specific instance of. These words may be seen to represent essentially the same entity – apple. In the second example, the word *animal* is used to imply *wolf*. These words are also related, since *wolf* is a specific kind of *animal*.

Using related words to describe an idea is an indispensable accessory in writing text. Although the use of such words may be technically inaccurate – all *fruits* are not *apples* – an author relies on their relaxed connotations to cast an idea into words and introduce it in text, in relationship to other similarly represented ideas. An

[1] adapted from *Wikipedia*, *http://www.wikipedia.org*

arrangement of such groups of related words, leading from one idea to another, is what imparts meaning to text as a symbiotic collection of ideas.

The occurrence of such related words, often as a group in a localized region in text, is loosely said to form a *context*. A context is not only significant as a means of expression, but also is instrumental in interpretation of text, where interpretation involves understanding the intended meaning of words that possess multiple meanings, making sense of metaphors, identifying sarcasm etc. An example is the interpretation of the word *brush* in the second example sentence above. The context set up by words like *prey*, *lure* and *ambush* indicate that *brush* denotes scrub vegetation, and not a device like a paintbrush. Individual ambiguities in words that are part of a context are resolved by their collective indication of an idea.

The significance of contexts to understanding of text make their automatic identification highly desirable for various text mining algorithms. An algorithm with the ability to recognize contexts may use them to go beyond mere superficial processing of text and possibly mimic human understanding of text.

Methods of automatically identifying contexts intimately relate to the precise definition of a context. A "related set of words" is a vague definition of a context; the task at hand often makes it necessary to further qualify this definition. Such definitions range from only considering words within a fixed distance from each other as forming a context, to using sophisticated mathematical models of word distribution in a text to describe it. Our work concerns itself with a family of algorithms called *lexical chaining* that are used to identify contexts. A lexical chaining algorithm identifies sets of words called *lexical chains* as contexts, with words in the set adhering to certain constraints regarding relationships to each other and their physical location in text. The groups of words: {*apple, Malus Domestica, fruit*} and { *wolf, animal*}, from the sample sentences above, respectively, are examples of lexical chains that a chaining algorithm might discover.

Existing lexical chaining algorithms are not entirely accurate in discovering contexts and are time-consuming enough to be unsuitable for online processing of documents. In our work we investigate approaches to address these shortcomings. We propose two algorithms that score over existing algorithms in these respects, and demonstrate their utility with comparative tests we have performed. Although our research specifically looks at lexical chaining, our aim is to cater to the broader objective of making context identification reliable and efficient, thus opening up further possibilities for their application.

## 1.2 Lexical Chaining - A Brief Overview

The idea of lexical chaining and the first chaining algorithm were proposed by Morris and Hirst (1991) . Various other chaining algorithms have since been suggested. Although chaining algorithms often significantly differ in their implementations, they are characterised by certain general principles regarding how they perform chaining and certain general properties of the chains discovered. We present a list of these general characteristics below. Items 1-3 on the list characterize the implementation of chaining algorithms. Item 5 is a property of the chains discovered. Item 4 is both a characteristic of implementations of chaining algorithms as well as chains discovered.

**1. An external knowledge source**

To group words together a chaining algorithm needs to know what words are related and in what manner. Referring to the examples above, a chaining algorithm must have a way to know that *apple* is a specific kind of *fruit* or a *wolf* is a specific kind of an *animal*. Such knowledge is usually provided through an external knowledge source that a chaining algorithm consults. The first chaining algorithm (Morris and Hirst (1991)) used the Rogets Thesaurus for the purpose. Most modern algorithms use a dictionary like Wordnet (described in Section 1.4).

**2. Constraints on the "closeness" of relationships that may be allowed**

An external knowledge source does not list relationships between *every* pair of words in a language. To find the relationship between two words one may often need to look up the definitions of the words the first word is defined in terms of, look up the definitions of words in these definitions and so on, until the second word is encountered . For example, while consulting the Merriam Webster dictionary one may infer that a *dog* and a *buffalo* are both mammals, and hence related, only by noting that *dog* is defined to be a *mammal* and *buffalo* is defined to be a *bovid*, which is defined to be a *ruminant* and a ruminant, in turn, is defined to be a *mammal*.

In short, two words may be transitively related. A chaining algorithm must be both equipped to discover such relationships and be wary of following too long a transitive path. Long paths tend to unreasonably relate words. Morris and Hirst (1991) provides an example of the risks of allowing unlimited transitivity: the transitive chain {*cow, sheep, wool, scarf, boots, hat, snow*}, where every two consecutive words are related, misleads a chaining algorithm to assume *cow* and *snow* are related and miss potential context boundaries.

Chaining algorithms typically impose a restriction on the lengths of such transitive relations.

**3. Constraints on the "type" of relationships that may be allowed**

Lexical chaining algorithms specify what kind of relationships between words they consider valid. For example, a part-of/whole-of relationship, in which one entity is a part of the other entity, like the relationship between *finger nail* and *finger*, is considered to be a valid relationship in the algorithm proposed in St-Onge (1995), but not in the one proposed in Galley and McKeown (2003).

**4. Locality of words**

Chains also assume that words may be indicative of a context only within a

certain local region in text. This is not a strict rule as certain words are considered related even if they occur far apart in text, but such physical nearness is preferred. The assumption stems from the observation that individual ideas are often described in highly localized regions.

**5. Word Sense Disambiguation (WSD)**

For a word that has more than one meaning, or a *polysemous* word, its relationship to others depends on the meaning it represents in a particular occurrence. If the word *dog* occurs as a part of the term *hot dog* in a text, it clearly is not related to *canine*. Hence, chaining algorithms determine the intended meaning of polysemous words as a necessary side-effect – since without this knowledge it is impossible to semantically group words. The meaning of a word is technically known as its *sense*. Identifying the most suitable sense for a word occurrence is known as *Word Sense Disambiguation* (WSD).

□

A chaining algorithm may be seen as a means to form clusters of words, each with a central idea, using items 1-4 from the above list as heuristics. The above characteristics broadly describe the essence of chaining algorithms. In the following sections we discuss further details of chaining algorithms by looking at the kinds of relationships they identify (Section 1.3), discussing Wordnet - a popularly used external knowledge source (Section 1.4) and detailing two particular implementations (Sections 1.4.1 and 1.4.2). Section 1.5 rounds up our discussion on chaining algorithms by looking at some of their applications.

It may be noted that most lexical chaining algorithms discover relationships only between nouns. Henceforth, when we mention words in the context of chaining they may be assumed to be nouns unless otherwise stated.

## 1.3 Patterns Captured

We looked at examples of some relationships a chaining algorithm may identify in Section 1.1: *wolf-animal*, *apple-fruit*. This section discusses in detail the kind of direct relationships chaining algorithms identify. These direct relationships act as building blocks for indirect transitive relationships that a chaining algorithm may identify.

Semantic relationships between words, or *lexical cohesion* as they are technically known (Morris and Hirst (1991)), fall under the broader category of relationships known as *cohesion*. In addition to semantic relations, cohesion is realized with back–reference and conjunction (Morris and Hirst (1991)).

Lexical cohesion can be further classified into the following types (Morris and Hirst (1991)):

1. Reiteration with identity of reference:

   Mary bit into a *peach*.
   Unfortunately the *peach* wasn't ripe.

   Both occurrences of the word peach are used to refer to the same entity.

2. Reiteration without identity of reference:

   Mary ate some *peaches*.
   She likes *peaches* very much.

   The second use of the word peaches does not refer to the peaches Mary ate, but peaches in general.

3. Reiteration by means of a superordinate etc:

   The *apple*, formally known as Malus Domestica, grows on a tree that is small and deciduous. The *fruit* matures in autumn and is typically 5 to 9 centimetres (2.0 to 3.5 in) in diameter.

   This example has been mentioned before. The relationship between apple and fruit here is that of reiteration by means of a superordinate since a more general term or a superordinate, *fruit*, refers to a specific instance, the subordinate, of the class.

   Superordinates and subordinates are alternatively known as *hypernyms* and *hyponyms* respectively. This is the terminology we would use hence.

*Synonymy* may also be used to achieve a similar kind of reiteration. There exists a relationship of synonymy between two words if their meanings are identical. Ex. *pleasure*, *joy*.

4. Systematic semantic relation (systematically classifiable):

   Mary likes *green* apples.
   She doesn't like *red* ones.

   Here, the lexical relation involves words that usually co-occur, the relationships between which can be systematically classified, in one of the following ways:
   (a) The words are members of an ordered set *one*, *two*, *three*
   (b) The words are members of an unordered set *white, black, red*
   (c) The words are antonyms of each other i.e. they exhibit oppositeness. Ex. *Good, evil*
   (d) The words are related through a whole-of/part-of relationship. This relationship is technically known as *Holonymy/Meronymy*. 'X' is a meronym of 'Y' if 'X' is a part of 'Y'. For example, a *nail* is a meronym of a *finger*. This makes 'Y' a holonym of 'X' i.e. *finger* is the holonym of *nail*.

5. Non-systematic semantic relation (not systematically classifiable):

   Mary spent three hours in the *garden* yesterday.
   She was *digging* potatoes.

   This is yet another form of lexical cohesion in which words are related due to frequent co-occurrence. The difference between this form of cohesion and the one discussed previously is the co-occurrence here is non-systematic – the exact relationship between words is hard to explain but they do tend to show up in similar contexts. Included in this class are examples like *car, lights, turning* that are connected in the context of driving a car. Out of the context, however, they don't exhibit a systematic relationship. Word associations are also included in this class of relationships. Ex. *priest - church*, *whistle - stop*.

The first three kinds of relationships are generally referred to as *reiteration* and the latter two as *collocation*.

As mentioned before, chaining algorithms are aided by an external knowledge source in identifying these relationships. The knowledge source is used as a first step to consult the relationships listed above. These are used to further relate words that may be transitively linked, subject to certain constraints characteristic of the particular algorithm being used.

Except non-systematic semantic relations, lexical chaining algorithms can identify and use most instances of the other relationships mentioned. The reason non-systematic semantic relations are hard to identify is, being atypical and dependent strongly on context, they are usually not listed in a knowledge source; a *canine* is a hypernym of *dog* regardless of context, hence this relation may be found listed in a knowledge source, but it is unlikely to find *car* and *turning* mentioned as related, as the existence of the relationship is highly specific to a context.

## 1.4 Existing Approaches to Lexical Chaining

In this section, we take a look at some approaches to lexical chaining to gain an insight into how chaining algorithms work. We had outlined some general principles that chaining algorithms abide by in Section 1.2. This section delves into the details of two chaining algorithms to look at differences in the implementation of these principles.

The first chaining algorithm we look at was proposed by St-Onge (1995). We discuss this algorithm since it was the first chaining algorithm proposed that uses the popular external knowledge source, Wordnet, and also, since we use it to create our training dataset. The chaining algorithm suggested by Galley and McKeown (2003) is subsequently presented, as an example of an alternative chaining algorithm. Another reason for discussing the latter is that amongst classical algorithms this has been known to provide the best WSD accuracy.

We begin with an overview of Wordnet since most modern chaining algorithms use it as an external knowledge resource.

### 1.4.1 Wordnet

Wordnet is a lexical database of English, that contains nouns, verbs, adjectives and adverbs grouped into sets of synonyms known as synsets, with each synset describing a distinct concept. A synset is comprised of a set of words that are used to describe the concept it represents. A polysemous word, therefore, would appear in more than one synset. A synset also contains a *gloss* or a *definition* describing the relevant concept. Fig 1.1 shows the word fish and the various synsets it belongs too. A synset is represented by its definition.

Wordnet connects synsets by means of *semantic* or *lexical* relationships. A semantic relationship occurs between the concepts synsets represent, while a lexical relationship exists between words. The relationship between *canine* and *dog* is an example of a semantic relationship – Wordnet precisely expresses this as hypernymy from the synset containing the words {*dog, domestic dog, Canis Familiaris*} to the synset with the words {*canine, canid*}. Antonymy is an example of a lexical relationship. *Black* is an antonym of *white*; however *blackness* and *whiteness*, which belong to the same synsets as black and white respectively, are not antonyms. Since we are concerned only with nouns, we summarize the relationships Wordnet provides between noun synsets or noun words:

1. Semantic Relationships:
   (a) Hypernymy/Hyponymy
   (b) Holonymy/Meronymy – Wordnet refines the relationship of holonymy/meronymy into the following types:
      i. Component-Object – *branch* is a component of *tree*.
      ii. Member-Collection – *fish* is a member of a *school* of fish.
      iii. Material-Object – *Aluminium* is the material the object *airplane* is made of.
2. Lexical Relationships:
   (a) Synonymy
   (b) Antonymy

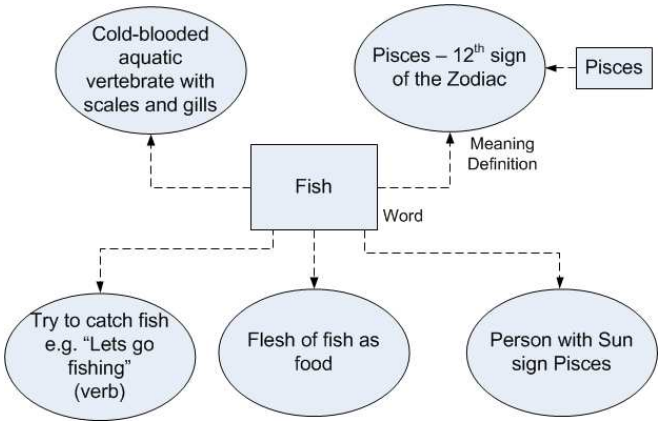


Figure 1.1: A polysemous word and its senses. A rectangle represents a word. Ellipses represent its senses. The word-sense relationship is indicated by a dashed line from a word to its sense.

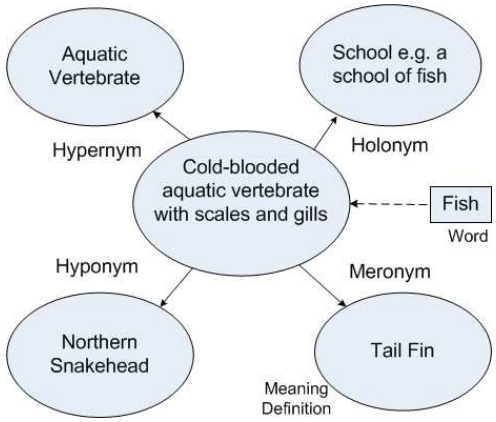


Figure 1.2: Relationships between a particular sense of the word *fish* to other senses in Wordnet.

Fig 1.2 illustrates semantic relationships between a synset containing the word *fish* and other synsets in Wordnet.

As an enriched dictionary, Wordnet is particularly suited for various tasks in natural language processing since it makes a lot implicit human knowledge explicit, and thus, usable by computers. Specifically, for the purpose of chaining it serves as a ready and easily accessible resource for discovering the relationships discussed in Section 1.3.

The interested reader may find further details on Wordnet in (Fellbaum (1998), Miller (1995)) .

We use the NLTK API for Wordnet (Bird and Klein (2009), Loper and Bird (2002)) in our research.

### 1.4.2 Lexical Chaining – St. Onge

We look at our first chaining algorithm in this section. This algorithm was proposed by St-Onge (1995) and applied to the task of detection and correction of malapropisms in text. We use the algorithm to create our training dataset.

The algorithm uses Wordnet as a source of semantic knowledge. It defines the following high-level categorization of relationships in Wordnet:

- Upward: hypernymy, holonymy
- Downward: hyponymy, meronymy
- Horizontal: also-see, antonymy

Intuitively, an upward direction points to something more "general".

