Leibniz Universität Hannover
Fakultät für Elektrotechnik
und Informatik
Institut für verteilte Systeme
Fachgebiet Wissensbasierte Systeme

A Comparison of Graph Matching Algorithms for Cluster Tracking

Author:
Zhivko Asenov

Examiner: **Prof. Dr. techn. Wolfgang Nejdl**
Second examiner: **Prof. Dr. Nicola Henze**

Supervisors:
Nina Tahmasebi
Dr. Thomas Risse

August 17, 2011

Abstract

In this master thesis we investigate the cluster tracking problem and present applicable algorithms for discovering changes in the word senses over time. To solve this problem we have developed methods for computing similarities between clusters representing word senses. We propose algorithms based on previously established approaches for measuring similarities such as Jaccard index and cosine similarity. We also use data structures for abstracting information such as feature vectors and centroids.
We test our approaches on clusters extracted from co-occurrence graphs where each graph represents words used in newspaper articles for a particular year. The words in the clusters represent a word sense. We have created a test set of manually evaluated cluster pairs and use it as a ground truth for our experiments. We use Jaccard similarity as a baseline and we find that in most cases the algorithms performed better.

Contents

List of Figures

Chapter 1

Introduction

The interpretation of the context of the words is a task that is feasible mostly by humans. To automatically interpret the meanings of words can be useful for solving problems in the natural language processing and semantic web domains.
Let us consider the following situation; a user needs information about the word **air plane** and is interested in articles about **air planes** as military machines. In a reply to this query, a modern search engine will deliver a big number of results about travelling and tourism. The results reflect the context of this word that is different compared to the meaning it had some decades ago. This phenomenon is typical, especially for the everyday language – words change their meaning.
In order to automatically track the changes of words meaning, we need to map words from texts to some flexible data structure. For this purpose, we can use a graph. We can take words extracted from digitalised texts and put them in a graph where each word is a **vertex** in the graph and the links between words are **edges**. A link between two words describes that these words fit to some pattern. A common pattern could be when we group words together according to their positioning in texts (e.g words separated by a comma or an ”and”). The next step will be to nest smaller sets of words which have a mutual meaning.

These sets are called **clusters**. Figure 1.1 gives us an example of a word graph clustered into different word meanings.

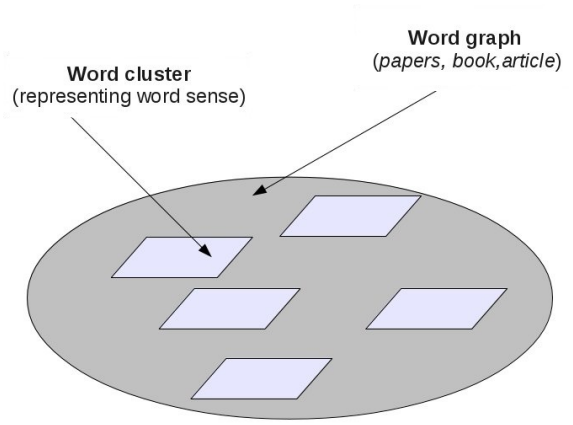


Figure 1.1: Word graph clustered into different words meanings

The main goal of this master thesis is to find methods for tracking word senses over time and to determine if the meaning of words have changed. This is achieved by investigating different solutions to the *cluster tracking problem*. More specifically, we are interested in graph matching algorithms which can provide sufficient and computationally efficient results. We will concentrate on everyday English language used in newspapers over long time periods as a starting point. It is left to the future work to extend these methods also to more domain specific language.

As we can see in Figure 1.2, we want to algorithmically compute similarities between clusters for each pair of graphs. The graphs represent collection of texts for a particular time period. The scenario that we follow is that we get clusters from different periods and start comparing their graph structures in order to detect similarities. We start by filtering clusters based on the Jaccard similarity [**?**] between the members in these clusters.

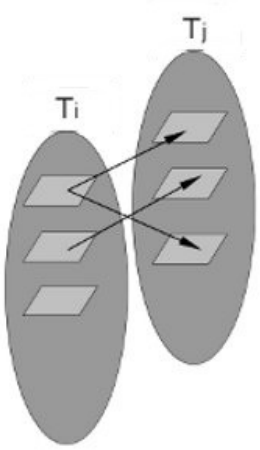


Figure 1.2: Word graphs from two different time periods **T**i and **T**j where each square represents one cluster

In this thesis, we will describe three further algorithms for computing cluster similarities. We analyse the results and draw conclusions. The estimated values delivered by the algorithms are verified against a test set of clusters evaluated by people with different professions and from different backgrounds.

Outline Of The Thesis

The rest of the thesis is organised as follows:

Chapter 2 provides a literature review, researched during the preparation phase of this thesis and outlines which ideas we considered for our work and how they contributed to the thesis.

Chapter 3 describes the theoretical models and the basic definitions used in this thesis. In this chapter, we give an overview of the basic algorithms and clarify the main characteristics that we took into consideration while developing the algorithms.

Chapter 4 introduces detailed overview of the applied algorithms.

Chapter 5 discusses the conducted experiments and shows the achieved results. The quality of the algorithms is assessed based on human evaluations.

Chapter 6 provides a summary of the thesis. At the end, we propose some ideas for possible future development of the already implemented ideas and we draw a conclusion.

Chapter 2

Related work

In this chapter, we will present different approaches of solving graph and cluster problems and we will discuss some aspects of them.

An interesting approach is introduced by Palla et al. [**?**], where an interactions between complex community structures are observed. Two networks are created that map two domains of interest. One network is built on the co-authorship of the Cornell University Library article archive (over 30000 authors, spanning for 142 months). The other network represents the phone calls of mobile phone company for 52 weeks. These two networks exhibit different characteristics. The articles network has bigger density and the groups representing author-paper relation are often overlapping. On the other hand, the phone calls network shows less group interconnectivity and the single communities are scattered over the network. Firstly, the authors extracted communities for each time step using the clique percolation method (CPM). CPM defines a community as a union of all sub-graphs with size $\boldsymbol{k}$ that can be reached from each other using a series of adjacent sub-graphs. The communities members can be reached through connected graphs and the communities may overlap.

