APRIORI APPROACH TO GRAPH-BASED CLUSTERING OF TEXT DOCUMENTS

by

Mahmud Shahriar Hossain

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2008

APPROVAL

of a thesis submitted by

Mahmud Shahriar Hossain

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citation, bibliographic style, and consistency, and is ready for submission to the Division of Graduate Education.

Dr. Rafal A. Angryk

Approved for the Department of Computer Science

Dr. John Paxton

Approved for the Division of Graduate Education

Dr. Carl A. Fox

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Mahmud Shahriar Hossain

April 2008

To

My Father

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

ABSTRACT

This thesis report introduces a new technique of document clustering based on frequent senses. The developed system, named GDClust (Graph-Based Document Clustering) [1], works with frequent senses rather than dealing with frequent keywords used in traditional text mining techniques. GDClust presents text documents as hierarchical document-graphs and uses an Apriori paradigm to find the frequent subgraphs, which reflect frequent senses. Discovered frequent subgraphs are then utilized to generate accurate sense-based document clusters. We propose a novel multilevel Gaussian minimum support strategy for candidate subgraph generation. Additionally, we introduce another novel mechanism called Subgraph-Extension mining that reduces the number of candidates and overhead imposed by the traditional Apriori-based candidate generation mechanism. GDClust utilizes an English language thesaurus (WordNet [2]) to construct document-graphs and exploits graph-based data mining techniques for sense discovery and clustering. It is an automated system and requires minimal human interaction for the clustering purpose.

CHAPTER 1

INTRODUCTION

In this chapter, we introduce the motivation of Graph-based Document Clustering, its mutual benefits over traditional clustering techniques, and the related literature. We call our system GDClust, which stands for Graph-based Document Clustering. We also developed an enhanced version of the GDClust-structure that significantly outperforms our original version of the system. We call this new system GDClust-Plus. The overall GDClust system is described in Chapter 2. As an improvement in GDClust-Plus, we introduced a new mechanism and named it Subgraph-Extension mining. It is described in Chapter 3. In the context of this thesis, we shall use the terms GDClust and GDClust-Plus interchangeably. When we refer to GDClust-Plus, we indicate our specific enhancement. It should be mentioned that both systems are built on the GDClust-architecture, but some of the graph-mining techniques are enhanced in GDClust-Plus to improve performance.

Problem Description and Motivation Behind Graph-based Document Clustering

The last decade has seen a significant increase in research on text clustering, natural language processing and textual information extraction. Most of these techniques rely on searching for identical words and counting their occurrences. The goal of our research is to develop a new, human-like, hierarchical document clustering technique driven by recent discoveries in the area of graph-based data mining and hierarchical clustering of text documents. The major motivation for our approach comes from typical

human behavior when people are given the task of organizing multiple documents. As an example, consider the behavior of a scientific book editor who is faced with the complicated problem of organizing multiple research papers into a single volume with a hierarchical table of contents. Typically, even papers from the same research area are written (1) in multiple writing styles (e.g. using “clusters” instead of “concentration points”), (2) on different levels of detail (e.g. survey papers versus works discussing the complexity of a single algorithm) and (3) in reference to different aspects of an analyzed area (e.g. clustering of numeric versus descriptive data). Instead of searching for identical words and counting their occurrences, as many well-known computer-based text clustering techniques do [3, 4, 5], the human brain usually remembers only a few crucial keywords, which provide the editor with a compressed representation of the analyzed document. These keywords, discovered thanks to the expert’s knowledge (replaced in our case by ontology), are then used by the book editor to fit a given research paper into a book’s organization scheme, reflected by the table of contents.

The major focus of this work was to develop techniques that deal effectively with the multiple levels of abstraction occurring in our natural language. We developed an approach that organizes text data into a meaningful hierarchy based more on the gradual similarity of ideas carried in the papers and reflected by the topic’s ontology rather than on the identity of words included in the papers. This provides a broad range of computer users with the ability to process text data in a more effective way.

As the outcome, we achieve a hierarchy of meaningful terms that are generated by the intersection of graphs (i.e. graph representation of text files) and the WordNet

ontology [2, 6] available for the topic. Human beings or computers can use this hierarchy to efficiently navigate enormous repositories of text data.

We sight an example where traditional keyword based techniques are unable to retrieve similar senses from two text documents, but GDClust is able to discover their similarity. Consider two documents, *doc1* and *doc2*. Suppose *doc1* contains the terms *insect*, *lepidopteran* and *caterpillar*, whereas *doc2* contains the terms *myriapod*, *merostomata* and *mealworm*. Although human beings can easily understand that both of the documents contain information about some *arthropod*, traditional text mining techniques will consider these two documents distinct from each other (i.e., in two

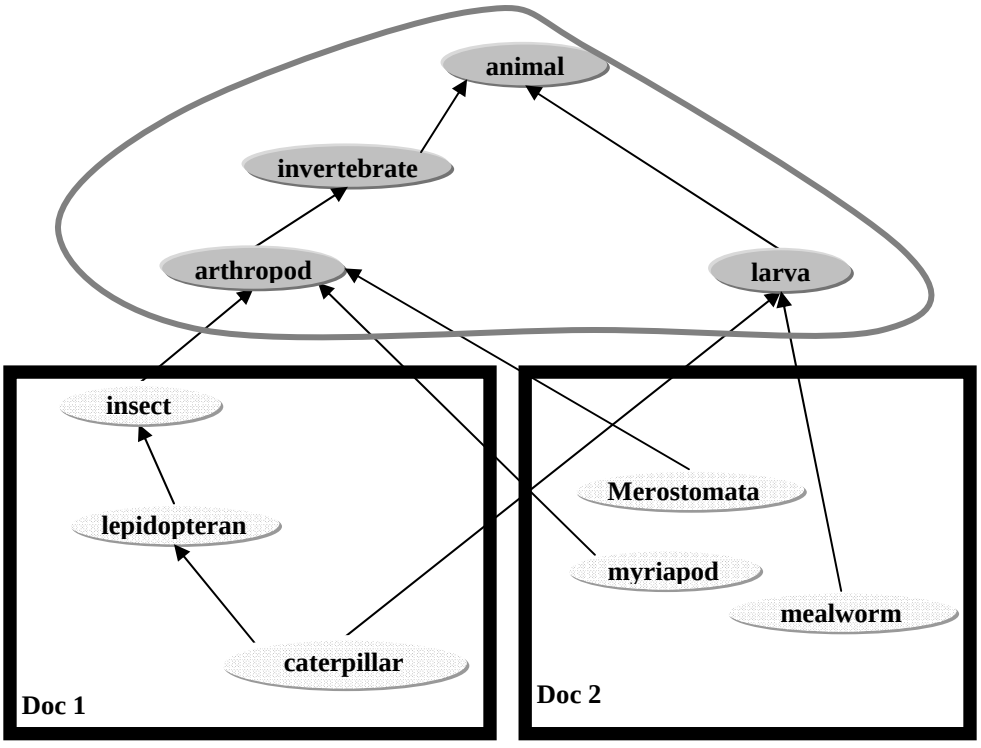


Figure 1: Traditional keyword-based approach and sense-based approach.

different clusters) because the documents do not have any common term. In contrast, GDClust offers a graph-based approach where the system is able to automatically investigate the abstractions of the terms. GDClust detects high similarity between *doc1* and *doc2* by detecting a common subgraph for these two documents in the language ontology and places them in the same cluster. Figure 1 illustrates this scenario between the traditional approach and our sense based approach. In a traditional approach, these two documents are totally different, as they do not have any common terms, but in our approach the abstractions of both documents overlap and signify high degree of similarity.

With implementing GDClust, we aim to develop a document clustering technique that is able to cluster documents using common senses rather than perfectly matching keywords. We provide a clustering technique that uses knowledge about terms, rather than trust in document corpora to provide such information implicitly. Therefore, the developed system works well both for a very large set of documents as well as for smaller document repositories with skewed (e.g., certain topic related) distributions of frequent terms.