It further defines the following relationships, in terms of the above categories, between two nouns:

1. Extra Strong: the nouns are identical or have identical base forms (e.g. *foot*, *feet* )
2. Strong: two words share a strong relationship if at least one of the following conditions are true:
   - the words share a synset
   - there is a horizontal link between some synset of the words
   - any kind of link between a synset of each word if one is a compound word or phrase including the other one

   Fig 1.3 provides an example of a Strong relationship, between the words *individual* and *person*.
3. Medium Strong: There is at least one allowable path between the some sense of each word. An allowable path is a path with the following characteristics:

   - No link may precede an upward link

- There may not be more than one change of direction except in the case that there is a change from upward to downward direction using a horizontal link
- The length of this path may not be more than a predetermined length (St. Onges thesis suggests 5 edges). We refer to this as the allowable path length.

Fig 1.4 illustrates a Medium Strong relationship between the words *apple* and *carrot*.

The above relationships are listed in decreasing order of their strengths. The strength of all Medium Strong relationships are not the same and decrease with path length and no. of change of directions. Specifically,

$$\begin{aligned} strength &= C - path\_length \\ &\quad - k * no\_of\_changes\_in\_direction \end{aligned} \tag{1.1}$$

Here, $C$ and $k$ are parameters to be determined.

The algorithm proceeds by scanning one sentence at a time, and processing them as follows:

1. Look for Extra Strong relationships between words in the sentence and chains already created. If such relationships are found add words to the respective chains.
2. Amongst the leftover words, look for Extra Strong relationship to chains – this is to check there are any potential Extra Strong relationships between these words and words added in step 1. If found, add these words to their respective chains. Now, look for Strong relationships between leftover words and chains, and if found, add words to the respective chains.
3. In a manner analogous to 2, from amongst the leftover words we look for Extra Strong relationships, strong relationships and Medium Strong relationships, in that order, and add words to respective chains.
4. The leftover words from 3 are used to form new chains.

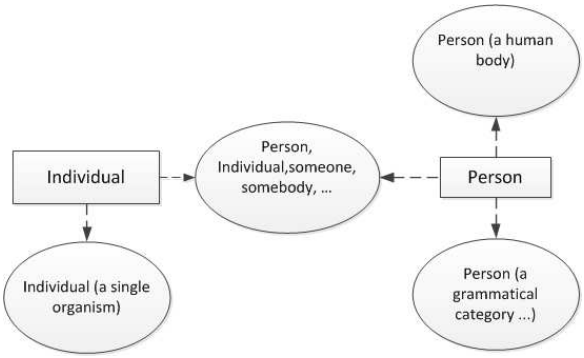


Figure 1.3: Strong Relationship

In case of ties between chains for accepting a candidate word, the word goes to the chain that was most recently modified. Candidature to chains is also limited by the *scope* of search: a word cannot be member of a chain via a Strong relationship if the sentence it comes from is more than 7 sentences away from the sentence the last word added to the chain comes from. The search scope of a Medium Strong relationship is 3. Extra Strong relationships have infinite search scope.

When a word is added to a chain, the senses that are not part of the relationship between this word and the word it is related to in the chain (let us call it $w$) are dropped from both words. This elimination of senses is further cascaded backwards through related words in the chain, starting with $w$. This cascaded elimination of senses results in word sense disambiguation in a chain.

Consider the following text [1]:

> American democratic thought, pointed up the relation between the Protestant movement in this country and development of a social religion, which he called the American Democratic Faith.
>
> Those familiar with his work will remember that he placed the incipience of the democratic faith at around 1850.

Fig 1.5 provides an example of a lexical chain involving the words {*thought*,

[1] *Christianity and the Tragic Vision*, Brainard Cheney. From the SemCor dataset.

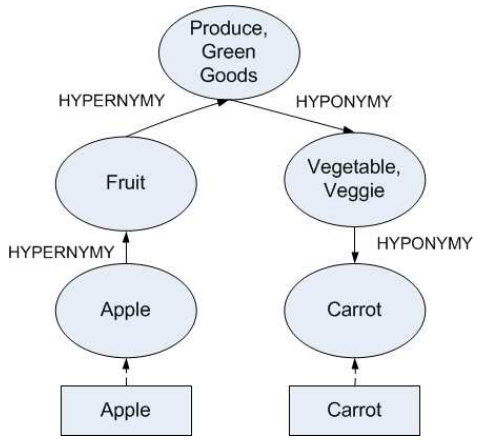


Figure 1.4: Medium Strong Relationship

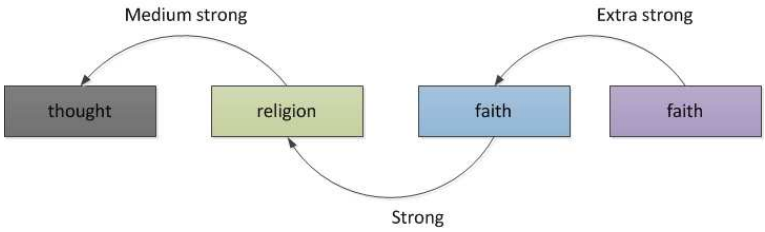


Figure 1.5: Sample lexical chain and relationships in it.

*religion, faith, faith*} discovered by St. Onges algorithm in the above text. Figs 1.6 and 1.7 illustrate the relationships between the words in the chain.

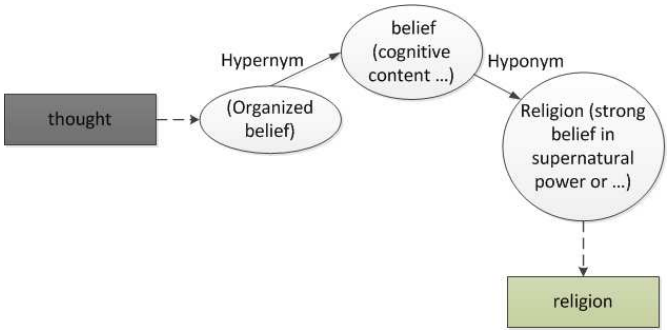


Figure 1.6: Medium Strong relationship in chain.


Figure 1.7: Strong Relationship in chain.

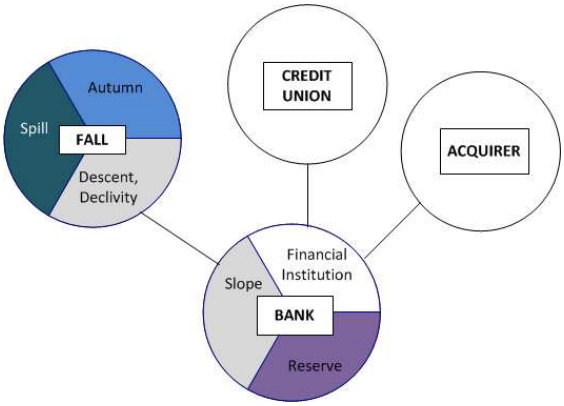


Figure 1.8: Disambiguation Graph

### 1.4.3 Lexical Chaining – Galley McKeown

The chaining algorithm proposed by Galley McKeown significantly differs from St. Onges algorithm in the fact that WSD and chain construction are distinct phases. The algorithm can be decomposed into the following steps:

**Step 1.** The text is read in and all possible relationships between all senses of all words are noted. No disambiguation is done at this point. This relationship structure can be visualized as in the graph in Figure 1.8, where a node represents a word. Each node depicts the various senses of a word by colored regions. Since Wordnet relates senses, the links do not actually connect nodes, but specific senses of words. This graph is called the *disambiguation graph*.

To represent the disambiguation graph an array indexed with senses in Wordnet is used. When a word is encountered while scanning the text, multiple entries of it are made in the array against all its senses/indexes. Figure 1.9 shows this structure. A sense of the word is linked to a sense of another word if they are hypernyms/hyponyms or siblings – hyponyms of hypernyms e.g.dog and wolf (In

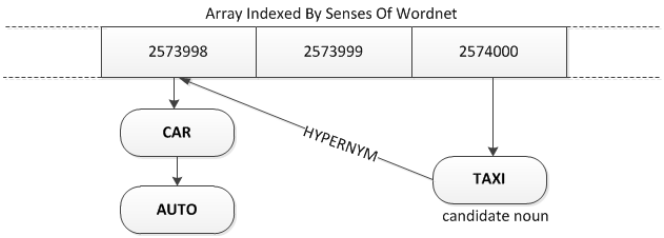


Figure 1.9: Storing a Disambiguation Graph

St. Onges algorithm siblings were accounted for by medium strong relationships). Links are assigned weights depending on the type of relationship and the distance in text between the words being related. Notably, different occurrences of the same word do not form separate entries in the array and only contribute to more links in the graph.

**Step 2.** WSD is performed for a word, by adding the weights of all links for a sense, for each sense. The sense with the highest sum of weights is assumed to be the intended sense for a word. Note that since different occurrences of word are not separately represented, this sense applies to all occurrences of the word. Thus, for a word, a constraint of one sense per discourse is imposed. (This constraint is justified by observations presented in Gale *et al.* (1992))

**Step 3.** The final step creates lexical chains by deleting links in the disambiguation graph that exist between incorrect senses of words. The semantic links that remain determine a unique interpretation of the text, and also represent the lexical chains produced by this algorithm.

## 1.5 Applications

The ability to identify contexts and context boundaries render chaining useful for various tasks. Context boundaries divide the text into broad topics, from which sentences may be selected to form summaries (Barzilay and Elhadad (1997)). Contexts identified by lexical chains may be used to discover discourse structure

in text – normally considered to be a difficult problem (Morris and Hirst (1991)). Lexical chains may be used to compare parts of two texts to automatically generate hyperlinks between them (Green (1996), Green (1997)). Lexical chaining may be also applied to the tasks of topic detection and tracking (Stokes (2004)) and segmentation (Stokes (2004)) by investigating regions of text where a number of chains end and new ones begin signifying the end of a topic and the start of a new one respectively. Information retrieval (Al-halimi and Kazman (1998), Stairmand (1997)) and detection and correction of malapropisms (St-Onge (1995)) are some other tasks chaining has been applied to with reasonable success.

In general, any task that can profit from a semantically rich representation of its underlying text can use lexical chaining as an effective pre-processing step. Although, identification of contexts is the primary reason for use of chaining in a task, the following factors further contribute to its utility:

1. A chaining algorithm does not require domain-specific data for training. [1]
2. For certain tasks (e.g. discourse analysis) it offers the easier alternative of performing superficial lexical analysis as opposed to (often) complex and deep semantic analysis.
3. Many traditional text processing algorithms use the notion that a context can be approximated by a fixed sized window around a word, where the size is empirically determined; given that chains may span the entire document and the length of chains is not a parameter for such algorithms, lexical chaining provides the interesting option of using variable length contexts.

## 1.6 Contribution of thesis

### 1.6.1 Objectives and Motivation

The objective of our research is to qualitatively improve the process of chaining. Existing lexical chaining algorithms suffer from certain drawbacks; however,

[1] Sometimes in the case of WSD this can turn out to be a disadvantage as we discuss in Chapter 4.

their varied applicability is a strong motivation to explore means to address these shortcomings. In our work, we investigate approaches to remedy the following limitations common to most chaining algorithms:

1. Low WSD accuracy – Lexical chaining algorithms offer poor WSD accuracy (Michelizzi (2005), Nelken and Shieber (2007)). This implies that the meaning of a word may be possibly misinterpreted during chaining and it could be grouped in an inappropriate chain. This negatively affects any task that uses lexical chaining as a prior step.
2. High processing time –Trying to determine whether two words are related or not by either a direct or a transitive semantic relationship, using an external knowledge source, and repeating the exercise for many pairs of words in a text, is time-consuming. This results in high processing time of chaining algorithms, and makes them unsuitable for online processing of documents. Research dedicated to this aspect of chaining (Silber and McCoy (2002), Galley and McKeown (2003)) has produced fast implementations; however, even these processing times leave much to be desired when considering processing of documents in real-time.

We propose two supervised chaining algorithms that have yielded promising results in these respects.

### 1.6.2 Overview of our work

Our algorithms are supervised in nature and use sense-tagged documents as training data. We measure the WSD accuracies and processing times of our algorithms and compare them against those of the lexical chaining algorithm proposed by St. Onge. St. Onges' algorithm is used for comparison because it has been shown to outperform other chaining algorithms on a variety of tasks by Stokes (2004). However, it may be noted that this specific chaining algorithm is not necessary for our methods to work as they are general enough to be used with little or no modification with other existing chaining algorithms. We also compare our algorithms against St. Onges on the task of segmentation to verify that the improvement in performance doesn't come at the cost of decreased ability to detect cohesive bound-

aries in text. For this task, we use the method of preparing datasets proposed in Choi (2000) and the segmentation algorithm proposed in Stokes (2004).

## 1.7 Organization of Thesis

The purpose of this chapter was to present an overview of lexical chaining - by looking at what it does, the need to do it, how such an algorithm may work, and its applications. The objective of our thesis is also presented. The remaining chapters elaborate our work, presenting the following details:

1. Chapter 2, Model Assumptions – The algorithms we propose assume a certain model of word and sense distribution in text. This chapter discusses the basis of these assumptions.
2. Chapter 3, Algorithms – This chapter presents our algorithms, the metrics by which we evaluate them and the results of the evaluation.
3. Chapter 4, Conclusions and Further Work – We discuss the conclusion from our work, possible improvements in our methods, and avenues for further research in this chapter.

# CHAPTER 2

# Model Assumptions

Our end objective is to propose chaining algorithms with higher WSD accuracy than existing algorithms, that take much lesser time to execute. In this chapter we look at two basic premises our models and algorithms rest on. These premises assume certain properties of word and sense distributions to be true. The first property is a reliable indicator of WSD accuracy. This chapter introduces the property and establishes its correspondence with WSD accuracy. Our models use this correspondence as their central premise.

We also present a second property of word-sense distribution in this chapter. This property suggests a way to narrow down search for intended senses of words. Adopted as a premise, this helps us reduce the time complexity of our chaining algorithms.

Before we elaborate on these properties, in the next section we introduce a few ideas that find significant use in our work and would help make subsequent discussions precise.

## 2.1 A few ideas involved in our work

### 2.1.1 Sense Tagged Lexical Chaining (STLC)

As mentioned previously, the process of chaining obtains both a contextual fragmentation of text in form of lexical chains and WSD. To independently evaluate the chaining capability of an algorithm, we introduce a constrained form of chaining called *Sense Tagged Lexical Chaining* in our work. STLC is restricted to discover chains alone, by

1. Providing one sense for each noun in the text, for the algorithm to choose from.
2. Not performing sense elimination.

For St. Onges algorithm this implies that there is no cascading of elimination to be done (as mentioned in Section 1.4.1), and candidature to chains is tested only against the senses (one per word) we provide.

Since WSD during chaining is prone to errors and it indirectly impacts chain formation, performing STLC allows us to observe the effectiveness of chaining or identifying contexts of an algorithm in isolation. If we provide the correct sense per word, STLC produces the optimal set of lexical chains computable by the algorithm.

Indeed, we use STLC in this manner to form an optimal set of lexical chains. Our algorithms are supervised in nature and they use this optimal set as training data to learn contexts.

STLC need not only be performed with correct senses; however, use of incorrect senses would promote flawed chaining. Section 2.2 provides an example of STLC performed with incorrect sense tags for use in one of our experiments.