Palla et al. were able to make some conclusions about the network specifics while investigating the communities obtained by CPM. By comparing the average weight of the inside links W_c to the average weight of the inter-community links W_{ic} , the authors verified that the interaction between members within different groups is less intensive. Next, a classification of the possibly occurring events in a community was made. A community can grow or shrink (resolving in group merging or splitting), new communities can be formed or can disappear. Palla et al. develop a method to detect the changes occurring while the above described events take place. The method has the following steps:

- determine the community size s and age τ
- using the auto-correlation function C(t), to compute the relative overlap between community A(t) in step t_0 and step $t_0 + t$

$$C(t) \equiv \frac{|A(t_0) \bigcap A(t_0 + t)|}{|A(t_0) \bigcup A(t_0 + t)|}$$

where $|A(t_0) \bigcap A(t_0 + t)|$ is the number of common members in $A(t_0)$ and $A(t_0 + t)$ and $|A(t_0) \bigcup A(t_0 + t)|$ is the union of the members for the same time steps.

Based on these data, the authors are able to conclude that larger communities are changing faster than the smaller ones. Another observation is that larger communities have an extended lifetime only when they are dynamic, whereas small communities with a static structure have a longer existence.
Mei and Zhai proposed a solution for discovering evolutionary patterns of themes in a text stream in their paper [**?**]. They focused their work on documents that have a meaningful time stamp (news articles and scientific literature). The authors developed methods for discovering and summarising, so called *evolutionary theme patterns* (ETP) which basically represent the evolution of subtopics in text stream. The subtopics are covering some event that has

> "an underlying temporal and evolutionary structure." [**?**]

Mei and Zhai divided their solution in three main parts:

- discover latent themes from text
- discover evolutionary theme relations and construct an evolution graph of themes
- analyse life cycles of themes

Two data sets were used – collection of news articles covering the tsunami event (19.12.2004-08.02.2005) and abstracts of the ACM KDD (Knowledge Discovery and Data Mining) conference papers from 1999-2004. The authors formulated the following definitions used in the paper:

- **theme** - a probabilistic distribution of words characterising topic or subtopic
- **theme span** γ is the spanning interval l of theme θ , where l is defined for start and terminating time stamps $(s(\gamma), t(\gamma))$

- **evolutionary transition** - describes an evolution of theme $\gamma_1 = \langle \theta_1, s(\gamma_1), t(\gamma_1) \rangle$ to $\gamma_2 = \langle \theta_2, s(\gamma_2), t(\gamma_2) \rangle$, denoted by $\gamma_1 \prec \gamma_2$
- **theme evolution thread** - a sequence of theme spans that evolve from one to another.
- **theme evolution graph** - weighted directed graph G=(N,E) where a **vertex** is a *theme span*, an **edge** is an *evolutionary transition* and the weight represents the evolution distance. Each path in the graph stands for a *theme evolution thread*.

The authors point out that the major task is the automatic extraction of theme evolution graph from a text collection. This graph is used to organise and summarise the topics and subtopics of the texts in a meaningful way. In the graph the evolutions of themes are visualised for short distance with thin and for longer distance with bold links. Another task is to measure the strength of the themes over time - starting and terminating time are computed and traced. The theme extraction is done by using a probabilistic mixture model where

> "words are regarded as data drawn from mixture model with component models for the theme word distributions and a background word distribution. Words in the same document share the same mixing weights." [**?**]

To compute the model is used the Expectation Maximization(EM). The algorithms computing the model can estimate a local maximum of likelihood for the themes.
Afterwards, evolutionary transitions are discovered using the Kullback-Leibler [**?**] divergence that measures distances between probability distributions. By setting a proper threshold it is possible to control the strength of the themes transitions. Having the theme spans extracted the authors build the evolution graph.

> "The theme evolution graph gives us a microcosmic view of the ETPs - revealing the major theme spans within each time interval and their evolutionary structures." [**?**]

In order to track the themes over a time period the authors define a **theme life cycle**. A theme life cycle is defined as the number of time periods in which the theme was used to generate words. With the help of the Hidden Markov Model(HMM) the interactions between trans-collection themes in the whole document collection are modelled and decoded. A HMM basically describes a probabilistic model for a sequence of states $S = (s_1, s_2, ...)$ that

have a predefined output values $O = (o_1, o_2, ...)$. The output values and the states have also probability distributions.
The actual modelling of the themes is done as follows: the document collection is represented by a long sequence of words generated by the HMM. K trans-collection themes are extracted from the collection by the mixture model and a fully connected HMM with k+1 states is constructed. Applying this model, the authors can decode the whole collection of documents with labels of themes and compute the strengths of the themes at a time period. With the help of these computations Mei and Zhai were able to track the changes in the life cycle of the themes. The authors evaluated their methods with data sets based on the news articles and they performed two experiments: partitioning the collection into time periods, discovering the theme evolution graph and the trans-collection themes and analysing the life cycles. The authors were able to correctly extract many topics and subtopics and to trace different theme threads for many articles. The different threads exhibit interesting patterns in their evolution and behaviour over the observed time interval.

Another interesting solution concentrated on temporally versioned document collections is proposed by Berberich et. al. in [**?**]. The authors are addressing the problem of *time-travel text search* - searching a collection of documents which have additional time information. They introduce a method using the inverted file index as base line and improve its results by applying an approximate temporal coalescing to reduce the index size. The basic idea of the authors is to take a normal keyword query and to extend it with the ability to "understand" time stamps for each search. The data model is built as follows: a collection of versioned documents **D** where every document **d** contains a sequence of its time versions $d^{t_1}, d^{t_2}, d^{t_3}, ...$ Each version is a vector of searchable terms or features. Also a validity is defined $\text{val}(d^{t_i})$ giving the latest version of a document. The importance of a document version to a given search query q^{t_i} depends on the plain term frequency of the terms in d_i normalised by the length of the version and the average length of the document itself. An inverted file index is basically mapping all terms to their idf-score and inverted list. There is an information about every term describing its position in the document and its frequency. The inverted file index is extended with additional information about the validity of the terms meta data.
The query process is organised as follows:

> "... for each query term the corresponding idf-score valid at the time t is retrieved from the extended vocabulary. Then index lists are sequentially read from disk, thereby accumulating the information

contained in the postings." [?]