Scope

In this thesis, we developed a graph-mining technique for clustering text documents. We represent the documents of a repository as graphs. Our system depends on background knowledge of the English language ontology that is constructed from the

IS-A relationships of noun words of the WordNet lexical reference system [2]. In this work, we have concentrated on the noun keywords of the documents.

We utilize an Apriori paradigm [7] to mine subgraphs that was originally developed for mining frequent itemsets in a market basket dataset [8]. We exploit Hierarchical Agglomerative Clustering (HAC) [9] to cluster text documents based on the appearance of frequent subgraphs in the graph representations of the documents. Another aim of this thesis is to analyze clustering quality by investigating different similarity measures. We use the Silhouette Coefficient to calculate the quality of the clustering. We also compare the quality of clustering using GDClust with the traditional keyword-based approaches.

Literature Review

The benefit of our document clustering approach is that it can group documents in the same cluster even if they do not contain common keywords. Instead, the clustered documents possess the same sense, discovered by the similarity of common abstract terms and relations between them, which is reflected in our document-graphs. Other existing clustering techniques cannot perform this sort of discovery or do this work only to a limited degree, such as Latent Semantic Index (LSI) [10, 11]. LSI can cluster documents even if their keywords are not common, but it depends on other common words appearing frequently in the document while looking for a specific keyword. Although this kind of clustering is very popular, it depends entirely on a large number of input documents to broaden knowledge, which is naturally limited to the information

implicitly stored in the clustered documents' corpus. Other document clustering techniques [3, 4, 5] depend on either probabilistic methods or distance and similarity measures between keywords. Their performance depends on the selection of keywords and on the construction of feature vectors for documents. All these mechanisms suffer from the fact that they do not offer human-like, sense-based document clustering.

Developing algorithms that discover all frequently occurring subgraphs in a large graph database, is particularly challenging and computationally intensive, since graph isomorphism plays a key role throughout the computations [12]. Nevertheless, various researchers have used graph models in complex datasets and found them useful in the chemical domains [13, 14, 15, 16], computer vision technology [17, 18], image and object retrieval [19], social network analysis [20] and machine learning [21, 22, 23]. In our work, we utilize the power of graphs to model a complex sense of text data.

There had been extensive research work on generating association rules from frequent itemsets [24, 25]. Agrawal et al. [7] proposed the Apriori approach for association rule mining [8]. Park et al. [26] proposed a hash table-based version of the Apriori approach improve the efficiency of association rule mining. Additionally, some transaction reduction approaches have been proposed by Agrawal et al. [7], Park et al. [26] and Han et al. [27]. In our work, we utilize a variation of multilevel association rule mining [27] for the frequent sense discovery process by proposing a novel Gaussian minimum support strategy [1] for the frequent subgraph discovery on multiple levels of the English language taxonomy.

Kuramochi et al. [12, 28] present an efficient algorithm named Frequent Subgraph Discovery (FSG) that finds all frequent subgraphs in a large graph database. In the paper, the researchers evaluated the performance of the algorithm using both real and artificial datasets. Their results show that despite the underlying complexity associated with graph isomorphism, FSG is effective in finding all frequently occurring subgraphs in datasets containing over 100,000 graphs and scales linearly with respect to the size of the database. Moreover, Yan et al. [29] describe an algorithm called gSpan (graph-based Substructure pattern mining) that discovers frequent substructures without candidate generation. gSpan builds a new lexicographic order among graphs and maps each graph to a unique minimum Depth First Search code (commonly, known as DFS-code), its canonical label. Based on this lexicographic order, gSpan adopts the depth-first search strategy to mine connected frequent subgraphs efficiently. The performance study, portrayed in the paper, shows that gSpan substantially outperforms some of the other substructure mining algorithms [29], sometimes by an order of magnitude. The existence of efficient algorithms to mine frequent subgraphs from graphs leads us to believe that constructing document-graphs and discovering frequent subgraphs to gain sense-based clustering of our work is feasible. Therefore, in our preliminary version of GDClust [1], we used the FSG strategy in the Apriori paradigm. Later, we introduced the Subgraph-Extension mining technique for efficient candidate generation. We call this enhanced system GDClust-Plus.

There are other well-known subgraph discovery systems like DSPM (Diagonally Subgraph Pattern Mining) [30], TreeMiner [31], GRAFIL (Graph Similarity Filtering)

[32], PIS (Partition-based Graph Index and Search) [33] and SUBDUE [34]. All these systems deal with multiple aspects of efficient frequent subgraph mining. Most of these systems have been tested on real and artificial datasets of chemical compounds. However, none of them has been used to mine text data. In this report, we discuss GDClust that performs frequent subgraph discovery from a text repository with the aim of document clustering.

The work closest to our approach of sense-based clustering that we managed to find in recent literature is a graph query refinement method proposed by Tomita et al. [35]. Their system depends on user interaction for the hierarchic organization of a text query. In contrast, we depend on a predefined ontology (WordNet) to automatically retrieve frequent subgraphs from text documents. GDClust offers a fully automated system that utilizes the Apriori-based subgraph discovery technique to harness the capability of sense-based document clustering.

Outline of the Thesis

We give an overview of the GDClust architecture in Chapter 2. Crucial internal mechanisms of our work are explained in details in Chapter 3 and Chapter 4. We concentrate on graph-mining strategies and their enhancements in Chapter 3. Chapter 4 includes the Hierarchical Agglomerative Clustering (HAC) technique and evaluation measures to assess our clusters' accuracy. We describe important implementation details in Chapter 5. All the experimental results of this work are placed in Chapter 6. Then, we

conclude this thesis in Chapter 7 describing the current limitations of GDClust and our plans for improvements in the future.

CHAPTER 2

GDClust OVERVIEW

In this chapter, we provide an overview of our GDClust system for sense discovery and document clustering. GDClust takes text documents as input, converts each document into corresponding document-graph, finds subgraphs in the document-graphs to establish similarity between senses of the documents and finally outputs possible sense-based clusters of those text documents. This chapter illustrates a top-view of the overall GDClust system.

Architecture

GDClust is composed of two basic components: (1) Conversion Module and (2) GDClust Module. It uses three different databases: (1) Text Document Archive, (2) WordNet and (3) Document Graph Database (DGD). Overview of the entire system is shown in Figure 2.

Conversion Module

Conversion Module is responsible for converting each text document into its corresponding graph representation. The module uses WordNet [2] as its background knowledge. It utilizes BOW Toolkit [36] to retrieve meaningful keywords from the documents. After the retrieval, keywords are stored in the Term-Document Archive. The Conversion Unit picks the document's keywords from the archive (all keywords for a single document at one time) and converts them into a corresponding document-graph.

The Conversion Unit utilizes the WordNet's IS-A hierarchy (hypernymy-hyponymy relationship) to construct the document-graphs. It targets one document at a time to minimize the memory consumption because the generated document-graphs (for say *20,000* document-graphs) may not fit into the main memory. All document-graphs are stored in the same repository, called *Document-Graph Database (DGD)*. The document-graph construction algorithm is described in Table 1 of "Construction of Document-Graph and Master Document-Graph" section.

GDClust Module

GDClust module is composed of two units: *Subgraph Discovery Unit* and *Clustering Unit*. The Subgraph Discovery Unit picks up document-graphs from DGD to discover frequent subgraphs representing frequent senses. Discovered subgraphs are assigned ID's and stored in DGD. The Clustering Unit takes the advantage of the frequent subgraph discovery process. It clusters documents utilizing the discovered frequent senses by the Subgraph Discovery Unit. All other information regarding the clustering (frequent subgraphs, frequency counts of the subgraphs in the documents, etc.) is stored in DGD for future use. Chapter 3 and Chapter 4 respectively describe the subgraph discovery process and the clustering mechanism used by GDClust module.