### 2.1.2 Ordering Points To Identify the Clustering Structure (OPTICS)

As would be detailed later, we perform clustering of lexical chains as part of our work (Section 2.2.2). We use OPTICS (Ankerst *et al.* (1999)) for the purpose. The technique is briefly described here.

To begin with, OPTICS does not strictly produce clusters. It produces an intermediate representation - an ordering of data points - that corresponds to the density structure of the dataset. Actual clusters may be conveniently identified us-

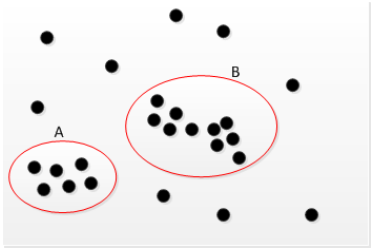


Figure 2.1: Clusters defined by dense regions

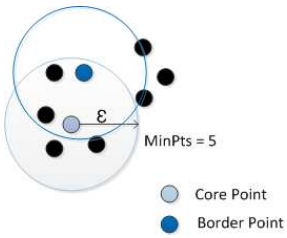


Figure 2.2: Core and border points

ing this representation. The clusters thus discovered conform closely to the notion of clusters suggested by the DBSCAN (Ester *et al.* (1996)) clustering algorithm.

Both OPTICS and DBSCAN define clusters as localized regions of high spatial density. For example in Fig. 2.1 the clusters identified would be the regions A and B. This is a very natural definition of clusters and frees these algorithms of any additional assumptions over the data. This is different from an algorithm like the k-means clustering algorithm which being a Voronoi tessellation of the plane, can only produce convex clusters. The minimum requirement in case of OPTICS and DBSCAN is that of a distance measure between points in the database.

These algorithms implicitly also recognize *noise*, or points that do not belong to any cluster, as against certain other clustering algorithms that mandate membership to a cluster for every point.

A cluster, as per either of these algorithms, is composed of 2 kinds of points – *core points* and *border points*. A core point is a point inside the cluster that possesses a minimum required density around it. This notion is formalized by specifying that a core point must have at least a *MinPts* number of points within a neighborhood

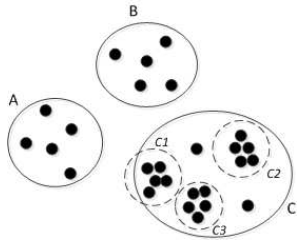


Figure 2.3: Clusters of different densities. No one setting of parameters can identify both the clusterings A, B, C and C1, C2, C3

radius of $\epsilon$. A border point, may have a lower density, but must be present within the $\epsilon$ neighborhood of a core point. Core and border points are illustrated in Fig 2.2. It is interesting that a distinction is made between core and border points – this takes into account the practical observation that the points on the periphery of a cluster are located at neighborhoods of lower density compared to points within, and cannot be identified to be a member of the cluster by enforcing a global required neighborhood density value for all points in a cluster.

A point $p$ located in the $\epsilon$-neighborhood of a core point $q$ is said to be *directly density reachable* from $q$. This relationship is not symmetric, since $p$ might be a border point. A point $p$ is said to be *density reachable* from point $q$, if there exists a set of points $\{q, x_1, x_2,...,x_i, x_{i+1},..., x_n, p\}$ such that $x_1$ is directly density reachable from $q$, $x_{i+1}$ is directly density reachable from $x_i$, and $p$ is directly density reachable from $x_n$. DBSCAN identifies a cluster as a maximal set of points formed by a core point and all points that are density reachable from it.

Although DBSCAN efficiently computes density based clusters, it needs to be provided with input parameters $\epsilon$ and *MinPts*. A particular setting of these parameters promotes discovery of clusters of a particular density only. For example, in Fig 2.3 there is no one setting of parameters that can discover both clusters A,B,C *and* clusters C1, C2, C3. Too high a suggestion for density would miss the relatively sparse clusters A,B,C; parameter inputs corresponding to a lower density would group together clusters C1, C2, C3 under cluster C. Of course, the problem can be remedied by running DBSCAN for various input parameters on the database

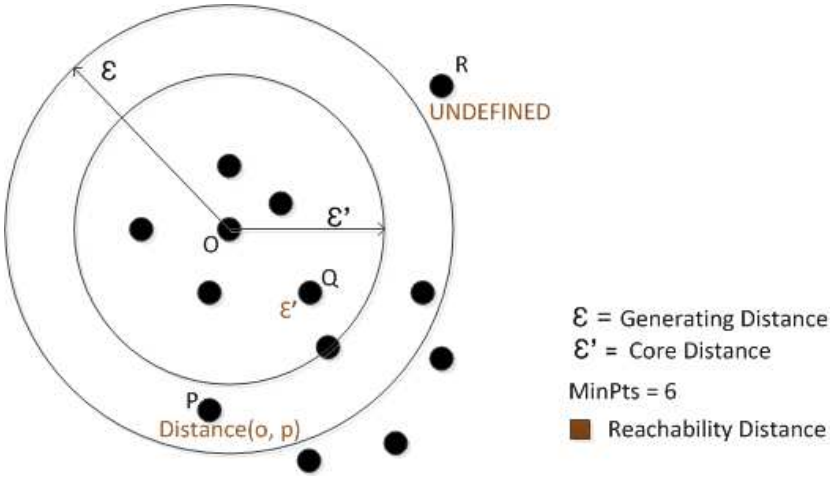


Figure 2.4: Core and reachability distances. Note how the reachability distance of points P,Q,R are different owing to their locations.

- however, the process is not only time consuming, but depending on how finely we vary our parameters in discrete steps, there is always a chance we might miss some interesting clustering of the dataset.

OPTICS offers a solution to these problems by providing an intermediate representation which corresponds to an infinite setting of density parameters. It also provides a means of visualizing its output from which clusters for a particular parameter setting of density maybe conveniently obtained.

OPTICS takes the parameters $\epsilon$ and *MinPts* as its input. *MinPts* denotes the same notion as in DBSCAN. $\epsilon$ denotes the *generating distance* – the largest distance considered for clusters. Clusters can be extracted for all $\epsilon_i$ such that $0 \leq \epsilon_i \leq \epsilon$.

OPTICS defines the following additional terms:

1. *Core Distance* of a point – The core distance of a point is defined for points satisfying the core point criteria wrt $\epsilon$ and *MinPts*. The core distance is the distance to the $MinPts^{th}$ point in the *epsilon* neighborhood of the core point.

   To take an example, lets say *MinPts* = 6. A core point $O$ might have 8 points within its $\epsilon$ neighborhood. The distance $\epsilon'$ from $O$ to the $6^{th}$ farthest point from it becomes its core distance. This is shown in Fig 2.4.

   Put simply, the core distance of a point is the smallest neighborhood one may consider so that it stays a core point wrt *MinPts*.

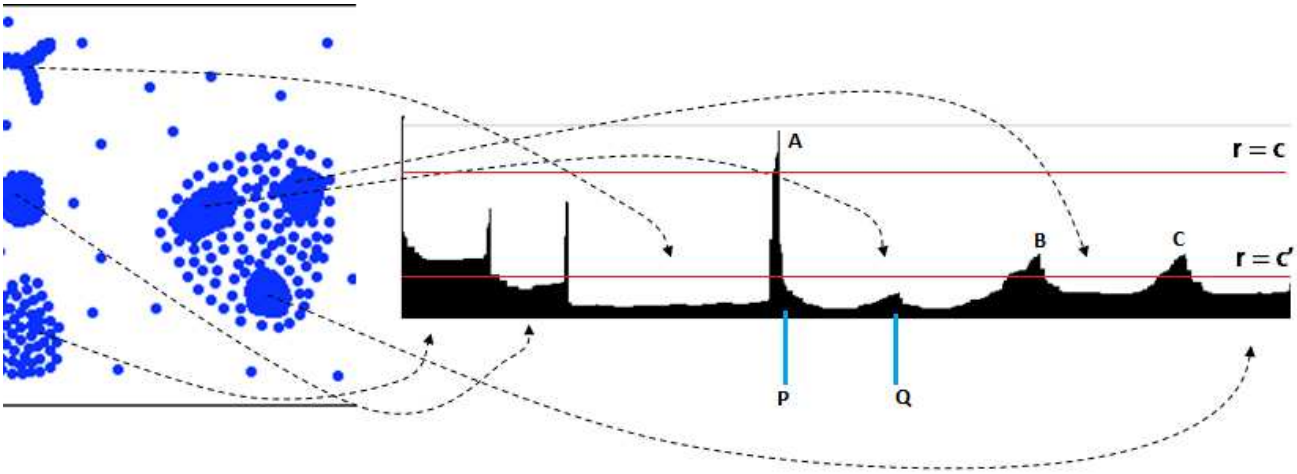


Figure 2.5: Reachability plot. Clusters are shown mapped to regions in the plot with dashed arrows.

2. *Reachability Distance* of a point $p$ wrt a point $q$ – The reachability distance is defined only if $q$ is a core point and $p$ is within the $\epsilon$ neighborhood of $q$. It is defined as the maximum of the core distance of $q$ and distance between $p$ and $q$.

   Note that smaller values of generating distance imply a lesser number of points have a valid value of reachability distance wrt $q$.

When processing a dataset, OPTICS maintains two data structures - a min-priority queue, with priority key as reachability distances, and an ordered list. The ordered list and the reachability distances are the final output produced. A simplified pseudo-code of OPTICS is presented as Algorithm 1.

Briefly, what OPTICS does is it starts with an unprocessed point that it adds to the ordered list. If this also happens to be a core point, the reachability distances of all its neighbors (wrt $\epsilon, MinPts$) are added/updated into the priority queue. Then, points in this priority queue are processed one by one. When a point, say $q$, is removed from the queue for processing it is added to the ordered list, and the neighbors of $q$ now get added to or moved up in the queue, depending on their reachability distance from $q$. A reachability distance value is updated only if the new reachability distance wrt to $q$ is smaller than the existing value.

```
Data: Database D, ϵ, MinPts.
List L and min-priority queue Q.
Result: L
for each unprocessed point p ∈ D do
    Mark p as processed
    Add p to end of L
    Make Q empty
    if p is a core point then
        UpdateForPoint( p )
        while Q is not empty do
            q ← Q.GetFirstElement()
            Mark q as processed
            Add q to end of L
            if q is a core point then
                UpdateForPoint( q )
            end
        end
    end
end

UpdateForPoint( d ) {
N ← GetNeighbors(d)
for each o ∈ N do
    if o is not processed then
        new_reachability_distance ← reachability distance wrt d
        if reachability distance of o not defined then
            Q.insert(o, new_reachability_distance )
        else
            if current reachability distance of o > new_reachability_distance then
                Q.update(o, new_reachability_distance)
            end
        end
    end
end
}
```

**Algorithm 1:** Simplified pseudo-code, OPTICS

From the ordered list and reachability distances computed by OPTICS, a *reachability plot* is constructed. A reachability plot is graph between points on the x-axis, ordered as per the ordered list output by OPTICS vs their reachability distances on the y-axis. Fig 2.5 shows an example [1]. Concentrating on the section $P - Q$ of the plot, the working of OPTICS may be better understood. Points in the neighborhood of $P$ get arranged wrt reachability distances from $P$, after the call to *UpdateForPoint*($p$) is made. Since this is the order in which points are added to the ordered list by calls to $Q$.*GetFirstElement*(), we see a rising curve as we move further to the right of $P$. However when a point $q$ is encountered, it recalculates the reachability distances of some points wrt $q$, and adds new points that were missing from the neighborhood of $p$. Since the new reachability distances are updated only when they are less than the existing values, we see relatively low values again to the right of $q$, which again, are seen to be slowly increasing.

To visually obtain a rough idea of the clusters for a particular density setting, we draw a line parallel to the x-axis for a value of reachability distance and note the divisions the peaks above the line cause. The points in each division form a cluster. This is also illustrated in Fig 2.5. The line corresponding to $r = c$ identifies 2 clusters in the dataset, while the line $r = c'$ identifies 5. The segregations on the graph provide a rough idea of clusters for a particular setting - the original paper (Ankerst *et al.* (1999)) discusses an accurate algorithm to do so. As the figure illustrates, lowering the reachability distance, makes clusters less inclusive, and enables us to discover more densely packed clusters. Lowering the value of reachability distance corresponds to lowering the value of the generating distance. This makes the core point criteria more strict, so lesser points satisfy it, which leads to lesser number of density reachable points being included in a cluster.

We use the implementation of OPTICS provided by the tool WEKA (Hall *et al.* (2009)) in our research.

[1] Adapted from the tutorial: http://osl.iu.edu/ chemuell/projects/presentations/optics-v1.pdf

### 2.1.3 Similarity between lexical chains

We are often faced with the need to find similarity between two lexical chains. A similarity measure for chains would also enable us to find similarity between documents represented by their chains. Although there has been some work towards formulating good similarity measures for chains (Green (1997), Nahnsen *et al.* (2005)), there is no one well established method for doing so. For our work we have developed a notion of similarity that accounts for the various semantic relationships a chain is composed of. We elaborate the idea using two sample chains P and Q , which are represented by multi-sets of word-sense pairs:

$$P = \{(w_1, s_a), (w_2, s_b), (w_1, s_a)\}$$
$$Q = \{(w_3, s_b), (w_1, s_a), (w_2, s_b), (w_4, s_c)\}$$

Here, $w_i$ is a noun word, $s_i$ is a synset.
Also, let us assume $s_c$ is one edge away from $s_b$ in Wordnet (an example could be synsets related by a hypernymy/hyponymy relationship).

The similarity between P and Q is calculated as follows:

**Step 1** : Account for word-sense pair overlap .

Create frequency vectors of word sense pairs from P,Q- let us call them $\vec{v_{P_1}}$, $\vec{v_{Q_1}}$ respectively. The first component of similarity is calculated between these two vectors as:

$$Sim_a = cos(\vec{v_{P_1}}, \vec{v_{Q_1}}) = \frac{\vec{v_{P_1}}.\vec{v_{Q_1}}}{\left\|\vec{v_{P_1}}\right\| . \left\|\vec{v_{Q_1}}\right\|}$$

In this case,

$$\vec{v_{P_1}} = \{(w_1, s_a) : 2, (w_2, s_b) : 1\}$$
$$\vec{v_{Q_1}} = \{(w_3, s_b) : 1, (w_1, s_a) : 1, (w_2, s_b) : 1, (w_4, s_c) : 1\}$$
$$Sim_a = \frac{2.1 + 1.1}{\sqrt{(2^2 + 1^2)}.\sqrt{(1^2 + 1^2 + 1^2 + 1^2)}}$$
$$= \frac{3}{\sqrt{20}} = 0.67$$

**Step 2** : We now look for senses that appear in both chains that are not paired with the same word. The value of similarity obtained from the previous step is subtracted from 1.0 (the maximum similarity possible) and is scaled with $\alpha$ ($0 \leq \alpha \leq 1$). The quantity $Sim_b$ is calculated as follows:

Let the frequency vectors of senses be $\vec{v_{P_2}}$, $\vec{v_{Q_2}}$. Then,

$$Sim_b = (1 - Sim_a)\alpha \frac{\vec{v_{P_2}}.\vec{v_{Q_2}} - \vec{v_{P_1}}.\vec{v_{Q_1}}}{\|\vec{v_{P_2}}\|.\|\vec{v_{Q_2}}\|}$$

Here, assuming $\alpha = 0.1$, we have:

$$\vec{v_{P_2}} = \{s_a : 2, s_b : 1\}$$
$$\vec{v_{Q_2}} = \{s_b : 2, s_a : 1, s_c : 1\}$$
$$Sim_b = (1 - 0.67) * 0.1 * \frac{(2.1 + 1.2) - 3}{\sqrt{(5)}.\sqrt{(6)}}$$
$$= 0.006$$

**Step 3** : Senses in a chain that are one edge away from senses in the other chains, are accounted for, in this step. The edge may be of any type. The sum of similarities from the previous two steps subtracted from 1.0 is scaled with $\beta$ ($0 \leq \beta \leq 1$) and the quantity $Sim_c$ is calculated:

Let,

Distinct senses in P = m

Distinct senses in chain Q = n

Senses in P with at least one sense in Q that is one edge away in Wordnet= b
Senses in Q with at least one sense in P that is one edge away in Wordnet= a

$$Sim_c = (1 - Sim_a - Sim_b)\beta\frac{a+b}{m+n}$$

Here, assuming $\beta = 0.1$:

$$\begin{aligned} m = 2, n = 3 \\ a = 1, b = 1 \\ Sim_c = (1 - 0.67 - 0.006) * 0.1 * \frac{2}{5} \\ = 0.013 \end{aligned}$$

We define similarity and distance between chains as,

$$Similarity = Sim_a + Sim_b + Sim_c \tag{2.1}$$

$$Distance = 1 - Similarity \tag{2.2}$$

In our case,

$$\begin{aligned} Similarity = 0.67 + 0.006 + 0.013 = 0.689 \\ Distance = 1 - 0.689 = 0.311 \end{aligned}$$

Note that the value of *similarity* is in the range [0,1], since each component accounts only for the sum of previous similarities subtracted from 1. $\alpha$ and $\beta$ are parameters to be determined empirically.