In order to optimise the problem with the index size, Berberich et.al use the approximate temporal coalescing method. The main idea here is that the changes in documents are minor and big parts of them stayed unchanged. For adjacent versions of the document the meta data does not change at all. The proposed methods are tested on the English Wikipedia revision history (01.2001-12.2005) with 892,255 documents having around 14 Million versions and on the European Archive *.gov.uk* collection containing weekly crawls having around 500,000 document with approximately 8.7 Million versions. The authors were able to reduce significantly the index size for the both datasets and their methods provided high accurate results. Berberich et.al. had multiple runs with different temporal coalescing parameters without affecting the query results.

Papadimitriou et. al. proposed in their paper [?] methods for detecting anomalies in a web graph. A web graph is part of the offline component of a search engine where all downloaded web pages and their links are organised and mapped to the graph. For guaranteeing an accurate online searches the web graphs, covering web page informations for an enormous amount of sites and for relatively long periods of time, need to be automatically checked for missing information. This can be achieved when the web graphs from every snapshot are compared to each other and significant difference are detected. To solve this problem the authors analyse several existing methods for computing similarities between graphs and apply them in order to detect changes in the graph structure such as:

- *missing connected graph* (e.g. a host is missing while data was gathered)
- *missing random vertices* (e.g. disk failure in web graph storage machines)
- *random topological changes* (e.g. bugs in web graph data management software)

Papadimitriou et. al investigate the following methods:

- vertex/edge overlap - Jaccard index between vertex or/and edge sets
- vector similarity - distance of adjacency matrices principal eigen-vectors
- vertex ranking - rank correlation between sorted(by Pagerank) vertex lists
- sequence similarity - convert graphs to vertex sequence and calculate ratio of common subsequence

- signature similarity - get graphs fingerprints using LSH and calculate Hamming distance-based similarity

For every snapshot similarity between graphs are to be computed with each of the methods. If the similarity is less than a specially computed threshold, then an anomaly is suspected.
The evaluation of the methods showed that all of them successfully detect missing sub-graphs , three of them(*vertex/edge overlap*,*vector similarity* and *signature similarity*) detect missing vertices. *Vertices similarity* and *signature similarity* were able to detect all anomalies.

Yiling and Fonseca introduced in their paper [?] a solution of an interesting problem in the semantic web domain. They proposed method that maps semantic concepts between ontologies. Concepts from two ontologies are grouped in clusters. Each cluster is composed of concepts from the same ontology. The basic idea of the proposed method is to locate clusters of concepts for a pair of ontologies that are similar to each other. This problem is modelled as a weighted bipartite graph partitioning problem.

> "A bipartite graph has two disjoint vertex sets. Co-clustering a bipartite graphs is to simultaneously group several vertices into similar clusters for each vertex set." [?]

In such a way it is possible to build many-to-many mappings between concepts clusters. The authors take two different ontologies about the same subject and model them as weighted bipartite graph. The graph has two vertex sets where each set represents the concepts nodes of each ontology. The graph edges connect nodes from one set to another. Concepts from the same ontology are not connected. The weight of an edge is the similarity between two concepts (only weights above some precomputed threshold are presented in the graph).
After the graph is built a co-clustering method is applied on the bipartite graph. It results in cutting through the vertex sets of the graph.

> "The cut of a partition is defined as the sum of weights of those edges that are "broken" in the partitions." [?]

The partition forms two new bipartite graphs. Every sub-graph has again vertex sets representing concepts cluster with same features as before except the missing connection with the other cluster pairs. The similarity between two concept clusters can be measured as a sum of all weights for all edges connecting the two clusters.

Yiling and Fonseca investigated an interesting approach for modelling an ontology as bipartite graph and using its special structure for clustering concepts and computing similarity between the cluster pairs.

There are many other papers which were analysed during the preparation phase of this master thesis. Another group of papers dealing with experiments on graphs with algorithms based on the *Jaccard index*, *graph edit distance*, *cosine similarity*, *dice similarity*, were not discussed here. This literature review mentioned some of the papers which have challenging problems and complex ideas. The described methods were not applied in the algorithmic part of this thesis because for the most of them, the data model needed to be built in a special way. Using these methods would require to change our data model which is beyond the scope of this master thesis.

Chapter 3

Definitions

For each problem that needs to be solved by computers, we need to build an abstraction of the "world" representing the problem and to map it to some data model. So, we can analyse important features of the problem and propose solutions.
The data model that we use has a graph-based structure and the methods that we are going to introduce use this graph structure. This chapter describes definitions of the graph theory and some theoretical backgrounds considering similarity algorithms. We will give also some definitions of vector space specific structures. They can be used for abstracting feature information and comparing clusters.

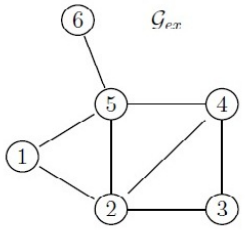
3.1 Graph And Graph Based Structures

A graph is flexible data model that allows us to describe any objects having interactions with each other.

Definition 3.1 - Graph

An **undirected graph G** consists of two components – a set of *vertices* **V** and a set of unordered pairs of *vertices* **E** called *edges*. A common notation is : G=(V,E) where **E** is a subset of all possible combinations of two vertices **VxV**.

A graph can be also represented as an adjacency matrix. The rows and the columns of the matrix are labelled with the graph vertices. If two vertices are connected, a value of 1 is put in the matrix, otherwise – 0 [**?**]. Figure 3.1 gives an example of a graph and its adjacency matrix.