Text Document Archive

Text Document Archive contains the actual input for GDClust. This archive contains the text documents that need to be clustered. In our experiments, we have used text documents from *20* News Groups dataset [37]. *20* News Groups dataset is used in

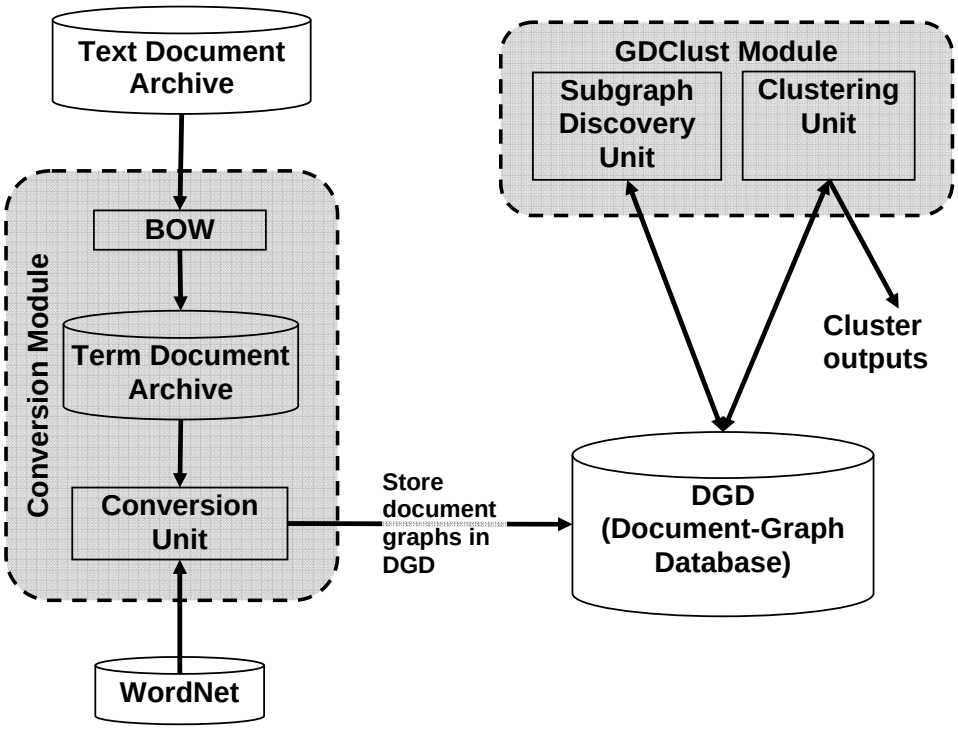


Figure 2: Architecture of GDClust and its flow of data.

different literature and considered as a benchmarking dataset for clustering and classification purpose.

WordNet Dictionary Database

GDClust utilized the WordNet lexical reference system as background knowledge. Our approach does not directly look at the meaning of the related keywords in the WordNet dictionary rather it utilizes the language ontology from references of synsets and their semantic relations. In the context of WordNet ontology, a *synset* is a set of synonymous words in WordNet. GDClust is not a traditional corpora-based mechanism of sense disambiguation. It does not use the description or example sentences of synsets provided by WordNet to distinguish meaning of two keywords. Instead, it

completely relies on document-graphs and the discovered subgraphs in them to find similarity between two documents. Therefore, the use of WordNet in GDClust is limited to the retrieval of the language ontology only.

WordNet divides its whole lexical reference system into five categories: nouns, verbs, adjectives, adverbs and function words [38]. Function words are basically non-content words like prepositions and conjunctions that may mislead language-processing tasks since they are non-informative. In our work, we have concentrated on noun synsets. In WordNet, synsets are usually connected to other synsets via a number of semantic relations. These relations vary based on the type of word. For example, nouns have five kinds of relations, which are stated below [39]:

(1) *hypernyms*: *Y* is a hypernym of *X* if every *X* is a kind of *Y*. Example: *Agaric* is a hypernym of *mushroom*, because *mushroom* is a kind of *agaric*.

(2) *hyponyms*: *Y* is a hyponym of *X* if every *Y* is a kind of *X*. Example: *Mushroom* is a hyponym of *agaric*, because *mushroom* is a kind of *agaric*.

(3) *coordinate terms*: *Y* is a coordinate term of *X* if *X* and *Y* share a hypernym. Example: *Lepiota* is a coordinate term of *mushroom*, because both *Lepiota* and *mushroom* are *agaric*s.

(4) *holonym*: *Y* is a holonym of *X* if *X* is a part of *Y*. Example: *Fomes* is a holonym of *agaric*, because *agaric* is a member of genus *Fomes*.

(5) *meronym*: *Y* is a meronym of *X* if *Y* is a part of *X*. Example: *Agaric* is a meronym of *Fomes*, because *agaric* is a member of genus *Fomes*.

We have utilized the *hypernymy-hyponymy* relationships of noun synsets to build up the English language ontology.

Document-Graph Database

GDClust uses BOW Toolkit and the WordNet 2.1 taxonomy to convert a document to its corresponding document-graph. The outcomes of the Conversion Module of Figure 2 are document-graphs representing the input text documents. These documents are stored in Document-Graph Database to be used later by the GDClust module. Besides the regular document-graphs, we store a Master Document-Graph (MDG) in the Document-Graph Database, which helps us later during subgraph discovery by providing the language ontology. The construction mechanism of Master Document-Graph is described in the following section.

Construction of Document-Graph and Master Document-Graph

GDClust utilizes document-graphs representing text documents. Constructed document-graphs become inputs for the GDClust Module of Figure 2. In this section, we describe the algorithm for constructing document-graphs and our Master Document-Graph.

Table 1 illustrates the algorithm for construction of individual document-graphs. WordNet provides a hierarchic representation of English words. We utilized the WordNet's noun taxonomy, which provides a *hypernymy-hyponymy* relation between concepts and allows constructing a *concept tree* with up to maximum *18* levels of

Table 1: Algorithm for construction of document-graphs.

(1)	For each document D_i, construct a document-graph G_i, where $1 \leq i \leq n$, and n is the total number of documents {
(2)	For each keyword, k_j where $1 \leq j \leq m$ and m is the number of keywords in document D_i {
(3)	Traverse WordNet taxonomy up to the topmost level. During the traversal, consider each synset as a vertex. E is considered as a directed edge between two vertices V_1 and V_2, iff V_2 is the hypernym of V_1.
(4)	E is labeled by $V_1::V_2$. If there is any repeated vertex or edge that was detected earlier for another keyword k_t $(t \neq j)$ of the same document, D_i, do not add the repeated vertices and edges to G_i, otherwise, add vertices and edges to G_i.
(5)	} *// End of "For each keyword"*
(6)	} *// End of "For each document"*

abstractions. A *concept* in a document-graph is a node containing the *synset* from WordNet. All nouns in WordNet are merged to a single topmost synset (i.e. *{entity}*). Our document-graph construction algorithm selects informative keywords from the document and retrieves corresponding synsets from WordNet. Then, it traverses to the topmost level of abstraction to discover all related abstract terms and their relations. The graph of the links between keywords' synsets of each document and their abstracts compose the individual document-graph.

As stated in previous section, Document-Graph Database also contains another large graph representing the whole ontology needed for subgraph discovery. We name this graph a *Master Document-Graph (MDG)*. An MDG is a single graph composed of all the edges and vertices of all the document-graphs in the database. In other words, a Master Document-Graph is a combined document-graph containing all the keywords found in all text documents. More formally,

$$E_{D_i} \subseteq E_{MDG} \tag{2.1}$$

where E_{D_i} indicates the edge-set of a document-graph D_i and E_{MDG} denotes the edge-set of the Master Document-Graph. Equation (2.1) indicates that the edge-set of a document-graph is a subset of the edge-set of the Master Document-Graph. Besides that,

$$E_{MDG} \equiv \bigcup_{D_i} E_{D_i} \tag{2.2}$$

Equation (2.2) indicates that the Master Document-Graph is composed of all the edges of all the document-graphs. Each edge of a document-graph has a unique DFS-code. This DFS-code is generated from the DFS order traversal of the DFS-tree [40] of the Master Document-Graph. GDClust forces the DFS-code of the edges of the Master Document-Graph to all the edges of all document-graphs. Therefore, an edge of a document-graph can be identified using the Master Document-Graph-driven DFS-code.