Similarity between documents represented by lexical chains is calculated as:

1. Let the first document contain $m$ chains. For each chain in the first document,

Table 2.1: Sample similarities between chains of documents

| | $L_{2,1}$ | $L_{2,2}$ |
|---|---|---|
| $L_{1,1}$ | 0.2 | 0.6 |
| $L_{1,2}$ | 0.9 | 0.4 |
| $L_{1,3}$ | 0.4 | 0.8 |

Table 2.2: Maximum Similarities

| | Maximum Similarity |
|---|---|
| $L_{1,1}$ | 0.6 |
| $L_{1,2}$ | 0.9 |
| $L_{1,3}$ | 0.8 |
| $L_{2,1}$ | 0.9 |
| $L_{2,2}$ | 0.8 |

calculate its similarity with each chain of the second document and note the value of the maximum of these similarities. Sum these maximum similarities and call it $Total_{Sim1}$.

2. Do the same for the second document i.e. for each chain in this document, calculate its maximum similarity across all chains, in the first document. Sum these maximum similarities and call it $Total_{Sim2}$. Assume the second document has $n$ chains.

3. Similarity between the documents is calculated as

$$\frac{Total_{Sim1} + Total_{Sim2}}{m + n} \tag{2.3}$$

We provide an example of calculating the similarity between two documents using the sample similarities listed in Table 2.1. $L_{i,j}$ denotes the $j^{th}$ chain of document $i$ in the table. The values in the cells denote similarity values between the corresponding chains. The maximum similarities are listed in Table 2.2.

Here,

$$Total_{sim1} = 0.6 + 0.9 + 0.8 = 2.3$$
$$Total_{sim2} = 0.9 + 0.8 = 1.7$$
$$m = 3, n = 2$$
$$Similarity = \frac{2.3 + 1.7}{3 + 2}$$
$$= 0.8$$

The above procedure of calculating similarity is summarized by the following formula:

$$Sim_{docs}(doc_1, doc_2) = \frac{\sum_{i=1}^{n} \max_{1 \le j \le m} Sim_{chains}(c_i^1, c_j^2) + \sum_{j=1}^{m} \max_{1 \le i \le n} Sim_{chains}(c_j^2, c_i^1)}{n + m} \tag{2.4}$$

Here,

$Sim_{docs}(doc_1, doc_2)$ is the similarity between the document 1 and document 2,

$c_i^1$ is the $i^{th}$ chain of document 1,

$c_i^2$ is the $j^{th}$ chain of document 2,

$Sim_{chains}(x, y)$ is the similarity between chains x and y,

as determined by using Equation 2.1.

The parameters $\alpha, \beta$ were determined using a classification task over documents in the SEMCOR 3.0 corpus. The SEMCOR 3.0 corpus contains 186 documents with nouns tagged with senses from Wordnet by humans. Documents in the SEMCOR 3.0 corpus are a subset of the documents in the Brown corpus, each of which is assigned a category. We determined the values for parameters $\alpha$, $\beta$ using a k-NN classification task for documents, with class labels as the categories from the Brown corpus.

The values we use are:

$$\alpha = 0.2$$

$$\beta = 0.2$$

We find this similarity measure particularly suited for our work due to the following reasons:

1. Lexical chains are essentially clusters of semantically related words. Therefore, ideally, two different chains formed on the same document should be as disjoint as possible. The value of similarity calculated using our measure between pairs of chains in a document, averaged over all pairs, is very close to zero. Thus, our similarity measure complements representation by lexical chains.

2. Since the dataset we use, SEMCOR 3.0, is a small one with 186 documents, similarity measures using statistically motivated representations like *Latent Semantic Indexing* (Deerwester *et al.* (1990)) or *tf-idf* (Jones (1972)) may not be suitable. Additionally, our candidates for measuring similarity are chains - which are much smaller than an average document. This further incentivizes the use of a similarity measure like ours, that consults an external knowledge source.

## 2.2 Assumptions

We now take a look at some of the premises that serve as the foundation of our algorithms. Our first observation identifies an indicator of WSD accuracy. The second observation looks at the range of senses a word may assume in similar contexts. Their role in our algorithms is briefly discussed after the observations are presented.

### 2.2.1 Chain similarity as an indicator of WSD accuracy

If we have sufficient sense-tagged data in a training set, to approximate the word sense distribution in the whole corpus, it is reasonable to assume that the contexts defined by lexical chains in the training set would be similar to the ones present in the test set. Since the sense of a word determines the context it contributes to, we use this assumption to further claim that the particular assignment of senses to words in the test set is a fairly accurate assignment - one with high WSD accuracy - which produces, via STLC, chains similar to those in the training set. We consider using STLC here since we are evaluating our imposed sense assignments and want the chaining algorithm to produce lexical chains using these.

We substantiate this claim by performing the following experiment. We use 150 documents from SEMCOR 3.0 as our training set, with 36 documents in the test set.

**Step 1** Conduct STLC on each document of the training set using St. Onges algorithm. Use the correct sense tags provided, for STLC.

**Step 2** Store all chains obtained in the last step, in a list – we call this list our *chain database*. The chain database holds all contexts from the training set.

Given our assumption that the word and sense distributions in the training set closely reflect the corresponding distributions in the test set, the chain database also represents contexts in the test set.

**Step 3** Pick a particular document from the test set. To each word in it, assign a random sense from its set of possible senses. We call such an assignment of senses to words a *sense configuration* of the document. Note the WSD accuracy for this sense configuration.

**Step 4** Perform STLC on the sense configuration obtained in **Step 3**. Note that since sense assignments are random, most assignments are probably incorrect. For

Table 2.3: Correlation Coefficients

| Allowable Path Length | Pearson | Spearman |
|---|---|---|
| 3 | 0.70 | 0.67 |
| 4 | 0.70 | 0.68 |
| 5 | 0.67 | 0.65 |

each lexical chain obtained, find out its similarity with each chain from the chain database, and retain the highest similarity value. This value is referred to as the *context similarity* of the chain. For a chain in the test set this value represents the context it is most similar to in the training set. Average the context similarities for the test document – we refer to this value as the *document context similarity*.

**Step 5** Repeat **Steps 3** and **4** for multiple sense-configurations of a document, for multiple test documents. Calculate the correlation between the document context similarities and WSD accuracies for the various sense-configurations.

If our claim is true, then a sense configuration with high document context similarity would also result in a high WSD accuracy. This indeed turns out to be the case, as is seen from the positive correlation between the quantities presented in Table 2.3. Rank correlation coefficients (both Spearman and Pearson) are presented from the above experiment. We had run our experiment with different values of allowable path lengths – 3, 4, 5.

The correlation is not very high when we use an allowable path length of 5. Since chaining with an allowable path length of 5 is computationally intensive and it seems to offer us no relative advantage over shorter allowable path lengths, we will only consider allowable path lengths 3 and 4 in further discussions.

That context similarity is an indicator of WSD accuracy is a significant observation that our models use. Context similarity to contexts from the training set is the property that our algorithms attempt to maximize, to achieve high WSD accuracy.

### 2.2.2 Low sense-entropy in clusters

St. Onges chaining algorithm processes individual documents to find chains. It is possible that contexts described by chains in different documents may be similar. For example, while a chain in one document may contain words like {*flower, plants, daisy*}, another document might contain a chain with the words {*flower, plants, rose*}. It seems highly likely that both chains represent highly similar contexts. Thus, while talking about unique contexts present in a *corpus*, it is reasonable to group similar contexts across documents. In our work, this grouping is achieved by clustering of chains from the training set.

We use OPTICS for our purpose. Once the clusters are formed we do not look at a cluster as being composed of individual chains; we regard the cluster itself as a *global lexical chain* spanning multiple documents. Since a lexical chain represents a context in a document, we analogously think of a global lexical chain as representing a *global context*. In practice, a cluster is stored as a multiset of word-sense pairs from its constituent chains. The frequency counts of word-sense pairs are stored too.

Since words in a cluster come from different documents, with senses already tagged, it is interesting to consider the possibility of a cluster containing two or more instances of the same word, not all with the same sense. Such a possibility could be the result of:

1. Grouping together of not-so similar chains by the clustering algorithm
2. Fine-grained sense distinctions for a word in Wordnet, with different chains in the same cluster containing the word with these slightly different senses.

   For example, Wordnet lists the following senses, amongst others, for the noun *title*:

   (a) the name of a work of art or literary composition etc.
   (b) a general or descriptive heading for a section of a written work

In such a case, it is possible that the same word ends up in a cluster with different senses.

However, it turns out that this rarely occurs. We measure this possibility by calculating the sense entropy of a cluster using the following formulae:

Sense Entropy of a cluster $c$,

$$H_c = \frac{\sum_{w_i \in W_c} H_c(w_i)}{\mid W_c \mid} \tag{2.5}$$

where,

$W_c$ is the set of unique words in a cluster,

$H_c(w_i)$ is the sense entropy of word $w_i$ in cluster $c$

Sense Entropy of word $w$ in a cluster $c$,

$$H_c(w) = -\sum_{s_i \in S} p(s_i) \log_{|S|} p(s_i) \tag{2.6}$$

where,

$w$ is a word,

$S$ = set of senses for word w,

$s_i$ = a particular sense of w, $s_i \in$ S

$p(s_i)$ = probability of the word occurring with the sense $s_i$
in the cluster c (estimated with the relative frequency)

$\sum_{s_i \in S} p(s_i) = 1$

and,

$H_c(w) \in [0,1]$

$H_c(w)$ from Equation 2.6 takes a minimum value of 0 when a particular $p(s_i)$ has a value of 1. It takes a maximum value of 1, when all the probabilities are equal. In general, the more skewed the probability values are, or the more a particular sense dominates in occurrence, the closer $H_c(w)$ is to 0.

Sense entropies of all words (across all clusters) are averaged. For various parameter settings of OPTICS we find the average entropy to be close to zero.

Given this extremely low value, we approximate the sense of a word in a cluster by one sense – the *dominant sense*, or the sense that occurs with the highest relative frequency for the word in the cluster.

Our algorithms try to assign senses to words in a way that the contexts they define are similar to the contexts represented by chain clusters from the training set. Amongst other considerations, this is done by looking at all clusters, one at a time, and trying to guess which of these contexts is a word most likely to be a part of, and with what sense. Our observation tells us that to represent a particular cluster the word can assume just one sense – its dominant sense from the training data in the cluster. This helps in making our search for correct senses efficient.

## 2.3 Role of assumptions

What roles the above assumptions play in formulating our algorithms? The first assumption provides us with the all important indicator for high WSD accuracy. Potentially, this assumption alone can be used to come up with a good chained representation of text. We could start with a random sense configuration and move to better ones using a heuristic search algorithm like *simulated annealing* (Kirkpatrick *et al.* (1983)) with an appropriate neighborhood function. In fact, we had initially attempted a similar approach. But the need to perform chaining on every sense configuration renders the method time-consuming and motivates the need for convenient models and efficient algorithms.

The significance of the second assumption lies in the fact that while trying to identify which cluster a word best represents, all possible senses of the word need not be investigated.

## 2.4 Summary

In this chapter we looked at the two assumptions our models and algorithms make, and the justification for the assumptions. Section 2.3 discussed briefly the significance of the assumptions for our models and algorithms. We also went over a few ideas like STLC, OPTICS and similarity between lexical chains which were important to understand the assumptions, and would aid in understanding further discussions. Although seemingly disparate, these ideas prepare the ground for understanding our models and algorithms that we present in the next chapter.

# CHAPTER 3

# Models and Algorithms

In this chapter we look at our models, algorithms and their evaluation.

The purpose of our algorithms is to improve the quality of lexical chaining as measured by WSD accuracy. Our observation regarding the correlation of WSD accuracy and context similarity discussed in Section 2.2 indicates that one way to do this is to assign senses to words and group them in a way that they represent a context. We find all contexts in the dataset during a training phase by:

1. Performing STLC with correct senses on documents in the training set
2. Clustering the chains thus obtained using OPTICS with our similarity/distance measure. This process was described in Section 2.2.2. These clusters provide us with all contexts in the dataset, and hence, possible contexts a group may represent.

   Indeed, our assumption here is that the training set is a good representation of the complete dataset.

The role of our models is to dictate exactly in what manner such a grouping may be done, by making certain assumptions about how contexts are distributed in text. Our first model assumes the context a word represents is independent of the contexts its neighboring words represent. Our second model assumes that the context represented by a word influences the context represented by the word following it.

We evaluate our algorithms on the WSD accuracy they obtain, their processing times and their performance on the task of segmentation.

Before we begin a discussion of our models and algorithms, we briefly describe some technical terms we use and conventions we follow, in the rest of the chapter:

1. We often refer to a cluster as a *global chain* or a *global context*, for the reason mentioned in Section 2.2.2.

2. When a word, as part of a group, represents a particular global context or a cluster, we loosely say it *belongs* to the cluster; in the sense of a datapoint belonging to a cluster.

3. The groups our algorithms form may be composed of non-consecutive words in the text. Note that in a clustering with $k$ clusters a maximum of $k$ groups are possible in a document - since every word must be assigned to one of the clusters. We refer to a consecutive set of words belonging to the same cluster as a *fragment*. A group may have more than one fragments. Consider two documents with cluster assignments as follows:

$$\text{Document A: } w_1 - c_1,\ w_2 - c_1,\ w_3 - c_2,\ w_4 - c_1,\ w_5 - c_2$$
$$\text{Document B: } w_1 - c_1,\ w_2 - c_1,\ w_3 - c_1,\ w_4 - c_2,\ w_5 - c_2$$

   Here,
   $w_i$ is the $i^{th}$ word in a document,
   $c_i$ is the $i^{th}$ cluster
   Both documents have two groups.
   Document 1 has the four fragments:
   (a) $w_1 - c_1, w_2 - c_1$
   (b) $w_3 - c_2$
   (c) $w_4 - c_1$
   (d) $w_5 - c_2$

   while Document 2 has two fragments:
   (a) $w_1 - c_1, w_2 - c_1, w_3 - c_1$
   (b) $w_4 - c_2, w_5 - c_2$

   These groups are analogous to traditional lexical chains since they represent unity in context, and in saying our algorithms perform lexical chaining we imply the identification of such groups.