$A_{\mathcal{G}_{ex}}$

	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	1	1	0
3	0	1	0	1	0	0
4	0	1	1	0	1	0
5	1	1	0	1	0	1
6	0	0	0	0	1	0

Figure 3.1: Graph – represented graphically and as an adjacency matrix [**?**]

Definition 3.2 - Sub-graph

A graph $G^{'}$ is a `sub-graph` of **G** if $V^{'}$ and $E^{'}$ are subsets of **V** and **E**. A **sub-graph** consists of a sub-set of vertices and edges which are presented in another graph.

Definition 3.3 - Path, Neighbourhood and Degree

A sequence of consecutive edges $e_1, e_2, e_3... \in V(G)$ in a graph **G** is called a **path**. Every vertex belonging to the edges contained in the sequence can be reached.
The number of vertices building the **path** is called *length* of a **path**. A **neighbourhood** $N_g(v)$ is the set of all vertices that have direct connection to the vertex **v** (share an edge).
A **degree** describes how "extended" a **neighbourhood** can be. **Degree** of

1 covers the basic definition of a neighbourhood - only the direct connections of a particular vertex to other vertices are observed. A higher value of a **degree** describes the maximum `length` of a **path** from a particular vertex **v** to its furthest neighbour.

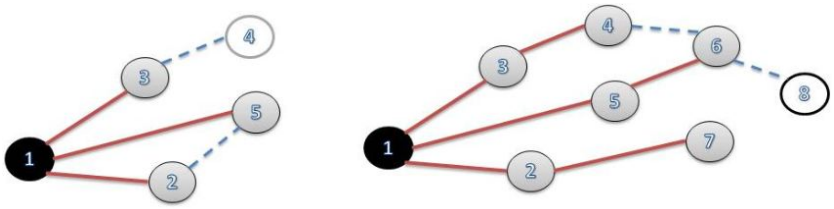


Figure 3.2: Neighbours of **vertex 1**. On the left: **degree=1**, on the right: **degree=2**

In Figure 3.2, we can see an example of graphs with different degrees of neighbourhood for a given vertex. The vertices that can be reached for a certain degree are coloured in grey.

Definition 3.4 - Cluster

There is no exact definition of a cluster. For different branches of science, there are different definitions of this term. Basically, clusters describe a group of items or objects that have similar behaviour or share similar properties. In the context of this master thesis, a **cluster** represents a set of words that have similar meaning.
For example: rock,music,drums,guitar is a **cluster** of words describing something about music and the music style rock. The members of a **cluster** we denote as *seed* and we will use this notation throughout the following sections in this thesis.

3.2 Data Structures in Vector Space

Definition 3.7 - Feature Vector

A **feature vector** is a n-dimensional vector containing a numerical representation of features describing an object. Let a vector w has the following elements: $w = (0, 1, 1, 0, 1, 0, 1, 1, 1)$. In the context of this master thesis, a **feature vector** describes how a particular word is connected to a set of other words. If two words are connected, the value in the vector is 1, otherwise it is 0. Figure 3.3 shows an example of two feature vectors placed in a coordinate system.

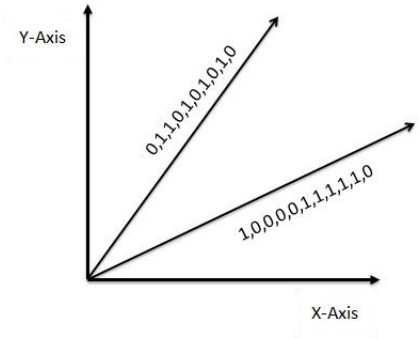


Figure 3.3: Feature vectors

Definition 3.8 - Centroid

In the context of **feature vectors**, a **centroid** describes an average representation of the objects encoded in the **feature vector**. So, if we have three feature vectors $w_1 = (0, 1, 1, 0, 1, 0, 1, 1, 1)$, $w_2 = (1, 1, 1, 1, 1, 0, 1, 1, 1)$ and $w_3 = (1, 1, 1, 0, 1, 0, 1, 0, 0)$, the **centroid C** of these vectors is going to have the following elements: $C = (2, 3, 3, 1, 3, 0, 3, 2, 2)$

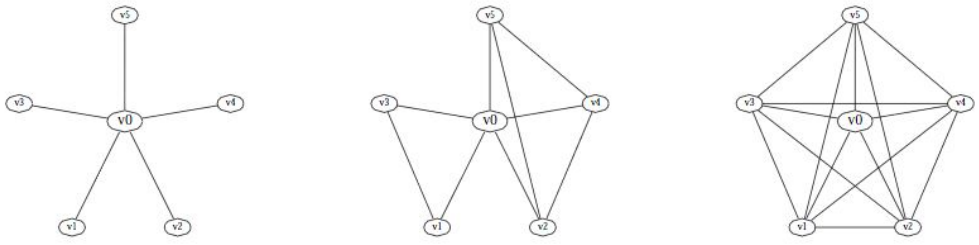
3.3 Similarity Measures

Definition 3.5 - Curvature

[?, p.31,p.33]A **curvature** value measures the interconnectedness of vertex's neighbours. This value gives an insight of the graph's topology and can be geometrically interpreted as the curvature of a graph at a given node. Let **v** be a vertex in a graph G and N(v) is the set of all neighbours of **v**. The **curvature** of **v** is defined as:

$$curv(v) = \frac{\#(\text{triangles } \mathbf{v} \text{ participates in})}{\#(\text{triangles } \mathbf{v} \text{ could possibly participate in})}$$

The **curvature** can vary between 0 and 1. If no of the **v**'s neighbours are connected then the value is 0 and if all neighbours are linked to each other then it is 1. Figure 3.4 shows an example of different curvatures for a vertex v_0.



(a) curv(v_0) = 0. (b) curv(v_0) = 0.4. (c) curv(v_0) = 1.

Figure 3.4: Example of different curvature values: from left to right - low, middle, high [?]