Later, in Chapter 3, we illustrate how a Master Document-Graph benefits the pruning mechanism at the very first level of the Apriori paradigm. We also show how it benefits subgraph discovery process by enhancing candidate generation mechanism.

CHAPTER 3

INTERNAL MECHANISMS: APRIORI PARADIGM

Both GDClust and GDClust-Plus use frequent subgraphs to represent senses common among the document-graphs. Two document-graphs, which contain some common frequent subgraphs, do not necessarily have to have common keywords. Our system not only looks at the original keywords but also the origin of the keywords and their neighboring (i.e. abstract) synsets. Two different words, leading to the same hypernym, have a good chance to generate two highly similar subgraphs, which reflects their shared sense. Our aim is to discover frequent senses rather than to look for frequent common keywords in the text documents. We utilize an Apriori paradigm for subgraph discovery in the document-graphs.

The association rule mining technique [8] presents a problem for discovering the associated items in a list of transactions. A sample transaction list is shown in Table 2. We can easily find that Bread is more likely to be associated with Peanut Butter. We can easily sort small lists. However, human judgment does not work that well when there are millions of transactions in the transaction-table. The aim of the Apriori algorithm is to

Table 2: A sample transaction table of market basket.

Transaction	Items
T1	Bread, Jelly, Peanut Butter
T2	Bread, Peanut Butter
T3	Bread, Milk, Peanut Butter
T4	Beer, Bread
T5	Beer, Milk

find frequent itemsets from a list of transactions. The algorithm concentrates on the corresponding supports of the items and itemsets. A support of *x%* for an association rule *R* means that *x%* of all the transactions under analysis show that items mentioned in *R* appear together in market baskets. The support of *{Bread, Peanut Butter}* from Table 2 is *60%* as Bread and Peanut Butter appear together in 60% of transactions.

In our work, we replace transactions with document-graphs, items with edges and item-sets with subgraphs (i.e., sets of connected edges). The association rule mining problem of market basket data analysis crops up in our research area in the form of frequent subgraph discovery problem. While discovering subgraphs, we take advantage of domain characteristics of the background knowledge for efficient processing (We discuss the domain characteristics of background knowledge later in this chapter). The advantages mainly emerge in the form of Master Document-Graph. This chapter illustrates the internal mechanism of our Apriori paradigm used in GDClust and its improved version proposed in GDClust-Plus.

Apriori Paradigm

Table 3 portrays the modified high-level algorithm for frequent subgraph discovery using the Apriori paradigm. The `find_frequent_1-edge_subgraphs` procedure utilizes the Dynamic minimum support strategy (explained in the next section) to select *1*-edge subgraphs from the document-graphs. The `apriori_gen` procedure in the algorithm joins and prunes the subgraphs. In the join operation, a *k*-edge candidate subgraph is generated by combining two *(k-1)*-edge subgraphs of L_{k-1}. This *k*-edge

Table 3: Apriori algorithm.

Input:
D: a database of document-graphs
min_sup: the minimum support threshold
Output:
L: frequent subgraphs in *D*
Method:

```
(1)   L1 = find_frequent_1-edge_subgraphs(D);
(2)   for (k=2; Lk-1≠Φ; k++){
(3)      Ck=apriori_gen(Lk-1);
(4)      for each document-graph g∈ D{
(5)         Cg= Ck∩g;
(6)         for each candidate subgraph s∈ Cg
(7)            s.count++;
(8)      }
(9)      Lk={ s∈ Ck | s.count ≥ min_sup}
(10)  }
(11)  return L=∪k Lk
```

subgraph becomes a member of L_k only if it passes the *min_sup* threshold. The details of this joining operation are described later in this chapter. In GDClust, we used FSG [12, 28] to generate potential candidates. In GDClust-Plus, this portion is enhanced, and we introduce a novel mechanism named Subgraph-Extension mining. Our major change to improve efficiency is incorporated in `apriori_gen` procedure. We explain Subgraph-Extension mining in section titled "Candidate Generation in GDClust-Plus: Subgraph Extension Mining".

Dynamic Minimum Support Strategy

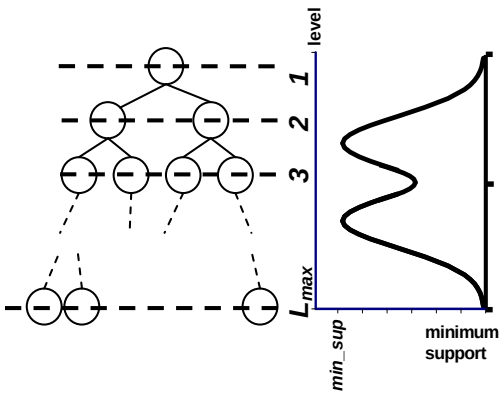
We use the WordNet ontology as background knowledge of the natural language domain. Since using the WordNet ontology results in a large graph of the whole language-domain used in our system, we introduce a Master Document-Graph (MDG)

and propose a Dynamic minimum support strategy in both GDClust and GDClust-Plus. Dynamic minimum support removes unnecessary edges so that they are not considered in the Apriori process. Unnecessary edges are those that appear in too many (or all) documents, as well as those that are too specific and appear rarely in the document-graphs.

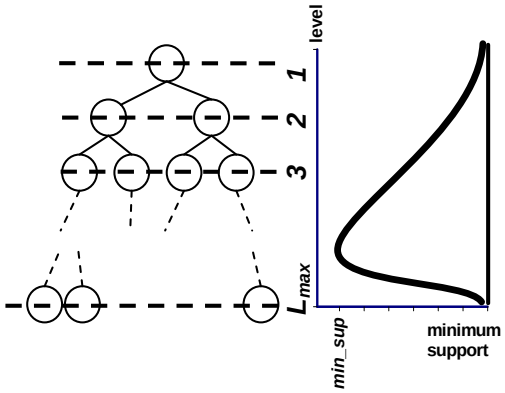
Motivation toward Gaussian Minimum Support Strategy

We observed that human-based communications tend to be conducted on common levels of abstraction. For instance, general day-to-day conversations contain neither too abstract nor too specific keywords. In contrast, lectures by professors and presentations by graduate students may contain specific keywords that are found at the bottom level of a natural language ontology. This establishes that the distribution of informative keywords is highly domain dependent.

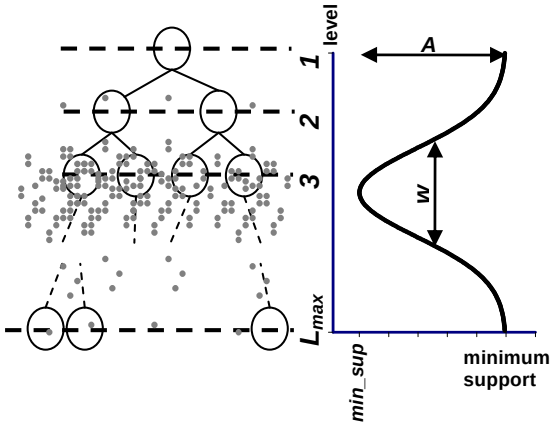
Figure 3(a) shows an example of abstraction-based distribution of keywords where keywords are concentrated at two levels of the ontology. This may indicate an ontology that has a complex *hypernymy-hyponymy* relation, in which it is hard to figure out a single level of communication. This type of behavior may be expected in popular magazine articles (e.g., the magazines of IEEE Communications Society), where some of the documents are general in notion but the others are specific. Figure 3(b) shows a document corpora with specific keywords. Research papers, scientific journals, technical books fall into this category.