4. We use the probability of a cluster producing a word as a measure of the likelihood of the cluster representing itself using the word in text. A few things to be noted are:
   (a) Clusters are composed of word-sense pairs. Thus, strictly speaking, they can produce only word-sense pairs. The reason we can afford to talk about the probability of a cluster producing a word is given our assumption of one sense per word per cluster (Section 2.2.2), the probability of cluster producing a word with any sense is equal to the probability of producing it with its dominant sense.

(b) The probabilities are estimated by relative frequencies. For example, if a cluster contains the word-sense pairs – $(w_1 - s_1), (w_1 - s_1), (w_2 - s_2)$, the probability of the cluster producing the word-sense pairs $(w_1 - s_1)$ and $(w_2 - s_2)$ are estimated to be 0.67 and 0.33 respectively.

5. The complexities of our algorithms are presented in terms of $k$, the number of clusters in the model, and $n$, the number of words in a document that are present in the training data.

We now present our models and algorithms.

## 3.1 Models

### 3.1.1 Unigram Model

#### 3.1.1.1 Model

In our first model, we assume independence of contexts, i.e., the cluster the current word belongs to is not influenced by the cluster of the word before it. We call this model the *unigram model* in reference to the popular *n-gram* models used in the area of Natural Language Processing. n-gram models use statistical properties of occurrences of $n$ entities (which can be letters, syllables, words etc.) as a sequence, to model text. A good discussion of n-gram models may be found in Jurafsky and Martin (2008). Since this model assumes no dependence amongst neighboring words and thus considers word sequences of length 1 only, we refer to it as a unigram model.

The basic idea is to find out which global context a word best corresponds to by looking at the probability of each cluster producing the word and picking the highest.

Using the assumptions of the model alone leads to the following problems:

1. A particular noun is always assigned to the same cluster, since the probability of a cluster producing a word is fixed as determined in the training phase.

2. There is no restriction on the number of fragments a document might have.

To address these problems, we use the formulation for the task of *text segmentation* proposed in Utiyama and Isahara (2001). Although we defer a discussion of segmentation to Section 3.2.4, we describe the formulation here. The formulation is suitably modified to account for cluster assignments to words.

Let W = $w_1 w_2 ... w_n$ be document with n words, S = $S_1 S_2 ... S_m$ be a particular fragmentation of it with m fragments with $S_i$ representing the $i^{th}$ fragment, and C = $C_1 C_2 ... C_m$, be the respective assignments of clusters to fragments (i.e., $S_i$ is assigned to $C_i$). The probability of a fragmentation S (with its corresponding cluster assignment C) is given by:

$$P(S \mid W) = \frac{P(W \mid S)P(S)}{P(W)} \tag{3.1}$$

The optimal fragmentation and cluster assignments are obtained as:

$$\hat{S} = \arg\max_{S} P(W \mid S)P(S) \tag{3.2}$$

since P(W) is constant for a given document.
Let $W_i$ be the words in $i^{th}$ fragment containing $n_i$ words, denoted by $W_i = w_1^i w_2^i w_3^i ... w_{n_i}^i$. Then,

$$\begin{aligned} P(W \mid S)P(S) &= \prod_{i=1}^{m} P(W_i \mid S)P(S) \\ &= \prod_{i=1}^{m} P(W_i \mid S_i)P(S) \end{aligned} \tag{3.3}$$

In Equation 3.3, we use an independence assumption over fragments.

We use a non-informative prior for P(S): $n^{-m}$

Here,

$$\begin{aligned} P(W_i \mid S_i) =& P(W_i, C_i) \\ =& P(W_i \mid C_i) P(C_i) \\ =& [\prod_{j=1}^{n_i} P(w_j^i \mid C_i)] P(C_i) \end{aligned} \tag{3.4}$$

Equation 3.4 uses our unigram model assumption that choice of clusters for neighboring words are independent of each other.

$P(w_j^i \mid C_i)$ is estimated by the relative frequency of the word $w_j^i$ occurring in $C_i$, as determined from the training set. We will refer to this by $f(w_j^i, C_i)$. We now rewrite Equation (3.3) as,

$$P(W \mid S)P(S) = \prod_{i=1}^{m} \prod_{j=1}^{n_i} f(w_j^i, C_i) P(C_i) n^{-m} \tag{3.5}$$

Referring to Equation (3.2), we may now rewrite the optimality condition as:

$$\hat{S} = \arg\max_S \Big[ \prod_{i=1}^{m} \max_c \Big[ \prod_{j=1}^{n_i} f(w_j^i, c) P(c) \Big] n^{-m} \Big] \tag{3.6}$$

Here, $c$ is a cluster.

Note that the use of the non-informative prior controls the number of fragments that a document might have. A high number of fragments leads to a small value of the prior.

To deal with potential arithmetic underflows owing to multiplication of many small probability values, we use logarithms of the various quantities and restate our optimality condition as follows:

$$\begin{aligned} &\arg\max_S P(W \mid S)P(S) = \\ &\qquad \arg\max_S \log \Big[ P(W \mid S)P(S) \Big] \end{aligned} \tag{3.7}$$

where,

$$\max_{S} \log\Big[P(W \mid S)P(S)\Big] =$$
$$\sum_{i=1}^{m}\Big[max_c\Big[\sum_{j=1}^{n_i}\log f(w_j^i, c) + logP(c)\Big] - m\log n\Big] \tag{3.8}$$

#### 3.1.1.2 Algorithm

To efficiently calculate cluster and sense assignments for the words, we use *dynamic programming* (DP). The optimal substructure for a DP solution comes from the following observation:

> Consider a document with $n$ words that has been fragmented optimally (wrt to the probability of fragmentation). Let the last fragment have $k$ words. Then the document with first $n-k$ words must be optimally fragmented.

This substructure is optimal since if the above claim were false it is possible that the first $n-k$ words are fragmented in a way that has a higher probability. As the probability of fragmentation of the whole document is the product of probabilities of individual fragments (Equation 3.3), this new fragmentation is better than the original one - which is a contradiction since we assumed our original fragmentation is optimal.

The dynamic programming algorithm is presented as Algorithm 2.

The algorithm works by scanning one word at a time. When it is processing the $i^{th}$ word of the document, the optimal fragmentation for the sequence of words $\{w_1, w_2, w_3, ..., w_j\}$ $\forall j \in \{1, 2, ..., i-1\}$ is known. The algorithm inspects the probabilities of the length of the last fragment (the one that contains the $i^{th}$ word) being one word long (just containing the $i^{th}$ word) to $i$ words long (this being the only fragment in the document) - line 1 in Algorithm 2. Since the overall fragmentation

**Data**: Document d, with n words. prior[1...k] is prior probabilities of clusters.
**Result**: Fragment Boundaries (fr), Cluster Assignments (clust).

logPr[1...n], fr[1...n], clust[1...n]
// logPr[i] denotes the log of probability of optimal fragmentation till word i
// fr[i]=j implies, in the optimal fragment till word i, the last fragment begins at word j
// clust[i]=j implies, in the optimal fragmentation till word i, the last fragment is assigned cluster j
fr[1] ← 1
$\hat{c} \leftarrow \arg\max_c[\log P(w_1 \mid c) + \log \text{prior}[c]]$
clust[1] = $\hat{c}$
logPr[1] ← $\log P(w_1 \mid \hat{c})$ + log prior[$\hat{c}$] - log n
// $w_i$ denotes the $i^{th}$ word in d
// $c$ denotes a cluster

// find optimal fragmentation till $w_i$
1 **for** $i \leftarrow 2$ *to* $n$ **do**
  // initialize: assume $w_i$ is a fragment by itself
2  best_c = $\arg\max_c[\log P(w_i \mid c) + \log \text{prior}[c]]$
  best_logPr = $\log P(w_i \mid \hat{c})$ + log prior[$\hat{c}$] + logPr[i-1]
  best_fr = i
  // check if $w_j, w_{j+1}...w_i$ from the last fragment
  **for** $j \leftarrow$ *i-1 to 2* **do**
3  $\hat{c} = \arg\max_c \left[ \sum_{t=j}^{i} [\log P(w_t \mid c)] + \log \text{prior}[c] \right]$
    temp = $\sum_{t=j}^{i} [\log P(w_t \mid \hat{c})]$ + log prior[$\hat{c}$]
    temp = temp + logPr[j-1]
    **if** *temp > best_logPr* **then**
      best_logPr = temp
      best_c = $\hat{c}$
      best_fr = j
    **end**
  **end**
  // check for the possibility that $w_1, w_2, ...w_j$ is just one fragment
4  $\hat{c} = \arg\max_c \left[ \sum_{t=1}^{i} [\log P(w_t \mid c)] + \log \text{prior}[c] \right]$
  temp = $\sum_{t=1}^{i} [\log P(w_t \mid \hat{c})]$ + log prior[$\hat{c}$]
  **if** *temp > best_logPr* **then**
    best_logPr = temp
    best_c = $\hat{c}$
    best_fr = 1
  **end**
  logPr[i] = temp - log n
  clust[i]= best_c
  seg[i] = best_fr
**end**

**Algorithm 2:** DP algorithm for the Unigram Model

probability is composed of the fragmentation probability of the last fragment and those of fragments before it which have already been calculated, the algorithm decides on the length of the last fragment by selecting the one that contributes to the highest overall probability.

For a particular length of the last fragment the probability also depends on the cluster it belongs to - so while considering a particular length, the algorithm evaluates the probabilities of the fragment belonging to various clusters, one at a time, and picks the one with the highest value - lines 2, 3, 4 in Algorithm 2.

#### 3.1.1.3 Time Complexity

The time complexity of this algorithm is O($kn^3$), where n is the length of the document in words and k is the number of clusters. The complexity may be obtained as follows:

1. While processing the $j^{th}$ word, the algorithms looks at the possibility of the last fragment being of various lengths from 1 to $j$. This constitutes $j$ iterations.
2. In each iteration the algorithm considers the possibility of the last fragment belonging to each of the clusters. This leads to evaluating $k$ possibilities.
3. For a particular cluster, calculating the probability of a group of $i$ words belonging to it is $O(i)$, since the probability of each word belonging to it must be individually looked up.
4. From **1**, **2** and **3**, processing the $j^{th}$ word has the following complexity:

$$\overbrace{j}^{\text{lengths of fragment}} \times \overbrace{k \times j}^{\text{inspect clusters}} \tag{3.9}$$

   For the whole document with $n$ words, we have the following approximate number of operations:

$$\sum_{j=1}^{n} k \cdot j^2 \approx k \cdot n^3 \tag{3.10}$$

   Hence, the time complexity of our algorithm is $O(kn^3)$.

### 3.1.2 Bigram Model

#### 3.1.2.1 Model

In our second model, we assume that the cluster the current word belongs to is influenced by the cluster of the previous word. We refer to this model as the *bigram model* since sequences of two words are considered - a word and its predecessor.

Another way to state our assumption is to say that for determining the context of the current word, we consider looking at the context of the previous word as equivalent to considering all previous context assignments, i.e.

$$P[w_i = c_i \mid w_{i-1} = c_{i-1}, w_{i-2} = c_{i-2}, ..., w_1 = c_1] = P[w_i = c_i \mid w_{i-1} = c_{i-1}] \tag{3.11}$$

Here,

$w_i$ is the $i^{th}$ word in the document,

$c_i$ is the cluster $w_i$ is assigned to

This particular property of a system is known as the *Markov* property of order 1 [1]. We use a *Hidden Markov Model* to model our problem since they provide us with a well established probabilistic infrastructure for studying such systems:

1. There are hidden causes or causes not directly observable that manifest themselves through observable effects. In our case, the hidden causes are contexts, that produce words as observable effects.

   Technically, a hidden cause is also known as a *hidden state*, and an observable effect is known as an *observation symbol*.

2. The current state of the system influences what state the system would be in next. This is similar to the dependence of contexts we assume.

The bigram model is equivalent to a HMM with clusters as states. Finding clusters corresponding to words is identical to the problem of finding the sequence of

[1] The generalization is a Markov property of order $n$, where we look at $n$ predecessors. For us, $n$=1.

hidden states that has the highest probability of producing a sequence of observations symbols. This is a well studied problem for HMMs and an efficient solution is provided by the *Viterbi Algorithm*.

HMMs were introduced in a series of papers by L.E. Baum and his colleagues - Baum and Petrie (1966), Baum and Eagon (1967), Baum and Sell (1968), Baum *et al.* (1970), Baum (1972). Rabiner (1989) is a popular tutorial on the topic.

At a given time, a state in a HMM produces only one symbol. One might suspect that this makes this model inappropriate in our case since in addition to a word we also wish to predict its sense. But we note that given our approximation of the senses of a word in a cluster by one sense (Section 2.2.2), this is still equivalent to finding out the state has produced a particular word, and then assigning its dominant sense from the cluster. The Viterbi algorithm is modified accordingly to take this fact into account.

Also, to make a rather simple comparison to HMMs, we had previously stated that the observation symbols in our model are words. However, the observations in our model are more correctly word-sense pairs.

We denote a sequence of $T$ observations by $\{O_1, O_2, ..., O_T\}$, where $O_t$ represents the $t^{th}$ symbol in the sequence. The state the system is in when $O_t$ is produced by it, is denoted by $q_t$. Since states are clusters, $q_t \in \{c_1, c_2, c_3, ...., c_k\}$. The probability distribution of observation symbols for state $c_j$ is B = $\{b_j(\text{i})\}$, where $b_j(\text{i})$ is the probability of the $i^{th}$ observation symbol $(w_i, s_i)$ being produced by $c_j$, i.e.

$$b_j(i) = P[(w_i, s_i) \mid q_t = c_j] \tag{3.12}$$

The words in our document form our observation sequence: $O = w_1, w_2, ..., w_n$

We list our HMM parameters below:

1. $k$ , the number of states/clusters in our model.
2. $m$, the number of observation symbols in the model. These are all the word-sense pairs seen in the training set.

Table 3.1: Sample Transition Probabilities

| $a_{ij}$ | j=1 | 2 | 3 |
|---|---|---|---|
| i=1 | 0.5 | 0.5 | 0 |
| 2 | 0.33 | 0 | 0.67 |
| 3 | 0 | 1.0 | 0 |

3. $A = \{a_{ij}\}$, the state transition probability distribution, where

$$a_{ij} = P[q_{t+1} = c_j \mid q_t = c_i], 1 \leq i, j \leq k \tag{3.13}$$

The probabilities are estimated by frequencies. For a particular clustering, word-sense pairs in the training set are labelled by their clusters and the transitions between labels are calculated and used. For example, if we had 3 clusters $c_1, c_2, c_3$ and two documents in the training set with the following cluster to word assignments:

$$\begin{aligned} \text{Document A: } & w_1 - c_1, w_2 - c_1, w_3 - c_2, w_4 - c_1 \\ \text{Document B: } & w_1 - c_2, w_2 - c_3, w_3 - c_2, w_4 - c_3 \end{aligned}$$

the transition probabilities would be as listed in Table 3.1.