Definition 3.6 - Jaccard Index

A **Jaccard Index** is a basic similarity measure that calculates how similar two sample sets are, by computing the fraction between the size of the intersection and the size of the union of the sample sets. Let **X** and **Y** be two sets. The Jaccard Index is defined as follows:

$$\frac{|X \cup Y|}{|X \cap Y|}$$

Definition 3.9 - Cosine Similarity

Let **A** and **B** be two vectors. The **cosine similarity** measures the closeness between two vectors, that means to measure the angles between the two vectors. One possible way to describe it is as follows:

$$\frac{A \cap B}{\sqrt{\mid A \mid} * \sqrt{\mid B \mid}}$$

Cosine similarity of -1 is for vectors having opposite directions. Values from 0 to 1 denote an increasing similarity.

Chapter 4

Algorithms

In this chapter we will introduce the data model that we base our algorithms on. Afterwards, we will describe in details the algorithms that we propose as a solution for the cluster tracking problem.

4.1 Introduction Of The Used Data Model

The data model used in this master thesis is built as a *word co-occurrence graph*. This graph is created from extracted words of newspaper articles from two digitalised archives – the Times Archive[?] and the New York Times Archive[?]. The Times Archive articles span from year 1785 to 1985 and the New York Times – from year 1987 to 2007.
Tahmasebi et.al. describe in their paper [?] how the articles were processed and according to which pattern words from the articles are mapped to the graph. In the next section, we will go through the important steps of this process in brief.
Firstly, the data was cleaned and prepared for natural language processing. After that, nouns and noun phrases were identified by applying a part-of-speech tagger and lemmatiser. If a lemma can be derived from a word in an article, then it is added to a term list which represents a *dictionary* of words used in articles for a particular year.
The next step is to create the *co-occurrence graph* using the dictionary and the following pattern:

> ”terms from the dictionary, that are found in the text separated by an "and", an "or" or a comma are considered to be co-occurring.” [?]

After a graph is constructed for an entire year, all co-occurrences are filtered and these that have frequency above a given threshold are kept in the graph. By applying the curvature clustering algorithm by Dorow [**?**], for every node in the co-occurrence graph a curvature value is computed. All nodes that have a curvature value below some threshold are removed. After that, the remaining graph contains connected components that are called clusters and are considered to represent word senses. The clusters are extended with the nearest neighbours of its members.
The evaluation of the data sets indicates that the implemented algorithms show good quality in the extraction of word senses from historic documents.

4.2 General Preconditions And Computations

In the following sections, we will discuss the implemented algorithms and describe which features of the data model are used for measuring similarity between clusters.
First we start with a short introduction of some basic computations that are necessary for every algorithm.
The first step is to take two *co-occurrence graphs* for two subsequent years, e.g. 1875 and 1876. For each of these graphs, there is a set of clusters that represents all of the word senses in the graph. With $\$_i$ and $\$_j$ we will denote the sets of clusters for year y_i and y_j respectively. For y_i and y_j, it holds that $y_j = y_i + 1$.
For every cluster pair a Jaccard similarity is computed between the cluster

Algorithm 1 Basic Computations

```
DEFINE: $_i {Set of clusters for year y_i}
DEFINE: $_j {Set of clusters for year y_j}
for ∀c_i in $_i do
  for ∀c_j in $_j do
    jaccIdxSim = jaccIdxSim(c_i, c_j) {The Jaccard Index is calculated
    between the components of the clusters}
    if jaccIdxSim > threshold then
      compute cluster similarity with ALG_xy
    end if
  end for
end for
```

elements. If this value is above a given threshold we use the clusters as input for the algorithms computing the similarity between clusters. The threshold

used in algorithm 1 is 0.3. If elements from one cluster are a sub-set of the elements of the second cluster, this pair of clusters is not processed by the similarity algorithms.

Extract Part Of Co-Occurrence Graph

In our algorithms we compare clusters pairwise. For each cluster we extract a part of the co-occurrence graph from which the clusters where created. For every word in the cluster, considered as a *seed*, we extract its n-degree neighbours to get a sub-graph containing the cluster members and their n-degree neighbours. These additional nodes will help describe the nodes in the cluster further and add information. Using the sub-graphs for finding similarities, we limit our computations to a small set of nodes.
Figure 4.1 shows an example of extracted part of co-occurrence graph. The seed contains nodes 1-5. The 1-degree neighbours are nodes 6-11. The rest of the graph is depicted as white coloured nodes.

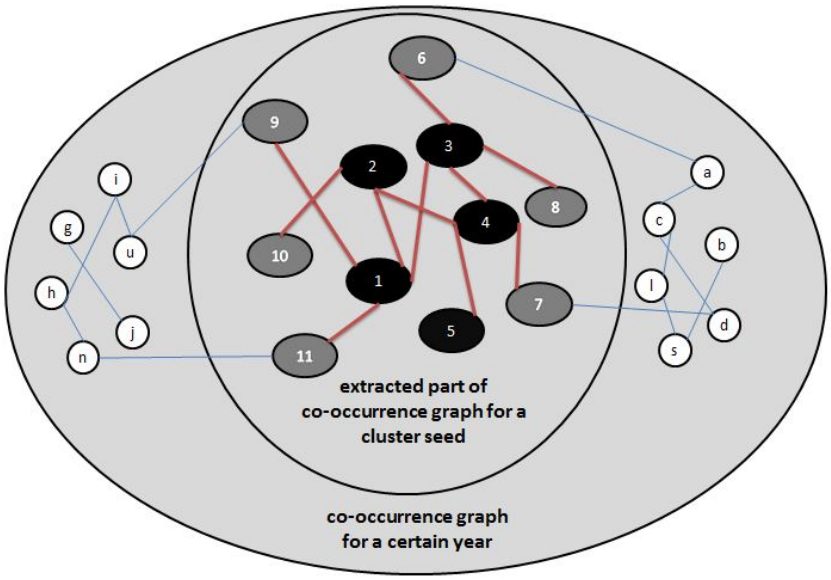


Figure 4.1: Extracting part of a co-occurrence graph for a cluster (the cluster members indicated in black) with 1-**degree neighbourhood** (indicated in grey).