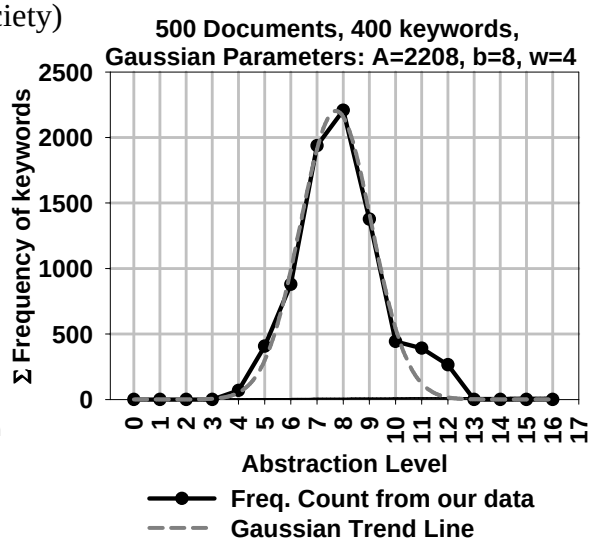
(a) Scientific and general topics (e.g., Magazines of IEEE Communications Society)



(b) Scientific Data



(c) General Topics



(d) Experimental result on 20-News Groups Dataset

Figure 3: Behavior of datasets and the relation with MDG taxonomy.

In our work, we used the *20* News Groups dataset [37] for all experiments. Figure 3(c) is a representation of our keywords in the context of WordNet's ontology. The gray dots indicate keywords found in our repository when matched to the levels of the ontology. It shows that the majority of the keywords are located at the mid-levels of the hierarchy, and very few keywords are placed at the bottom and top levels. Our experimental result on a subset of the 20 News Groups dataset (shown in Figure 3 (d)) supports our keyword-distribution theory. The black line of Figure 3(d) indicates that the

most frequent words are found at *level 8* of the WordNet ontology. Gaussian trend-line (the gray dashed line) can be compared with the black line to find a support for a Gaussian shaped distribution of keywords. Therefore, since our domain follows the Gaussian trend, we remove edges from the Apriori paradigm in a Gaussian fashion.

We use Gaussian minimum support strategy to limit the number of candidate subgraphs with extremely abstract and very specific meanings. Since WordNet's ontology merges to a single term (i.e., *"entity"*), the topmost level of abstraction is a common vertex for all the generated document-graphs. Because of this, subgraphs involving vertices from the top levels of abstraction will be less useful for clustering. Moreover, terms near the lowest level of abstraction are less important because they appear rarely in the document-graphs, and as a result, terms appearing in the intermediate levels of the taxonomy generate more representative clusters labels than subgraphs containing terms at high and low levels of abstraction.

Our Apriori paradigm imposes the minimum support to the *1*-edge subgraphs at the very beginning of the Apriori algorithm in Gaussian normalization fashion. It assigns different minimum support thresholds based on the term's abstraction level, and to do this assignment in less time, the paradigm uses the Master Document-Graph instead of WordNet. Each edge of the Master Document-Graph is ranked according to the levels in WordNet taxonomy. Currently, WordNet has *18* abstraction levels in its noun taxonomy, but the edges of the Master Document-Graph do not have to cover all the levels. Therefore, the maximum abstraction level in the Master Document-Graph is bounded by $l_{max} \leq 18$.

The Gaussian function possesses a shape that matches our criteria. It has a smaller minimum support for the terms located at the intermediate levels, and the function assigns higher minimum support thresholds to terms located at the lower and higher levels of the Master Document-Graph. The approach makes the mid-levels of the taxonomy formed by Master Document-Graph more important. It also assumes, based on observation, that the generated document-graphs contain a lot of common, but uninteresting, subgraphs at the topmost level and distinct, but not frequent, subgraphs at the bottom levels. The first would generate large clusters with low inter-cluster similarity, and the second would generate a huge number of very small clusters.

The Gaussian function can be defined as:

$$f(x) = Ae^{-(x-b)^2 \div 2c^2} \tag{3.1}$$

where *A* is the height of the Gaussian peak, *b* is the position of the center of the peak and *c* is defined as:

$$c = \frac{w}{2\sqrt{2\ln(2)}} \tag{3.2}$$

where *w* is the width of the curve at *A/2*. In our case, $b = l_{max} / 2$. We apply this behavior to model the minimum support of mining multilevel senses from the WordNet taxonomy. This is illustrated in Figure 4 (Figure 3(c) is repeated in Figure 4). The hierarchy drawn in the figure indicates our Master Document-Graph. The Gaussian graph indicates that the varying minimum support threshold is largest at the highest and lowest levels (i.e., level *1* and level l_{max}). The model generates our pre-defined minimum support, *min_sup* only at the mid level of the taxonomy and applies a gradual increment

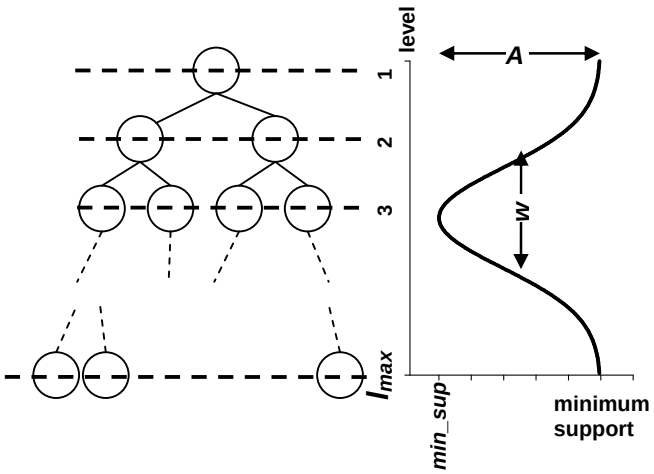


Figure 4: Gaussian minimum support strategy for multilevel mining.

of minimum support at higher and lower levels. One can shift the *min_sup* value to other levels by changing *b* of equation (1). Moreover, more *1*-edge subgraphs can be removed from Apriori's candidate list by reducing *w* to make the curve narrower. The impacts of different values of *min_sup* and *w* on subgraph mining are explained in Chapter 6.

Gaussian Minimum Support Strategy and Motivation toward Data Dimensionality

In a document-graph, some edges are directly linked with *keyword-synsets* (since at least one of its vertices contains the keyword-synset) while most are not. Although our Gaussian minimum support strategy removes edges from *1*-edge candidate list (L_1) in the `find_frequent_1-edge_subgraphs` procedure of the Apriori paradigm (Table 3), the aim is not to prune edges related to keywords but rather to emphasize all edges near frequent keywords. This pruning physically changes neither the document-graphs nor the Master Document-Graph. Dynamic minimum support strategy is used to prune edges from *1*-edge candidate list (L_1), just before feeding L_1 to the next iteration of the Apriori algorithm. In all subsequent iterations of Apriori, the minimum support threshold

is no longer dynamic. Rather, it becomes a constant *min_sup* (this is the lowest minimum support threshold of Figure 4). All the edges pruned from L_1 in `find_frequent_1-edge_subgraphs` never become a part of the higher-order candidate lists (L_n where $n>1$). We can quickly perform the pruning operation on L_1 because our edges in L_1 have incorporated information about their corresponding abstraction level in the Master Document-Graph. Details of this procedure are described in chapter 5 (section title: "*LabeledEdge* Object").