$a_{23} = 0.67$ since there are 2 such transitions: $(w_1 - c_2) \rightarrow (w_2 - c_3)$ and $(w_3 - c_2) \rightarrow (w_4 - c_3)$ from Document B – of the total 3 transitions from $q_2$. Other probabilities are calculated similarly. Note that values in a row in the table must add up to 1.

4. $B = \{b_j(i)\}$, the observation symbol probability distribution.

5. $\pi = \{\pi_i\}$, the initial state distribution, where

$$\pi_i = P[q_1 = c_i], 1 \leq i \tag{3.14}$$

The initial probabilities are also modelled using relative frequencies. As in the case of determining the state transition probability distribution, word-sense pairs in the training set are labelled with their clusters, for a particular clustering. The proportion of documents that start with the label $c_i$ form our estimate of $\pi_i$ for the particular clustering.

The model parameters are collectively denoted by $\lambda$.

#### 3.1.2.2 Algorithm

We now walk through the Viterbi algorithm. The notations and the presentation of the algorithm have been adopted from Rabiner (1989).

We assume we have T observations.

We define the quantity,

$$\delta_t(i) = \max_{q_1,q_2,...,q_{t-1}} P[q_1, q_2, ..., q_t = c_i, \\ (w_1, s_1), (w_2, s_2), ...(w_j, s_j), ...(w_t, s_t) \mid \lambda]$$

Here $\delta_t(i)$ is the highest probability along a path which accounts for the first t observations $w_j$ *and* the sense assignments $s_j$ to them, and ends in state $c_i$. $\lambda$ represents the model parameters. Our aim is to calculate $\delta_T(i)$ for $1 \leq i \leq k$ and pick the maximum value. This gives us the best estimate for the last state. As discussed later, the last state allows us to backtrack and discover the most probable sequence of states to have generated the observation sequence.

The induction step for $\delta_{t+1}$ is:

$$\delta_{t+1}(j) = \max_{1 \leq i \leq k} \left[\delta_t(i) \cdot a_{ij}\right] \cdot \max_{s \in S_{c_j, w_{t+1}}} b_j(t+1) \tag{3.15}$$

where, $S_{c_j,w_{t+1}}$ is the set of all senses for the word $w_{t+1}$ that cluster $c_j$ can generate.

but,

$$\max_{s \in S_{c_j,w_{t+1}}} b_j(t+1) = \max_{s \in S_{c_j,w_{t+1}}} P(w_{t+1}, s_{t+1}|c_j) \\ = P(w_{t+1}, \hat{s}_{j,w_{t+1}}|c_j)$$

where, $\hat{s}_{j,w_{t+1}}$ is the dominant sense of $w_{t+1}$ in state $c_j$.

Thus, Equation (3.15) may be rewritten as:

$$\delta_{t+1}(j) = \max_{1 \le i \le k} \left[ \delta_t(i) \cdot a_{ij} \right] \cdot P(w_{t+1}, \hat{s}_{j,w_{t+1}} | c_j) \tag{3.16}$$

Using the above definitions we present our algorithm as follows:

**1) Initialization:**

$$\begin{aligned} \delta_1(i) &= \pi_i b_i(1), \quad 1 \le i \le k \\ \psi_1(i) &= 0 \end{aligned}$$

where,

$\psi_t(j)$ is an array that is used to keep track of the argument that maximized Equation 3.16, for each $t$ and $j$. This is used to retrieve the state sequence.

**2) Recursion:**

$$\begin{aligned} \delta_{t+1}(j) &= \max_{1 \le i \le k} \left[ \delta_t(i) \cdot a_{ij} \right] \cdot P(w_{t+1}, \hat{s}_{j,w_{t+1}} | c_j) \\ \psi_t(j) &= \arg\max_{1 \le i \le k} [\delta_t(i) \cdot a_{ij}] \end{aligned}$$

where,

$1 \le i \le k$,

$2 \le t \le n$, where $n$ is length of the observation sequence.

**3) Termination:**

$$\begin{aligned} \hat{P} &= \max_{1 \le i \le k} \left[ \delta_n(i) \right] \\ \hat{q_n} &= \arg\max_{1 \le i \le k} \left[ \delta_n(i) \right] \end{aligned}$$

where,

$\hat{P}$ denotes the maximum probability of the observations being generated,

$\hat{q_n}$, the final hidden state.

**4) State sequence backtracking:**

$$\hat{q_t} = \delta_{t+1}(\hat{q_{t+1}}), \quad t = n-1, n-2, ..., 1 \tag{3.17}$$

#### 3.1.2.3 Time Complexity

The time complexity of the algorithm is O($nk^2$) - same as the Viterbi algorithm. This can be seen by noting that while calculating $\delta_t(i)$ from $\delta_t(i-1)$ for a state, we maximize over all clusters, contributing to $k$ computations. For every symbol, we compute $\delta_t(i)$ for all $k$ states. Thus for a symbol, we have $k^2$ computations. For an observation sequence of length $n$, we have $n \times k^2$ computations.

## 3.2 Evaluation

This section looks at the performance of our algorithms against St. Onges chaining algorithm on certain tasks. We discuss these tasks and the metrics we use to measure effectiveness at them, and compare results from the algorithms. We begin with a brief description of our experimental set-up.

### 3.2.1 Experimental Setup

We use the *SemCor 3.0* corpus, short for *Semantic Concordance 3.0* corpus, as our dataset. Documents in SemCor 3.0 are a subset of the Brown corpus. The Brown corpus is a collection of English documents with manually tagged part-of-speech of the words in it. The documents in the SemCor 3.0 corpus retain this information, and additionally possess manually tagged senses for words in it. The senses used are from WordNet. The number "3.0" in the name *SemCor 3.0* signifies the fact that senses from version 3.0 of WordNet were used to tag words. The original SemCor dataset is SemCor 1.6, which used senses from WordNet 1.6 for tags.

The corpus consists of a total of 352 documents, of which 186 documents have sense tags for many content words. This is the subset of SemCor 3.0 we use. The remaining 166 documents have tags exclusively for verbs.

In the relevant subset, 150 documents formed our training set and 36 documents were used as our test set.

Since St. Onges chaining algorithm uses only nouns, we only train on words from SemCor 3.0 that meet the following criteria:

1. The word must have one of the following part-of speech tags:

   (a) Noun, singular or mass. Marked with the tag NN.
   (b) Noun, plural - NNS
   (c) Proper noun, singular - NP
   (d) Proper noun, plural - NPS

   The tag names (NN, NNS, NP, NPS) come from the Penn TreeBank tag set (Marcus *et al.* (1993)).

2. The word must have a valid WordNet sense tag. Not all content words in documents have a WordNet sense tag. Also, a version of SemCor is upgraded automatically for using senses from a newer version of WordNet - by using a map between senses from the older WordNet to the newer version and re-tagging SemCor. In the process of upgrade, senses are occasionally *lost* - the newer WordNet does not have a sense corresponding to a sense from the older WordNet. In such cases, the SemCor word is tagged with a sense of 0. We discard such words too.

3. The word must have a valid WordNet lemma associated with it. WordNet indexes words by their lemmas or base forms *only*. Words with valid sense tags in SemCor also have the WordNet lemma mentioned. In case a word is missing this information, we discard the word, since otherwise it becomes difficult and potentially inaccurate to further process it using WordNet.

Statistics regarding words meeting the above criteria are presented in Table 3.2.

STLC is performed on the training set of documents using the sense tags provided, giving us our *training set of lexical chains* . These chains are clustered using OPTICS with the similarity/distance measure discussed in Section 2.1.2. We run OPTICS with the following settings of various parameters:

Table 3.2: Words per category used from SemCor 3.0

| **POS** | **Train** | **Test** |
|---|---|---|
| NN | 66,405 | 12,843 |
| NNS | 1 | 0 |
| NP | 6761 | 1980 |
| NPS | 0 | 0 |
| **Total** | 73,167 | 14,823 |

1. **Allowable Path length** - This is the maximum path length allowed in a medium strong relationship by St. Onges algorithm. Allowable path lengths of 3 and 4 were used (Section 2.2).

2. **Non-Trivial chain length**: Lexical chaining often produces numerous small chains. Their presence increases the processing times of algorithms that use chains as input, but no significant additional semantic information is obtained. Small chains with length less than the specified *non-trivial chain length* were discarded from our experiments. We used non-Trivial chain lengths of 3, 4 and 5.

3. **OPTICS parameters** - The values of *MinPts* used were 3 and 6. We used $\epsilon = 0.9$ for our experiments. Theoretically, the value of $\epsilon$ can be set to the maximum possible, since OPTICS would compute the ordering for any value of $\epsilon$ smaller than this. But, the maximum possible value means that every $\epsilon$ -neighborhood query would return all data points. This makes a run of OPTICS extremely resource intensive. It is for this reason we had to limit $\epsilon$ to 0.9 - a value slightly lesser than the maximum value of 1.0.

OPTICS was run for all 12 combinations of the above settings. From the respective reachability plots a total of 16 clusterings were visually identified (some plots indicated more than one clustering, present at different values of the reachability distance). For each run OPTICS also identified noise (points not belonging to any cluster).

The clusters identified define the states or global contexts for our algorithms.

We now describe the various tasks and means of evaluating performance on them.

Table 3.3: WSD Accuracies and Coverages

| Model | Precision. | % Improvement | % Coverage |
|---|---|---|---|
| Unigram-NOISE | 70.48 | 39.48 | 81.84 |
| Unigram | 69.52 | 37.58 | 65.56 |
| Bigram-NOISE | 70.58 | 39.68 | 81.83 |
| Bigram | **72.57** | **43.62** | 65.73 |
| **St. Onge** | 50.53 | - | $\approx 100.00$ |
| *Galley-McKeown* | 59.64 | - | - |

### 3.2.2 WSD Accuracy

This is the primary means of evaluating our chaining algorithms. We measure the precision of St. Onges algorithm and our algorithms on the task of WSD. We also present the respective values for coverage. The quantities are defined thus:

Let,

$c$ = no. of nouns diambiguated correctly,

$i$ = no. of nouns disambiguated incorrectly,

$n$ = no. of nouns not disambiguated

$$Precision = \frac{c}{c+i} \tag{3.18}$$

$$Coverage = \frac{c+i}{c+i+n} \tag{3.19}$$

Table 3.3 presents the data from our experiments.

For our algorithms the coverage is equal to proportion of words in the test set present in the training set. St. Onges algorithm has a coverage of close to 100% since it can disambiguate a word as along as it is present in Wordnet and we use a word in our experiment only when it has a valid Wordnet sense (Section 3.2.1).

The WSD was performed on 36 documents in the test set alone. *Unigram* and

*Bigram* denote the Unigram and Bigram models respectively. As a heuristic means to improve coverage, we had ran our algorithms with an additional state called *NOISE* that groups together all chains identified as noise by OPTICS. This was done for all the 16 clusterings we had obtained. *Unigram-NOISE* and *Bigram-NOISE* denote these runs. Percentage improvements in precision, with respect to St. Onges algorithm are also mentioned.

Only the best precision values are presented for a model across all clusterings.

We also present the value of precision as obtained by the algorithm presented in Galley and McKeown (2003). This paper reports values for an implementation of the algorithm that assigns as default the first sense in WordNet for non-disambiguated words. Since we explicitly take disambiguated words into account we reproduce the precision value from Stokes (2004) that reports performance of an implementation that does not impose tags on non-disambiguated words.

We mention the algorithm from Galley and McKeown (2003), since this is known to provide the best WSD accuracy amongst traditional chaining algorithms (Galley and McKeown (2003), Nelken and Shieber (2007)). However, given the fact that the algorithm is not run on our dataset, the reported precision may only be used for rough comparison against the other data.

Michelizzi (2005) also discusses some WSD results using the various relationships defined by the St.Onges' algorithm. However, the algorithm that employs these relationships is not a chaining algorithm, and is targeted towards achieving WSD alone. Michelizzi (2005) reports a F-score of 25%.

It is seen that our algorithms report similar values, with the bigram model reporting the highest score. All of our algorithms boast of a significant improvement over St. Onges algorithm. The best improvement we obtain is 43.62 %.

Before we conclude this section we would like to briefly discuss a discrepancy between the WSD accuracy for St. Onges algorithm we obtain and that reported in Stokes (2004). Table 3.4 presents the relevant data. As discussed in Section 2.2, we

Table 3.4: WSD accuracy - St. Onges' algorithm

| Source | Allowable Path Length | Recall % | Precision % | F1 % |
|---|---|---|---|---|
| Our implementation | 3 | 46.58 | 56.39 | 50.95 |
| Our implementation | 4 | 47.89 | 53.50 | 50.52 |
| Stokes (2004) | 4 | 56.92 | 59.45 | 58.20 |

do not use an allowable path length of 5. Here, precision is calculated as before. The quantities recall and F-score are defined thus:

$$Recall = \frac{c}{c + i + n} \tag{3.20}$$

$$F - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.21}$$

We attribute this discrepancy to certain differences between the way the experiments were performed. On correspondence with the author we discovered that her experiments differed from ours in the following ways:

- Semcor 1.6 was used
- The words used for chaining were the ones tagged with part-of-speech NN (Noun, singular or mass) only.
- A sample of 74 documents was used by the author. The values reported for our implementation was obtained by averaging over all 186 documents of subset of SemCor 3.0 we use.

Owing to certain computational and time constraints, we were unable to replicate her experiments.

The values of precision for St. Onges algorithm in Tables 3.4 and 3.3 slightly differ since they are reported for different sets of documents. In the former case, all 186 documents were used, whereas in the latter case, only the test set consisting of 36 documents was used.

Table 3.5: Average Processing Time per document

| Model | WSD acc. | Time Taken(sec) | % Improvement |
|---|---|---|---|
| Unigram-NOISE | 70.48 | 11.14 | 5.38x |
| Unigram | 69.52 | 5.89 | 10.19x |
| Bigram-NOISE | 70.58 | 1.42 | 42.25x |
| Bigram | 72.57 | **1.09** | 55.04x |
| **St. Onge** | 50.53 | $\sim 60$ | - |
| *Silber and McCoy (2002)* | 54.48 | 18 | - |

### 3.2.3 Processing Time

Our initial implementation of St.Onges algorithm did not involve any kind of caching of senses and relationships in memory. This naive implementation took roughly 5 hrs to find chains in a 500-noun document. We identified computation of a possible medium strong relationship on the fly between two words, to be a performance bottleneck - it requires conducting a breadth first search from one word, via only allowable paths, till either the other word is found or we have reached the maximum allowed path length. To avoid this overhead at runtime, we pre-computed all possible medium-strong relationships between pairs of nouns in Wordnet, up to a path length of 5, and stored them in a database. We used MySQL for the purpose, and the data was indexed with a B-Tree index. Our table consists of approximately 194 million entries. We modified our implementation to look-up medium strong relationships instead of computing them resulting in a processing time of roughly 1 min for a 500-noun document - an improvement of 300x.

We use this value of processing time as our benchmark. Table 3.5 presents processing times for various models. The processing time is the average time taken for processing one document, averaged over the 36 documents in the test set. For a model the processing time is presented only for the clustering that yielded the highest value of WSD accuracy. We reproduce these values from Table 3.3 for convenience.