Building Feature Vectors and Centroids

Let C be a cluster in a word co-occurrence graph $\mathbf{G}$, $C = (w_1, w_2, \ldots)$. Let $subG$ be the sub-graph of G with the cluster members and their $1-$degree neighbours. For each word $w_i \in subG$ we build a feature vector. The feature vectors represent the interconnectedness of the word to all other words in the sub-graph. We compute a **centroid** for a sub-graph by adding all feature vectors for the sub-graph.

4.3 Similarity Algorithm 1

The first algorithm for measuring similarity between clusters uses **feature vectors** and **cosine similarity** between **centroids**. We process a pair of clusters from two *co-occurrence graphs* for subsequent years. For every cluster we extract a sub-graph as described in section 4.2 and find the centroid of the sub-graph. The similarity between the clusters is calculated as the cosine similarity between the corresponding sub-graph centroids. The algorithm is described in pseudo code in Algorithm 2.

Algorithm 2 Similarity Algorithm 1

DEFINE: *clusterSimilarity* to hold the end result of similarity
DEFINE: $degree = 1$; $threshold = 0.3$
DEFINE: $subG_i$ and $subG_j$ to hold the extracted part of co-occurrence graphs G_i and G_j
DEFINE: $FVset_i$ and $FVset_j$ to hold the feature vectors
DEFINE: $centroid_i$ and $centroid_j$
DEFINE: $\$_i$ {Set of clusters for year y_i}
DEFINE: $\$_j$ {Set of clusters for year y_j}
for all c_i, c_j in $\{\$_i, \$_j\}$ such that **jaccIdxSim**$(seed_i \in c_i, seed_j \in c_j) > threshold$ **do**
  $subG_i =$ **ExtractPartOfCoOccGraph**$(degree, seed_i \in c_i, G_i)$
  $subG_j =$ **ExtractPartOfCoOccGraph**$(degree, seed_j \in c_i, G_j)$
  $\forall w_i \in subG_i$ compFeatureVectorForWord() $\rightarrow FVset_i$
  $\forall w_j \in subG_j$ compFeatureVectorForWord() $\rightarrow FVset_j$
  $centroid_i =$ **computeCentroid**$(FVset_i)$
  $centroid_i =$ **computeCentroid**$(FVset_j)$
  clusterSimilarity = **computeCosineSim**$(centroid_i, centroid_j)$
end for

4.4 Similarity Algorithm 2

The second algorithm for computing similarities between clusters is based on a combination between cosine similarity and Jaccard similarity. As in Algorithm 1, we consider clusters pairwise and calculate the feature vectors for all words in the sub-graphs corresponding to the clusters. For each feature vector FV_j from the first sub-graph we calculate the maximum similarity to any feature vectors corresponding to the second sub-graph. The maximum similarity is added to a total similarity score between the clusters. When we have found the maximum similarity between all pairs of feature vectors in the two sub-graphs, we normalize the total similarity score with the total number of times in which the cosine similarity between two feature vectors was above a given threshold. In the final step we compute the Jaccard similarity between the cluster seeds and we calculate the average between the Jaccard similarity and the normalised total similarity score. This average value is the similarity between the two clusters. A pseudo code description for the algorithm can be found in Algorithm 3

Algorithm 3 Similarity Algorithm 2

DEFINE: *clusterSimilarity* to hold the end result of similarity
DEFINE: *tempMaxCoSineSim* to hold the temporal maximum of cosine similarity
DEFINE: *CS* to hold the sum of all temporal max. cos similarities
DEFINE: *degree* = 1.0; *threshold* = 0.3; normalise = 1.0;
DEFINE: $subG_i$ and $subG_j$ to hold the extracted part of co-occurrence graphs G_i and G_j
DEFINE: $FVset_i$ and $FVset_j$ to hold the feature vectors
DEFINE: $\$_i$ {Set of clusters for year y_i}
DEFINE: $\$_j$ {Set of clusters for year y_j}
for all c_i, c_j in $\{\$_i, \$_j\}$ such that **jaccIdxSim**$(seed_i \in c_i, seed_j \in c_j) >$ *threshold* **do**
  ExtractPartOfCoOccGraph$(degree, seed_i \in c_i, G_i) \rightarrow subG_i$
  ExtractPartOfCoOccGraph$(degree, seed_j \in c_i, G_j) \rightarrow subG_j$
  $\forall w_i \in subG_i$ compFeatureVectorForWord() $\rightarrow FVset_i$
  $\forall w_j \in subG_j$ compFeatureVectorForWord() $\rightarrow FVset_j$
  for $\forall fv_i \in FVset_i$ **do**
    tempMaxCoSineSim = 0.0
    for $\forall fv_j \in FVset_j$ **do**
      cosineSim = **computeCosineSim**(fv_i, fv_j)
      if cosineSim > threshold **then**
        tempMaxCosineSim = **MAX**$(tempMaxCosineSim, cosineSim)$
      end if
    end for
    if $tempMaxCosineSim > 0$ **then**
      normalise = normalise + 1.0
      CS = CS + tempMaxCosineSim
    end if
  end for
  clusterSimilarity = textbfAVG($\frac{CS}{normalize}$ + **jaccIdxSim**$(seed_i \in c_i, seed_j \in c_j)$
end for

4.5 Similarity Algorithm 3

In the third algorithm, we try a different approach in order to compare two clusters. This approach merges sub-graphs corresponding to two clusters and re-clusters the merged sub-graph with the curvature clustering algorithm used to create the clusters.
As in the previous two algorithms, we start by taking two co-occurrence graphs and we process their cluster sets, where we compare every pair of clusters which has Jaccard similarity above a certain threshold. So, for every cluster seed from each cluster set we extract a part of co-occurrence graph. Now, having these two sub-graphs $\mathbf{subG}_1$ and $\mathbf{subG}_2$ we can build a new graph as a result of a merging operation between $\mathbf{subG}_1$,$\mathbf{subG}_2$. On the merged graph we apply the curvature clustering algorithm which results in a new set of clusters. Next, we iterate over these clusters and calculate the Jaccard similarity between the new cluster and the two original clusters. Figure 4.2 shows a pipeline of the steps that we need, so we can get a new cluster set from the merged graph. .