The pruning mechanism works in such a way that the edges at the upper part of the Master Document-Graph are forced higher minimum support than the edges representing a moderate degree of abstraction. Similarly, an edge with extremely high specificity is also forced a higher minimum support than an edge with moderate abstraction. This does not disqualify such edges completely, but it implies that the edges with high and low abstraction values must appear in more document-graphs to be considered as candidates. Because an edge with a moderate (mid-level) abstraction value has the lowest minimum support threshold, it can appear in fewer document-graphs and

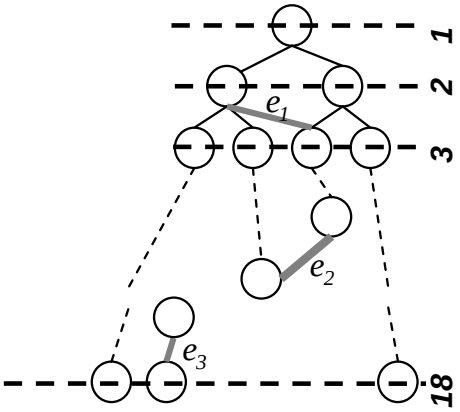


Figure 5: Illustration of edges at different depth of Master Document-Graph.

still be recognized as frequent. For example, consider the three edges e_1, e_2 and e_3 in L_1(Figure 5). Using an object-oriented paradigm, we can present an edge's abstraction level as `depth`. Assume $e_1.depth = 2$, $e_2.depth = 8$ and $e_3.depth = 17$. Also consider that, after applying the Gaussian minimum support strategy on L_1, the imposed minimum supports of these three edges become $e_1.minimum\_support = 98\%$, $e_2.minimum\_support = 5\%$ and $e_3.minimum\_support = 97\%$. If there are *200* document-graphs, then e_1 must appear in at least *196* document-graphs, e_2 must appear in at least *10* document-graphs and e_3 must be an edge of at least *194* document-graphs. This way, we get rid of the too abstract and too specific edges of L_1.

Now, after pruning *1*-edge candidates, we may be left with no edges directly linked with keyword-synsets. But we have at least some edges from the mid-levels, even if the keyword-synset is at the bottom. We can say this because the construction of Master Document-Graph ensures its *monotonic* behavior. We call our Master Document-Graph monotonic in terms of support count of the edges because an edge *a* connected

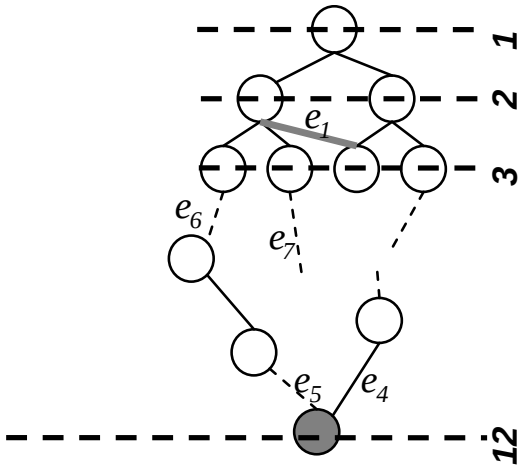


Figure 6: A document-graph containing only one keyword-synset.

(directly or not) to an edge *b* at the lower abstraction level always has a support higher, or at least equal, to the one represented in *b*. Hence, removing a keyword-edge *b* at the bottom of the tree would at least keep its abstraction *a* in midlevel. We show an example in Figure 6, where we consider that the document-graph has only one keyword-synset indicated by gray circle. For this document-graph, although edges that are directly attached to the keyword-synset (e.g., e_4 and e_5) may not appear in L_1 after the Gaussian minimum support strategy is applied, (because they have been imposed high minimum support threshold i.e., they must appear in a higher number of document-graphs to pass Gaussian-based threshold), some edges from the mid-levels (e.g., e_6, e_7, etc) will still survive in L_1 because they must have the lowest minimum support threshold (i.e., *min_sup*) and Master Document-Graph is monotonic. We refer to *min_sup* as the user-provided minimum support parameter indicated in Figure 4. This is generally the static minimum support used in regular bag-of-token-based approaches. If someone ensures that a keyword will survive in a vector-based representation of a document with a minimum support *min_sup* using the regular approach, then we can ensure that at least some edges will survive from the mid-levels for this keyword. This is true for two reasons:

1. The minimum support at exactly mid-level is *min_sup*
2. The monotonic behavior of Master Document-Graph ensures that the usual *support count* of an edge at mid-level (and for top level) is greater or equal to the usual *support count* of an edge at lower level.

Now, we summarize three cases:

1. For very specific keyword-synsets, we ensure that at least some edges would be picked up by Gaussian minimum support strategy if that keyword was considered frequent in traditional bag-of-tokens-based approach. From the experiment in Figure 3(d) we can see that keywords at the bottom levels are mostly infrequent. Previous arguments on the monotonic property of Master Document-Graphs show that if an infrequent keyword is frequent enough to pass the *min_sup* of the Apriori paradigm, even if its directly connected edge is removed from L_1, there will be at least one edge left in L_1 from mid-level of the MDG that represents its higher level of abstraction. Hence, Gaussian minimum support automatically picks mid-level abstractions of infrequent keyword-synsets.
2. For edges that are at mid-levels, Gaussian minimum support strategy applies minimum support near to *min_sup*, so that their candidacy becomes something like in the traditional approach. Figure 3(d) also shows that most frequent keyword-synsets are found at the mid-levels. So, Gaussian minimum support strategy directly keeps keyword-edges in L_1, where the keyword-synsets are found in mid-levels, and Figure 3(d) ensures that keywords are most frequent at mid-levels.
3. For very abstract keywords, we do not guarantee that any edge with or without a keyword-synset attached will appear in L_1 after we apply the Gaussian minimum support strategy. This is because, in our case, the abstract edges with attached keyword-synsets are poor candidates for clustering because of the result of the monotonic behavior of our MDG where the edges linked with abstract keywords tend

to appear in a majority of our document-graphs (sometimes, even in all the documents).

Candidate Generation

In this section, we describe the candidate generation mechanisms of GDClust and GDClust-Plus. GDClust-Plus outperforms GDClust due to its efficient candidate generation technique which is based on Subgraph-Extension mining.

Candidate Generation Mechanism in GDClust

The document-graph construction algorithm (Table 1) ensures that each document-graph does not contain more than one direct edge between two vertices. Additionally, the overall sense discovery concept ensures that a subgraph does not appear more than once in an individual document-graph. In our case, all the edges and vertices of a document-graph are uniquely labeled. We generate a *k*-edge candidate subgraph by combining two (*k-1)*-edge subgraphs where these two (*k-1)*-edge subgraphs have a common *core subgraph* [12] of *(k-2)*-edges. In GDClust, each *k-edge subgraph object* is composed of a connected edge-list and a list of edges that generated this *k*-edge subgraph from a *(k-1)*-edge subgraph. Consider the *6*-edge subgraph *lmnopq* of Figure 7 that has

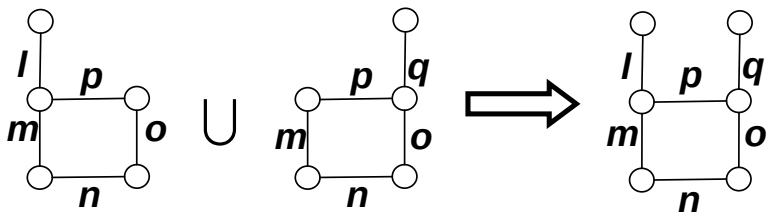


Figure 7: Generation of a *6*-edge subgraph from two *5*-edge subgraphs.

been generated from two *5*-edge subgraphs *lmnop* and *mnopq* (each of the *5*-edge subgraphs having a common core *mnop*).

GDClust requires multiple comparisons between core subgraphs while generating a higher order subgraph. To avoid the cost of comparison between each edge of subgraphs, GDClust assigns a unique code for each subgraph from the list of their edges. This code is stored as the *hash-code* of the subgraph object. Section "*Subgraph* Object" of Chapter 5 explains how hash-codes are generated for a subgraph object. Therefore, checking two core subgraphs to see whether they correspond to each other is just an integer hash-code comparison. The resulting candidate subgraphs, maintaining the minimum support, are chosen for the next iteration of the Apriori algorithm.