We present the processing time and the precision of the algorithm described

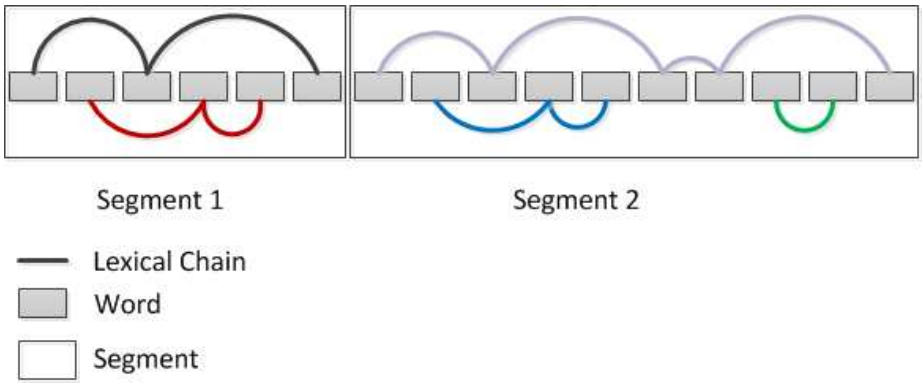


Figure 3.1: The relationship between lexical chains and segments

in Silber and McCoy (2002) since amongst traditional algorithms this is believed to be the fastest (Silber and McCoy (2002)). These values must be assumed to be rough estimates at best for the following reasons:

- The precision is reported from Galley and McKeown (2003). The performance is on a subset of SemCor 1.6, and the implementation imposes the first sense in WordNet as the default for non-disambiguated words (mentioned in Stokes (2004)). Since WordNet ranks senses of a word based on their frequency of occurrence in SemCor, this value is likely to be higher than the actual value for precision. We were unable to locate WSD accuracies for an implementation that does not impose a default sense.
- We were unable to obtain processing times on a dataset comparable to ours. Silber and McCoy (2002) mentions the algorithm takes 4 seconds on each of the documents used in Barzilay and Elhadad (1997). The latter paper uses a sample of documents from the TREC-3 dataset (Harman (1994)), with an average number of 30 sentences. Since the time complexity of the algorithm proposed in Silber and McCoy (2002) is O(n) where n is the number of nouns, and the average number of sentences in our test set is 139 sentences, we estimate its running time to be $\approx$ 18 seconds on a document like ours.

It is seen that processing times vary across models, with the bigram model yielding the best time. Chaining using the bigram model is about 55 times faster than our benchmark.

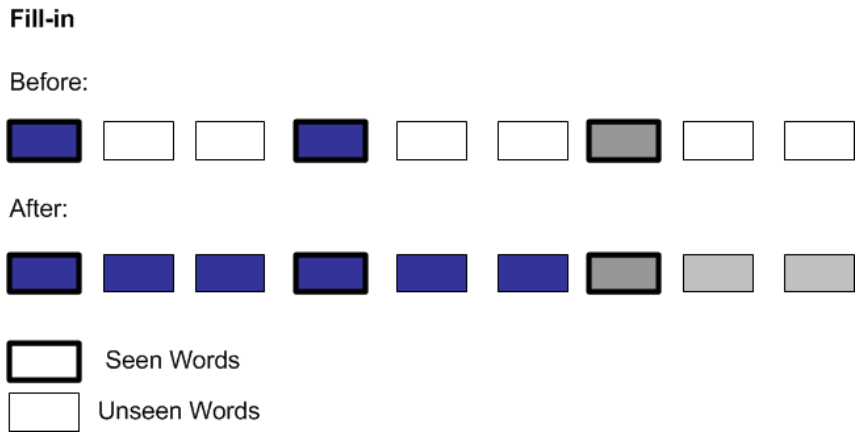


Figure 3.2: Fill-in

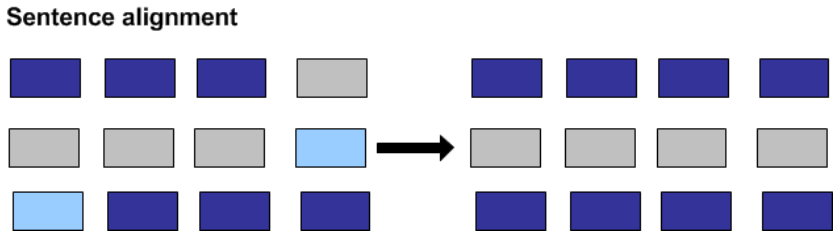


Figure 3.3: Sentence Boundary Alignment

### 3.2.4 Segmentation

We evaluate our algorithms on the task of *segmentation* too. We do not explicitly optimize our algorithms to perform well on this task. Our intent here is to ascertain that the improved WSD accuracies and lower processing times do not come at the cost of the ability to identify meaningful contexts. This is especially important in view of the low coverage our algorithms suffer from.

Segmentation is the task of dividing a document into topically coherent units called segments (Reynar (1998)).[2] The coherent units that we look for during segmentation are typically *larger* or more *inclusive* compared to the coherent regions chains identify. The output of a segmentation algorithm is ideally topics of discussion in text, as opposed to clusters of related words produced by chaining. Thus a chain can be seen as a "low-level" grouping of words which, typically, might not

[2]Literature on segmentation offers various definitions of a segment, depending upon the exact task at hand. This, however, best suits our purposes.

represent a topic. Topics may consist of more than one chain.

For example, a text discussing the right of a civilian to possess arms, is likely to have at two lexical chains - one, containing legal terms (ex. *law, rights, bill*), and the other composed of words related to arms (ex. *gun, revolver*). These chains define the contexts that the topic '*the right of a civilian to possess arms*' is composed of. The idea is illustrated in Figure 3.1. Two segments with multiple chains in each are shown in the diagram.

In actual text, the separation of topics is seldom clean with chain boundaries perfectly contained within segment boundaries. However, it has been observed that sentence boundaries, where a lot of chains end and new ones begin are good indicators of segment boundaries. The idea was originally mentioned in Morris and Hirst (1991). This intuition is captured in Stokes (2004) by calculating the sum of the number of chain endings and chain beginnings per sentence boundary and declaring it to be a segment boundary if this quantity is above a particular threshold $k$.

Before running the segmentation algorithm on our data we modify it with the following heuristics:

1. **Fill-in**: Since our coverage is low, it often becomes difficult to identify chain endings and beginnings around a sentence boundary; words around the boundary might not be present in the training set and thus not be assigned to any cluster. To workaround this problem, we impose cluster assignments on non-disambiguated words heuristically.

   Unseen words in a text are assigned the state of the last-seen word. If the first few words are unseen they are assigned the state of the first seen word. This helps in matching segmentation boundaries against the reference, when consecutive runs of nouns in sentences turn out to be unseen. Figure 3.2 illustrates the idea.

2. **Boundary alignment**: Since our algorithms do not take sentence boundaries into account while assigning clusters to words, words in the same sentence may get assigned to different clusters. The implication of this is unlikely; as contexts rarely change mid-sentence if at all. This heuristic explicitly aligns change of cluster assignments to sentence boundaries. Figure 3.3 illustrates the idea.

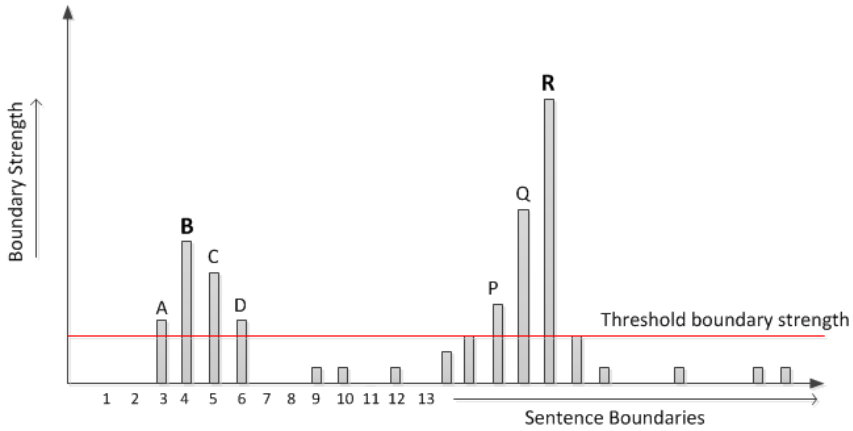


Figure 3.4: Minimum Segment Length

The modified data is segmented using the following algorithm (from Stokes (2004)):

1. Between each pair of sentences, determine the boundary strength: the sum of chain ends and chain beginnings at this point.
2. Find the average boundary strength, $b$, of the document. $b + k$ , where $k$ is a parameter, determines the threshold boundary strength. All points with a boundary strength value equal to or greater than this threshold is a potential topic boundary.
3. A parameter $d$ defines a minimum segmentation length - some of the potential topic boundaries, identified in the previous step, may be dropped if they are less than $d$ lexical units close to a higher scoring (with respect to boundary strength) potential boundary.

The parameter $d$ is used to avoid identification of very small segments. A segment boundary typically does not occur at one sentence boundary, but gradually over a few consecutive sentence boundaries. All these sentence boundaries have boundary strengths above the threshold, and would be normally identified as valid segment boundaries with each segment a sentence or so long. The parameter $d$ enforces a minimum segment length preventing identification of such boundaries. Figure 3.4 is a histogram of boundary strengths at various sentence boundaries. The red line denotes the threshold boundary strength which qualifies boundaries A, B, C, D, P, Q and R as valid segment boundaries. Enforcing a minimum segment

length of 2 we note that A, C, and D cannot be boundaries since within 2 units of each we have the higher scoring boundary B. Similarly, P and Q are filtered out owing to their proximity to R. This heuristic leaves us with boundaries B and R only.

The segmentation accuracies are reported using F-scores. Although metrics such as $P_k$, introduced in Choi (2000), and WindowDiff, proposed in Pevzner and Hearst (2002), are often used for calculating segmentation accuracies, they suffer from certain drawbacks.We do not discuss the metrics or their shortcomings here. Pevzner and Hearst (2002) discusses the shortcomings of the $P_k$ measure. WindowDiff, proposed in the same paper, remedies most deficiencies of the $P_k$ measure, but its interpretation is often difficult.

We calculate F-scores to measure exact matches as well as near misses. We do this by separately calculating F-scores for the following scenarios:

1. There is an exact match between reference and hypothesized boundaries
2. The reference and hypothesized boundaries are off by 1 sentence
3. They are off by 2 sentences

Scenarios 2 and 3 take into account near misses by assuming a prediction of a boundary is correct if it is within 1 or 2 sentences of the actual boundary. We define the F-score as:

$$F = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

where,

$$Recall = \frac{\text{no. of predicted boundaries matched with actual boundaries}}{\text{total no. of actual boundaries}}$$

$$Precision = \frac{\text{no. of predicted boundaries matched with actual boundaries}}{\text{total no. of predicted boundaries}}$$

Here, when we say the predicted and actual boundaries *match*, depending on the scenario being evaluated we imply that a boundary is predicted for an actual boundary at exactly where it is located, or within 1 or 2 sentences of it.

Figure 3.5 illustrates some actual and predicted segment boundaries. For scenario 1,

no. of predicted boundaries exactly matched with actual boundaries = 0
total no. of actual boundaries = 3
total no. of predicted boundaries = 2

Hence,

Recall = $\frac{0}{3} = 0$
Precision = $\frac{0}{2} = 0$
and F-score is defined to be 0.

When we accept a match which is off by a sentence (scenario 2), the various values are:

no. of predicted boundaries exactly matched with actual boundaries = 1
total no. of actual boundaries = 3
total no. of predicted boundaries = 2

Recall = $\frac{1}{3} = 0.33$
Precision = $\frac{1}{2} = 0.50$
F-score = $\frac{2 \times 0.33 \times 0.5}{0.33+0.5} = 0.40$

We use synthetically created documents for evaluating segmentation. This technique was originally described in Choi (2000). A synthetic document is formed by concatenating excerpts from various documents, the points of concatenation providing us with benchmark segmentation boundaries. Each excerpt is the first $n$ sentences of a document where $n$ can belong to the following ranges: (3,5), (3,11),

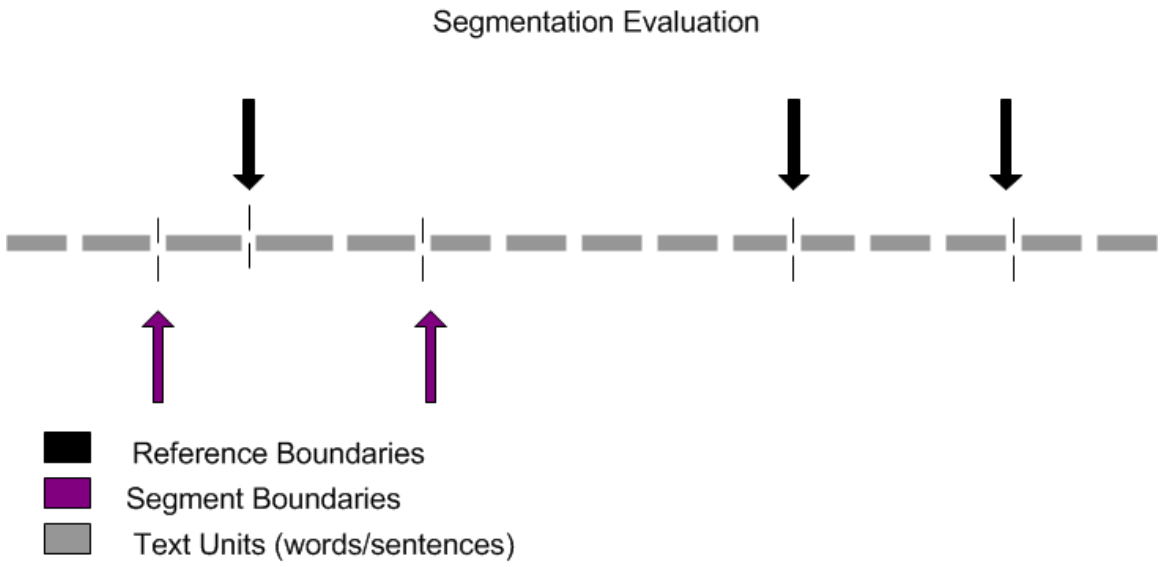


Figure 3.5: Measuring segmentation accuracy

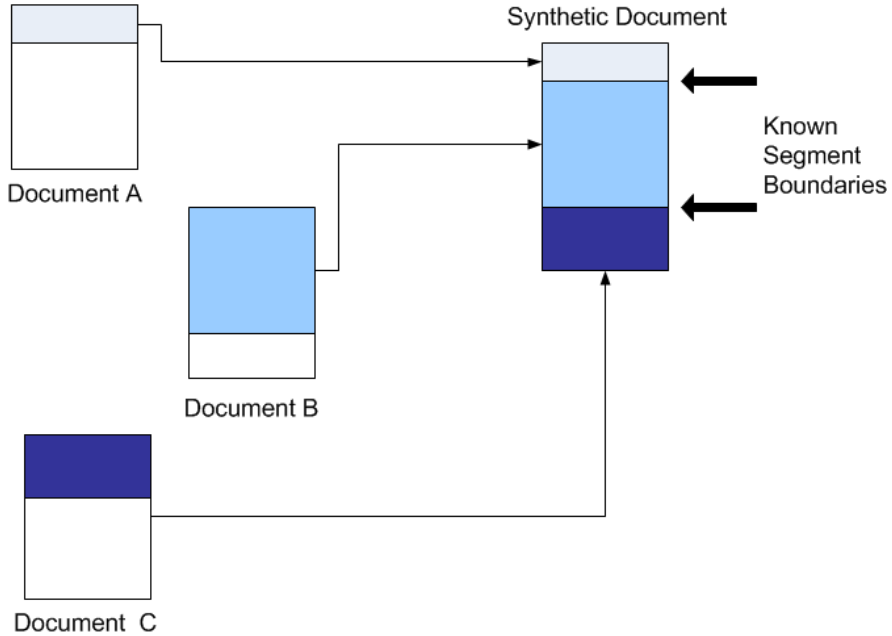


Figure 3.6: Synthetic Documents

Table 3.6: Segmentation Accuracies - Supervised Models

| Model | Subtask=0 | 1 | 2 |
|---|---|---|---|
| Unigram-NOISE | F1 = 0.12 | 0.4 | 0.51 |
| Unigram | **0.18** | **0.52** | **0.69** |
| Bigram-NOISE | 0.13 | 0.4 | 0.6 |
| Bigram | 0.09 | 0.3 | 0.43 |
| **St. Onge** | 0.17 | 0.47 | 0.64 |

(6,8), (9,11). A synthetic document consists of 10 such segments. Figure 3.6 shows the process.