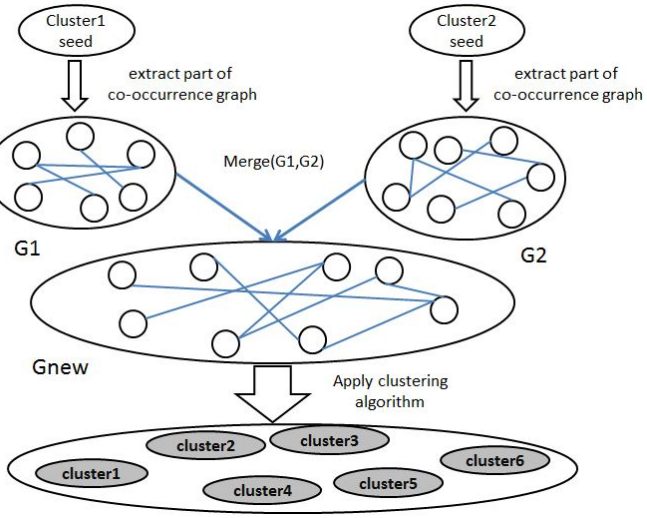


Figure 4.2: Algorithm 3 - Pipeline

Let c_1 and c_2 be the clusters we want to find the similarity between. c_{new} is a cluster from the new generated cluster set C_{merged}. For every c_{new} in C_{merged} we compute:

A= jaccIdxSim(c_1, c_{new})

B= jaccIdxSim(c_2, c_{new})

The similarity between c_1 and c_2 is considered to be the maximum $AVG(A, B)$ for any c_{new}.

Chapter 5

Experiments And Evaluation

This chapter describes the experiments and the evaluation conducted on the basis of the algorithms proposed in chapter 4. In Section 5.1, we explain which dataset we used for the experiments and how we built it. Section 5.2 describes the human evaluations that were performed. Afterwards, in Section 5.3 we give an overview of the experiments. In the next Section 5.4, we will introduce the results that the algorithms have achieved. We will also compare the measured values from the algorithms to the human evaluations. In Section 5.4 we present the results of our experiments and compare the experimental results to the human evaluations. Finally, we discuss the results in Section 5.5

5.1 Dataset

The clusters we use for these experiments are created using The Times Archive from The Times in London (LT) and the New York Times Archive (NYT). For every year there exists clusters of word senses created with the curvature clustering algorithm using curvature threshold 0.3. We have selected 4 pairs of subsequent years for LT and 4 pairs of subsequent years for NYT articles. For every pair of two subsequent years, we create a file. In this file, we put every pair of clusters from the two subsequent years that have Jaccard similarity above 0.3.
So, for year 1785 and 1786 we process the two cluster files – *clusters_1785* and *clusters_1786*. For each cluster from *cluster_1785* , we iterate over all clusters from *clusters_1786* and if they have Jaccard similarity above 0.3, we put these two clusters in the file *clusters_1785_1786*.

The file *clusters_1785_1786* contains all cluster pairs from 1785 and 1786 which have a Jaccard similarity of at least 0.3. The content of the file shown below:

idxCl1_TO_idxCl2: C_1785_62 <−> C_1786_50
Cluster1−>C_1785_62: ...excellent_kitchen;garden;stable;stabling;
Cluster2−>C_1786_50: ...farmyard;garden;six_acre;stable;stabling;...

The first line is a unique identifier of the cluster pair. On the next two lines, we can see two cluster seeds from 1785 and 1786 respectively.
The years and the number of clusters that we have selected for both archives can be seen in Table 5.1. In total, our dataset contains 8 files {4 files for the

type of articles	**years**	**number of cluster pairs**
LT	1785 - 1786	3
LT	1850 - 1851	30
LT	1900 - 1901	48
LT	1950 - 1951	43
NYT	1989 - 1990	66
NYT	1999 - 2000	83
NYT	2001 - 2002	87
NYT	2005 - 2006	62

Table 5.1: Dataset from NYT and LT articles for the selected pairs of years

LT articles and 4 files for the NYT articles }. The total number of clusters is 422.

5.2 Human Evaluations

In order to verify and assess the results from our algorithms, we asked 15 people to manually evaluate some of the clusters from Table 5.1. We chose persons from different professions and backgrounds: high school graduates, a lawyer, students, Ph.D. researchers, a technical communicator, a software developer and a logistic specialist.
The dataset used for the human evaluations was extracted from the dataset described in section 5.1. We have selected clusters from every file in the dataset randomly. We have limited the number of clusters that we took from each file to 30-50. As a result, we gained 237 unique cluster pairs that we mixed and saved in 8 files. The number of cluster pairs in the new generated files varies between 41 and 70. The total number of evaluated clusters is

400, where some of the clusters were evaluated multiple times by different evaluators. In these cases, we built the average value of the scores that the persons gave for the clusters. The content of the evaluation files is as follows:

idxCl1_TO_idxCl2: C_1785_62 <−> C_1786_50
#**human-score** {choose between: -1, 0, 25,50,75 and 100 percent}:
Cluster1− >C_1785_62: ...excellent_kitchen;garden;stable;stabling;
Cluster2− >C_1786_50: ...farmyard;garden;six_acre;stable;stabling;...

The first line is a unique identifier. Such identifiers exist for each cluster pair in the datasets used for the algorithms experiments and for the human evaluations. With the help of these identifiers, we can compare in an automatic way the results from the algorithms to the human evaluations. With -1, the evaluators mark a pair of cluster for which they can not decide if the clusters represent any mutual word sense. The values from 0 to 100 are to be chosen for an increasing similarity between the two clusters in the cluster pair.

5.3 Experiments

In the following section we will present the experiments that were performed.

We ran our three algorithms over the extracted 533 cluster pairs (see Section 5.1). The files in which we saved our results have the following structure:

idxCl1_TO_idxCl2: C_1785_62 <−> C_1786_50
Cluster1− >C_1785_62: ...excellent_kitchen;garden;stable;stabling;
Cluster2− >C_1786_50: ...farmyard;garden;six_acre;stable;stabling;...
JaccIdx-Clusters: ....
simAlg1: ...
simAlg2: ...
simAlg3: ...