Now, we describe an example to better illustrate the high cost of the *Core-subgraph strategy of GDClust*. Consider an instance in which we have a total of *21 5*-edge subgraphs in the candidate list L_5. We would try to generate a *6*-edge subgraph from this list. Consider the situation of generating candidates using one *5*-edge subgraph (e.g., *lmnop*) of L_5. The FSG approach of our original GDClust tries to combine all other *20* subgraphs with *lmnop* but succeeds, let us assume, only in three cases. Figure 8 illustrates that *lmnop* is successfully combined with only *mnopq*, *mnopr* and *mnops*. All *17* other attempts to generate a *6*-edge subgraph with *lmnop* fail because the *4*-edge core-subgraphs, analyzed in this case, do not match. Among these *17* failed attempts, only the attempt with *tmnoz* is shown in Figure 8 with the label "Not Generated". The rest of the failed attempts are indicated by dotted lines.

The instance of Figure 8 is depicted only for one subgraph (*lmnop*). For all these *5*-edge subgraphs of L_5, there would be a total of $21 \times 20 = 420$ blind attempts to generate *6*-edge subgraphs. Some of these attempts would succeed, but most would fail to generate acceptable *6*-edge candidates. Although the original GDClust [1] utilizes hash-codes of subgraphs and core-subgraphs for faster comparisons, it cannot avoid comparing a large number of hash-codes for all candidates. In GDClust-Plus, we smartly reduce the number of attempts to combine subgraphs by applying our Subgraph-Extension Mining technique, which is described in the following sub-section.

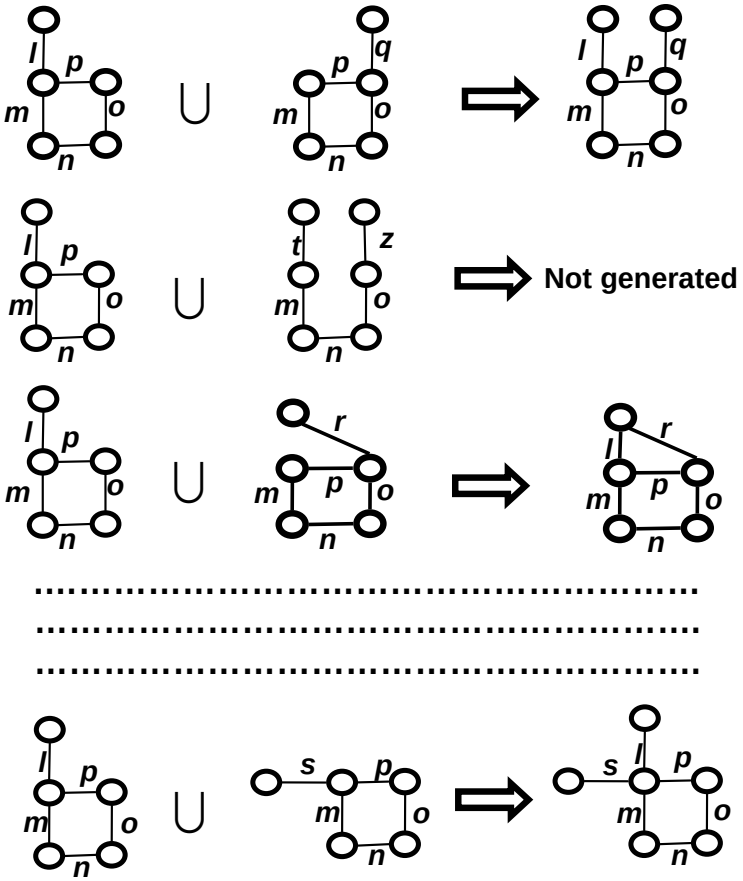


Figure 8: Attempts to combine *lmnop* with other *5*-edge subgraphs of (L_5).

Candidate Generation in GDClust-Plus: Subgraph Extension Mining

Rather than trying a brute-force strategy of all possible combinations, like in Frequent Subgraph Mining (FSG) [28], in GDClust-Plus we use the Master Document-Graph *(MDG)* as the source of background knowledge to reduce number of attempts needed to generate a *k*-edge subgraph from *(k-1)*-edge subgraphs. The Master Document-Graph ensures that an extension of a *k*-edge subgraph can generate a *(k+1)*-edge subgraph. We maintain a neighboring-edges' list for each *(k-1)*-edge subgraph (*see* "*Subgraph* Object" section of Chapter 5) and try to generate candidates for frequent higher order subgraphs by taking edges only from this neighboring-edges' list. The neighboring-edges' list of a subgraph contains only those edges that passed Dynamic minimum support strategy in `find_frequent_1-edge_subgraphs` procedure of Apriori Algorithm (Table 3), which further reduces the unnecessary generation of higher order subgraphs that will not pass *min_sup* of step 9 of the Apriori Algorithm.

Figure 9 shows the Subgraph-Extension mechanism for subgraph *lmnop*, which can be compared with FSG approach of Figure 8. The gray edges of Figure 9 indicate the subgraph subject toward an extension, while a black edge indicates an extension of the gray subgraph maintained in our MDG. The same instance is used for both Figure 8 and Figure 9. The neighboring-edges' list of *lmnop* contains edges *{q, r, s}*. Unlike in Figure 8, in the (GDClust-Plus) example presented in Figure 9, the new Subgraph-Extension mining strategy does not try to generate higher order subgraphs *20* times. Rather, it tries only three times, using the knowledge about neighboring edges of *lmnop* in MDG. This results in only three attempts to generate higher-order candidate subgraphs, and none of

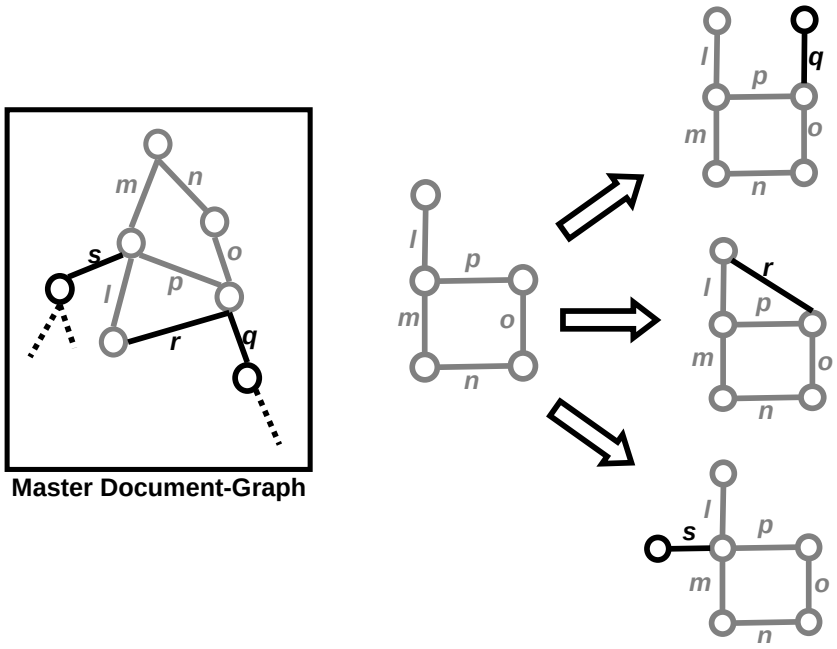


Figure 9: Subgraph-Extension Mining of subgraph *lmnop* for GDClust-Plus.

these attempts fails at step 3 of the Apriori Algorithm (Table 3) because the mechanism depends on the physical evidence of possible extension. Therefore, the Subgraph-Extension mining strategy in GDClust-Plus offers a novel knowledge-based mechanism that eliminates unnecessary comparisons (of core subgraphs) or attempts to combine subgraphs.