The segmentation parameters, $d$ and $k$, were obtained by 5-fold cross validation on the training set. The training set was divided into 5 folds, and for each fold a synthetic dataset of 500 documents was created from the documents in the fold. The chains from the original documents in the fold were used to obtain a clustering, with various parameter settings. The parameter settings are almost similar to ones used for WSD, except that we do not use a non-trivial chain length of 3 (the high number of chains renders the clustering computationally expensive). We, then, evaluate the segmentation accuracy for a fold, against its synthetic dataset, for the various parameter settings of the clustering and various settings of segmentation parameters ($d$, and $k$), and store the top 10 segmentation parameter settings. This is repeated for each of the 3 segmentation subtasks we are interested in (exact boundary match, near misses by 1 or 2 sentences). $d$ is assigned integral values in the range [0,8], while $k$ is varied in the range [0,5].

We don't report processing times for segmentation since the time taken to identify boundaries is very small compared to chaining. Also, the algorithm to identify boundaries is common to all the chaining algorithms considered.

Table 3.6 presents our observations on this task. We note that the Unigram model significantly outperforms other models.

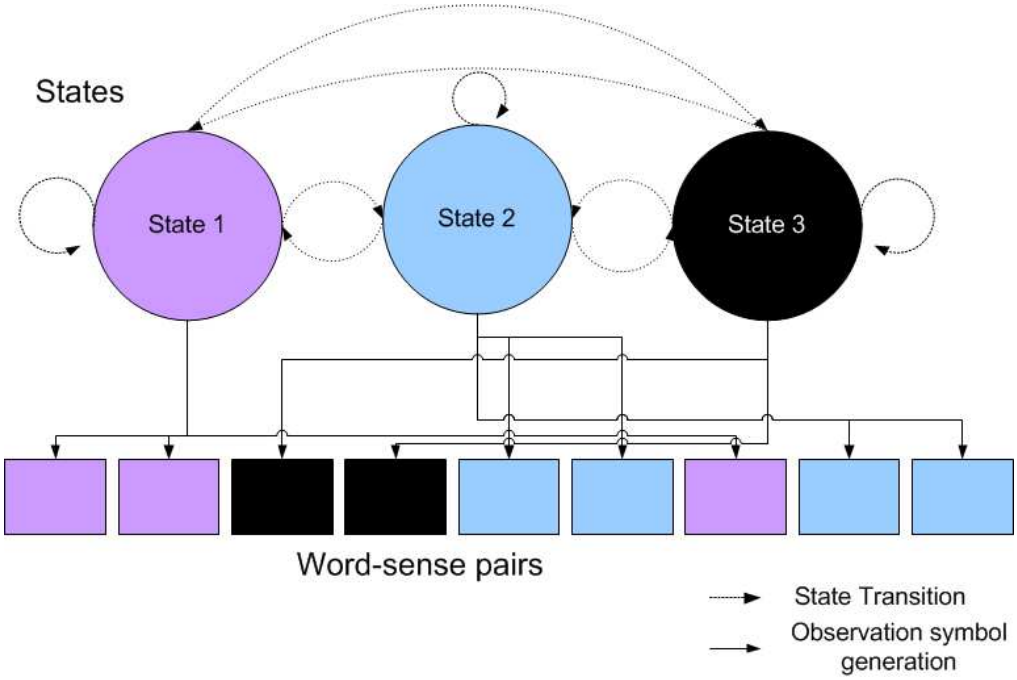


Figure 3.7: Schematic of models. The dotted lines exist only in case of the bigram model.

## 3.3 Summary

This chapter looked at the models we use and the corresponding algorithms. We saw that our models differ in the assumptions they make regarding distribution of contexts in a document. In case of our first model, the Unigram model, we assume that the context a word represents is not influenced by contexts represented by words around it. Our second model, the bigram model, makes the assumption that the context of word is influenced by the context of the word before it.

The models may be succinctly depicted using the schematic of Figure 3.7. Observation symbols and clusters are denoted by rectangles and clusters respectively. Dotted lines indicate transition probabilities and solid lines represent the association between an observation symbol and a cluster. The transitions between states are accounted for only in the bigram model.

We have also presented a comparison of our algorithms against St. Onges algorithm on the tasks of WSD and Segmentation, in addition to comparing their

running times. The data makes it evident that our algorithms out perform St. Onges algorithm on variety of metrics for these tasks. Data from the literature pertaining to certain other algorithms have also been presented for a rough comparison against these algorithms.

We present a much more complete discussion of the conclusions from our research in the next chapter.

# CHAPTER 4

# Conclusions and Future Work

In this chapter we present conclusions from our research in terms of the advantages our algorithms provide over existing chaining algorithms and disadvantages associated with their use. We also discuss some possibilities for future research that can build upon our work.

## 4.1 Advantages provided by our algorithms

1. Greater accuracy

   Results presented in Table 3.3, Section 3.2.2 clearly demonstrate that our algorithms do much better than St Onges algorithm in terms of WSD. The bigram model that discounts noise performs best, achieving a 44% improvement over our benchmark algorithm. It also produces better results than the algorithm proposed by Galley and McKeown (2003), which has the highest WSD accuracy amongst the classical algorithms.

   Since WSD accuracy measures the quality of chains produced by a chaining algorithm, we expect our algorithms to do much better on tasks that use them.

2. Faster running times

   Table 3.5 from Section 3.2.3 shows that using the bigram model that discounts noise, our execution is about 55 times faster than our benchmark algorithm. It may be argued that the faster execution is a result of a lower coverage of 65%. But even so, given that the time complexity of using the bigram model is $O(nk^2)$ and hence linear in the number of words, a reasonable guess for the running time for 100% coverage would be: $1.09 \times 100/65 = 1.7$ seconds. This is still 36 times faster than our benchmark.

   While fast running times are desirable in general, the order of the running times we obtain open up the possibility of using chaining in real-time systems. For example a possible application might be in intelligent news aggregators which need to continuously assimilate new or updated articles from their sources, process them and make them available to a user as soon as possible.

Amongst our models, the bigram model seems to describe text better.

It might be noted that our approach is especially valuable on account of the fact it is *both* faster and more accurate compared to classical algorithms, most of which achieve only one of these goals.

Although our experiments specifically use St. Onges algorithm, our approach is general enough to be adapted to other chaining algorithms. A different algorithm only affects the creation of training data - the STLC needs to be performed with this algorithm; other steps remain the same.

## 4.2 Disadvantages of using our algorithms

1. Low Coverage

   Since our algorithms are supervised they can only process words that have been seen in the training data. The size of the training set we use leads to a low coverage of 65%. Considering the fact that we only process nouns to begin with, which form 20-25% of an average text, decreased coverage is a particularly expensive trade-off. We discuss a possible remedy in the next section (item 4 in the list presented in Section 4.3).

2. Requirement of training data

   In contrast to classical algorithms, our algorithms need sense-tagged texts as training data. This can serve both as a disadvantage and an advantage.

   This is a disadvantage because the requirement constrains the use of our algorithms to the availability of such data, additionally making the accuracy dependent on the volume of such data available.

   It is an advantage in cases where a domain sees two words as more similar (or less similar) than the Wordnet hierarchy presupposes. In such a case, classical algorithms ignore the specifics of a domain as there is no way to incorporate such information into them. This indirectly affects the WSD accuracy. Sufficient training data would sensitize our algorithms to recognize such domain-dependent semantics.

## 4.3 Future work

We list below some possible areas of future research that may extend our work. We feel these are promising directions to work towards for improving our algorithms further.

1. Further exploration of the various parameter spaces is desirable. This includes the following parameter settings:
   (a) The parameters $\alpha$ and $\beta$ in our similarity function
   (b) Values of reachability distance used
   (c) Segmentation parameters $d$ and $k$

2. Our algorithms look only at nouns. This is a characteristic of most classical algorithms too. It might be of interest to look at how words belonging to other parts-of-speech may be analysed and used to identify contexts and context boundaries.

3. Some heuristics may be directly incorporated into the model. For example instead of forcing topic transitions to happen only at sentence boundaries as a heuristic (Section 3.2.4), this fact may be encoded in the models themselves; we could disallow assignment to more than one cluster for words in the same sentence.

4. The issue of low coverage might be addressed by modifying our algorithms to "switch" to normal chaining for unseen words. Such an algorithm could work in two phases: in the first, it will run our current algorithms as they are, to chain only words seen in the training data. In the second phase, it would start with the chains formed in the first phase, and chain unseen words using Wordnet, much like a classical algorithm.

   The second phase, thus, is guided by the chains formed in the first phase. We use our algorithms in the first phase since they offer higher WSD accuracy.

# REFERENCES


**Al-halimi, R.** and **R. Kazman** (1998). Temporal indexing through lexical chaining.

**Ankerst, M.**, **M. M. Breunig**, **H.-P. Kriegel**, and **J. Sander**, Optics: ordering points to identify the clustering structure. *In Proceedings of the 1999 ACM SIGMOD International conference on Management of data*, SIGMOD '99. ACM, New York, NY, USA, 1999. ISBN 1-58113-084-8. URL `http://doi.acm.org/10.1145/304182.304187`.

**Barzilay, R.** and **M. Elhadad**, Using lexical chains for text summarization. *In Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*. 1997.

**Baum, L.** and **T. Petrie** (1966). Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, **37**, 1554–1563.

**Baum, L.**, **T. Petrie**, **G. Soules**, and **N. Weiss** (1970). A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Annals of Mathematical Statistics*, **41**, 164–171.

**Baum, L.** and **G. Sell** (1968). Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, **27**, 211–227.

**Baum, L. E.** (1972). An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, **3**(1), 1–8. URL `http://research.microsoft.com/apps/pubs/default.aspx?id=64753`.

**Baum, L. E.** and **J. A. Eagon** (1967). An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. *Bulletin of The American Mathematical Society*, **73**, 360–364.

**Bird, E. L., Steven** and **E. Klein**, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

**Choi, F. Y. Y.**, Advances in domain independent linear text segmentation. *In Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.

**Deerwester, S.**, **S. T. Dumais**, **G. W. Furnas**, **T. K. Landauer**, and **R. Harshman** (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, **41**(6), 391–407.

**Ester, M.**, **H. peter Kriegel**, **J. S**, and **X. Xu**, A density-based algorithm for discovering clusters in large spatial databases with noise. AAAI Press, 1996.

**Fellbaum, C.**, *WordNet: An Electronical Lexical Database*. The MIT Press, Cambridge, MA, 1998.

**Gale, W. A.**, **K. W. Church**, and **D. Yarowsky**, One sense per discourse. *In Proceedings of the workshop on Speech and Natural Language*, HLT '91. Association for Computational Linguistics, Stroudsburg, PA, USA, 1992. ISBN 1-55860-272-0. URL `http://dx.doi.org/10.3115/1075527.1075579`.

**Galley, M.** and **K. McKeown**, Improving word sense disambiguation in lexical chaining. *In Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI–03)*. 2003. URL `http://www.cs.columbia.edu/nlp/papers/2003/galley_mckeown_03.pdf`.

**Green, S. J.**, Using lexical chains to build hypertext links in newspaper articles. *In AAAI 96 Workshop on Internet-based Information Systems*. 1996.

**Green, S. J.** (1997). *Automatically Generating Hypertext By Computing Semantic Similarity*. Ph.D. thesis, University Of Toronto, Canada.

**Hall, M.**, **E. Frank**, **G. Holmes**, **B. Pfahringer**, **P. Reutemann**, and **I. H. Witten** (2009). The weka data mining software: an update. *SIGKDD Explorations Newsletter*, **11**, 10–18. ISSN 1931-0145. URL `http://doi.acm.org/10.1145/1656274.1656278`.

**Harman, D.**, Overview of the third text retrieval conference (trec-3). *In TREC'94*. 1994.

**Jones, K. S.** (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, **28**, 11–21.

**Jurafsky, D.** and **J. H. Martin**, *Speech and Language Processing (2nd Edition)*. Prentice Hall, 2008, 2 edition. ISBN 0131873210.

**Kirkpatrick, S.**, **C. D. Gelatt**, and **M. P. Vecchi** (1983). Optimization by simulated annealing. *Science*, **220**(4598), 671–680. URL `http://www.sciencemag.org/content/220/4598/671.abstract`.

**Loper, E.** and **S. Bird**, Nltk: The natural language toolkit. *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*. 2002.

**Marcus, M. P.**, **B. Santorini**, and **M. A. Marcinkiewicz** (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, **19**(2), 313–330.

**Michelizzi, J.** (2005). *Semantic Relatedness Applied to All Words Sense Disambiguation*. Ph.D. thesis, University Of Minnesota.

**Miller, G. A.** (1995). Wordnet: A lexical database for english. *Communications of the ACM*, **38**, 39–41.

**Morris, J.** and **G. Hirst** (1991). Lexical cohesion computed by thesaural relations as an indicator of the structure of text. *Computational Linguistics*, **17**(1), 21–48.

**Nahnsen, T.**, **O. Uzuner**, and **B. Katz** (2005). Lexical chains and sliding locality windows in content-based text similarity detection. Technical report, CSAIL Technical Report, MIT.

**Nelken, R.** and **S. M. Shieber** (2007). Lexical chaining and word-sense-disambiguation. Technical Report TR-06-07, School of Engineering and Applied Sciences, Harvard University, Cambridge, MA.

**Pevzner, L.** and **M. A. Hearst** (2002). A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, **28**(1), 19–36. ISSN 0891-2017.

**Rabiner, L. R.**, A tutorial on hidden markov models and selected applications in speech recognition. *In Proceedings of the IEEE*. 1989.

**Reynar, J. C.** (1998). *Topic Segmentation: Algorithms and Applications*. Ph.D. thesis, University Of Pennsylvania.

**Silber, H. G.** and **K. F. McCoy** (2002). Efficiently computed lexical chains as an intermediate representation for automatic text summarization. *Computational Linguistics*, **28**, 487–496. ISSN 0891-2017. URL `http://dx.doi.org/10.1162/089120102762671954`.

**St-Onge, D.** (1995). *Detecting and Correcting Malapropisms with Lexical Chains*. Ph.D. thesis, University Of Toronto, Canada.

**Stairmand, M. A.**, Textual context analysis for information retrieval. *In SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, New York, NY, USA, 1997. ISBN 0-89791-836-3.

**Stokes, N.** (2004). *Applications of Lexical Cohesion Analysis in the Topic Detection and Tracking Domain*. Ph.D. thesis, National University Of Ireland, Dublin.

**Utiyama, M.** and **H. Isahara**, A statistical model for domain-independent text segmentation. *In Proceedings of the 39th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2001.