On the last four lines, we save the similarity values calculated for the given cluster pair and compare the similarity score for each cluster pair with the human evaluation for the same pair. We use the Jaccard similarity as a baseline and compare our 3 similarity algorithms to see which algorithm provided the best results. So, for every two subsequent years that we have selected, we create such file with the results saved in it.

5.4 Evaluation Of The Results

This Section describes the results achieved by the algorithms and investigates how good these results cover the human evaluations.
For each algorithm, we calculate **precision** and **recall** to assess the performance of the algorithm. The **recall R** is defined as follows:

$$R_i = \frac{\#\{\text{correctly guessed cluster pairs for interval i}\}}{\#\{\text{human evaluated cluster pairs in interval i}\}}$$

The recall is calculated for 5 different intervals: $[0, 0.25), [0.25, 0.5), [0.5, 0.75), [0.75, 1), 1$.
The sum of the recalls for each interval is the total recall:

$$R = \frac{1}{5} \sum_{i \in [0,0.25),[0.25,0.5),[0.5,0.75),[0.75,1),1} R_i$$

For the **precision P** we use the following definition:

$$P = \frac{\#\{\text{correctly guessed cluster pairs}\}}{\#\{\text{human evaluated cluster pairs}\}}$$

In the next subsection, we will introduce the results for each one of the proposed algorithms. Unfortunately it was not possible to run tests for Algorithm 3 (see 4.5) and we can not present results for this algorithm.

Results - Similarity Algorithms

In the first Table 5.2 we give a summary of how the human scores are distributed.

interval	**percentage**
$[0-25)$	6%
$[25-50)$	8%
$[50-75)$	27%
$[75-100)$	34%
100	24%

Table 5.2: Human Evaluations - Distribution Of The Human Score Values By Intervals Of Similarity

The following tables represent the calculated precision and recall for each of the algorithms and for the Jaccard similarity between the cluster members.

algorithm	**precision - degree=0**	**precision - degree=1**
Algorithm 1	41%	30%
Algorithm 2	39%	42%
jaccSimClusters	35%	35%

Table 5.3: *Precision* For Different Degrees Of Neighbourhood

algorithm	$(0-25]$	$(25-50]$	$(50-75]$	$(75-100]$	100	**total R**
Algorithm 1	0%	35%	89%	11%	0%	27%
Algorithm 2	0%	95%	27%	25%	0%	29%
jaccSimClusters	0%	80%	16%	20%	0%	23%

Table 5.4: *Recall* For Degree=0 In Different Intervals Of Similarity

algorithm	$(0-25]$	$(25-50]$	$(50-75]$	$(75-100]$	100	**total R**
Algorithm 1	27%	65%	30%	0%	0%	24%
Algorithm 2	0%	85%	65%	12%	0%	32%
jaccSimClusters	0%	80%	16%	20%	0%	23%

Table 5.5: *Recall* For Degree=1 In Different Intervals Of Similarity

5.5 Discussion

In this chapter we presented the results from the conducted experiments. We compared the results from the proposed algorithms to the human evaluations by calculating the *recall* and *precision* values. On the basis of these values we can conclude that the overall correctness of the algorithms is below average.

The statistics for *Algorithm 1* (see algorithm 4.3), shown in Table 5.3 indicate a higher precision (41% correctly guessed cluster pairs) for the experiment with **degree=0**. On the other hand, when **degree=1**, the results are less precise (30%). This can be explained by the growing number of words that we add to the centroids when expanding the words neighbourhood by **degree=1**. Because of this, the computation of the cosine similarity between the centroids is affected.

Algorithm 2(see algorithm 4.4), on the other hand achieves better results in the experiment with **degree=1** (42% correctly guessed cluster pairs). These results are slightly better than the precision value of the baseline i.e., Jaccard similarity between clusters (35% correctly guessed cluster pairs).

As the recall values presented in Table 5.4 and 5.5 show, the algorithms do not correctly measure the cluster similarity in all intervals. In intervals of low similarity the algorithms achieve low recall. The highest value of recall is reached in the intervals 25%-50% and 50%-75%. The algorithms can detect just a few or no cluster similarities in the both ends of the intervals. More work is needed to achieve higher recall and precision in the end intervals.

Chapter 6

Conclusion And Future Work

In this master thesis we investigated different measures of similarity between clusters of terms as a first step towards discovering language evolution. In our algorithms we tried to examine the neighbourhood of words that are member of clusters, extracting a part of the co-occurrence graph which was used to create the cluster in first place. We used the cluster itself as well as the 1-degree neighbours of the cluster members in order to compare clusters. In this manner we investigated if the new added information to the clusters helps tracking word sense changes. Some results indicated that constructing centroids with too many words make computation less precise. On the other hand, for experiments with an added neighbourhood, the cosine similarity between the feature vectors provided better results.
Overall, the evaluation showed that our algorithms provide promising results, however, there is a need for fine tuning the thresholds to compute the similarities between clusters more accurately.

Future Work

One approach to increase the quality of computing similarity between clusters is when we combine the proposed *Algorithm 1* and *Algorithm 2*(see Algorithms 4.3 and 4.4) and set proper thresholds. The fine tuning of the thresholds can improve the cosine similarities between the centroids. The size of the centroids can be shrunk according to the co-occurrence values of the terms represented in the centroids. A Jaccard similarity between centroids can be used as an additional control value in the algorithms.
Future investigations may involve the proposed methods in the paper [**?**] where for a given member of cluster we can calculate how high is the probability that this member can leave the cluster or not.

Acknowledgements

I would like to give my special thanks to my supervisor Nina Tahmasebi for her constant guidance and support.
Great thanks go to my girlfriend for her patience and understanding that she had in the last six months. I would like also to thank to all of the participators in the evaluations.
We would like to thank Times Newspapers Limited for providing the archive of The Times for our research.