A Performance comparison between GDClust and GDClust-Plus is shown in Chapter 6 (section title: “GDClust vs. GDClust-Plus”).

CHAPTER 4

INTERNAL MECHANISMS: CLUSTERING

This chapter describes the clustering mechanism which we used both with GDClust and GDClust-Plus. The first section of this chapter describes our clustering technique, and the second section presents the similarity measures used during our clustering experiments. We also present traditional clustering mechanisms and explain our measure for evaluation of clusters' quality in this chapter.

Hierarchical Agglomerative Clustering (HAC)

GDClust and GDClust-Plus use Hierarchical Agglomerative Clustering (HAC) [9] to group documents. The clustering unit (Figure 2) assigns unique identification to frequent subgraphs retrieved by the Apriori algorithm (Table 3). Information about document-graphs is stored in a hash table where the key is the document name. The corresponding value against a key contains a vector of the subgraphs' identification numbers, which appear in the corresponding document-graph. The hash table does not contain the frequent subgraphs; rather it contains only frequent subgraphs' identification numbers against each unique document name.

The clustering unit (Figure 2) constructs a dissimilarity matrix that stores dissimilarities between every pair of document-graphs. Dissimilarity between a pair of document-graphs G_1 and G_2 is measured using the formula:

$$d=1.0-similarity \tag{4.1}$$

The similarity values range between *[0, 1]* for all the measures we used. We describe the similarity measures used for our experiments in the following section.

Measures of Similarity between Document-Graphs

Similarity between two document-graphs can be found using different measures. We performed our experiments using the following graph similarity measures:

1. Matching coefficient
2. Jaccard coefficient
3. Dice coefficient
4. Cosine coefficient
5. Overlap coefficient.

Among these five types of similarity measures, Matching Coefficient is the simplest. It counts only the number of common subgraphs in two document-graphs. More formally:

$$sim(G_1, G_2)_{Matching} = count(FSG(G_1) \cap FSG(G_2)) \quad (4.2)$$

We normalized it to the range *[0, 1]*. It is clear that this similarity measure concentrates on the matched subgraphs only. If two document-graphs have a large number of matches with a large number of mismatches, the matching coefficient only counts the matches and totally ignores the mismatches. Matching coefficient can be used in the test where the number of mismatches is negligible. In consequence, it does not convey enough significance in document clustering domain, because two large document-graphs can possess lots of matching subgraphs but an equal number of mismatching subgraphs as

well. Hence, ignoring the number of mismatches does not allow for proper similarity evaluation of two documents (especially when at least one of them is of a large size).

Since both the common subgraphs and uncommon subgraphs are important, we need other similarity measures. Now, we describe Jaccard Coefficient [42, 43] that takes both numbers of matches and mismatches into account. The formula is:

$$sim(G_1, G_2)_{Jaccard} = \frac{count(FSG(G_1) \cap FSG(G_2))}{count(FSG(G_1) \cup FSG(G_2))} \quad (4.3)$$

where $FSG(G_1)$ and $FSG(G_2)$ are the sets of frequent subgraphs that appear in document-graph G_1 and G_2 respectively. The dividend of equation (4.3) is the number of common frequent subgraphs between document-graphs G_1 and G_2, and the divisor is the total number of unique frequent subgraphs in G_1 and G_2.

Although the Jaccard coefficient takes the number of mismatches into consideration in some way, it can strongly panelize these document-graphs which have small number of common subgraphs but also a much bigger number of subgraphs which do not match. We analyze and compare this similarity measure with the Dice coefficient [43, 44], which normalizes the number of common subgraphs by the total number of subgraphs in two documents:

$$sim(G_1, G_2)_{Dice} = \frac{2 \times count(FSG(G_1) \cap FSG(G_2))}{count(FSG(G_1)) + count(FSG(G_2))} \quad (4.4)$$

As a result of this normalization, the Dice coefficient does not penalize the document-graphs with a small number of common subgraphs as harshly as Jaccard coefficient.

To illustrate the effect of this normalization we show a simple example in Table 4. Let us assume that we have two document-graphs G_1 and G_2. Each of these document-graphs has a total of *10* frequent subgraphs, i.e., $count(FSG(G_1)) = 10$ and $count(FSG(G_2)) = 10$. Let us now consider cases with different numbers of common subgraphs occurring in both the document-graphs. We show that the Jaccard coefficient signifies lower similarity between G_1 and G_2 than the Dice coefficient. Table 4 shows the results of our analysis. It shows that for any number of common subgraphs, except when the number is *10*, the Jaccard coefficient is smaller than the Dice coefficient, and the percentage of difference between the Dice and Jaccard coefficients is highest (*47.4%*) when the number of common subgraphs is lowest (i.e., only one common subgraph). The percentage difference gradually decreases as the number of common subgraphs in both document-graphs increases. Therefore, the Jaccard coefficient penalizes document-graphs with smaller number of common subgraphs more than the Dice coefficient does. When

Table 4: Differences between Dice and Jaccard coefficients.
($count(FSG(G_1)) = 10$ and $count(FSG(G_2)) = 10$).

Number of common subgraphs	**Jaccard**	**Dice**	**Difference (in %)** $(\frac{Dice - Jaccard}{Dice} \times 100)$
1	0.0526	0.10	47.4
2	0.1111	0.20	44.4
3	0.1765	0.30	41.2
4	0.2500	0.40	37.5
5	0.3333	0.50	33.3
6	0.4286	0.60	28.6
7	0.5385	0.70	23.1
8	0.6667	0.80	16.7
9	0.8182	0.90	9.1
10	1.0000	1.00	0.0

number of common subgraphs tends to rise, the difference between Dice and Jaccard coefficient has a tendency to get smaller.

The Cosine coefficient becomes the same as the Dice coefficient only for document-graphs with identical number of frequent subgraphs. Plus, it penalizes the similarity value of two document-graphs less severely when the numbers of frequent subgraphs are very different. Due to the better stability of similarity values even when comparing documents with significantly different sizes, the Cosine coefficient is commonly used in many text-clustering applications. Also, in our case, we do not want to make two document-graphs dissimilar based only on the property that one of them has few frequent subgraphs compared the other. The Cosine coefficient [45] for binary vectors is defined as follows:

$$sim(G_1, G_2)_{Cosine} = \frac{count(FSG(G_1) \cap FSG(G_2))}{\sqrt{count(FSG(G_1)) \times count(FSG(G_2))}} \quad (4.5)$$

Table 5 shows that the Dice coefficient is smaller than the Cosine coefficient except, except when there are an equal number of frequent subgraphs in G_1 and G_2. We

Table 5: Differences between Cosine and Dice coefficients.

Number of common subgraphs	***count* (*FSG*(G_1))**	***count* (*FSG*(G_2))**	**Dice**	**Cosine**	**Difference (in %)** $(\frac{Cosine - Dice}{Cosine} \times 100)$
50	50	500	0.1818	0.3162	42.5
50	100	500	0.1667	0.2236	25.5
50	150	500	0.1538	0.1826	15.7
50	200	500	0.1429	0.1581	9.6
50	250	500	0.1333	0.1414	5.7
50	300	500	0.1250	0.1291	3.2
50	350	500	0.1176	0.1195	1.6
50	400	500	0.1111	0.1118	0.6
50	450	500	0.1053	0.1054	0.1
50	500	500	0.1000	0.1000	0.0

typically see that the higher the difference between the numbers of frequent subgraphs in G_1 and G_2, the higher the percent difference between the Cosine and Dice coefficients. Hence the Dice coefficient panelizes more harshly than the Cosine coefficient when the difference between the numbers of frequent subgraphs in the compared document-graphs is substantial. Both the coefficients tend to generate close similarity values when the difference between the numbers of frequent subgraphs in compared documents is small. It should be mentioned that we kept the number of common subgraphs fixed to *50* in Table 5, to make sure that the corresponding dividends of Dice and Cosine do not change due to varying number of common subgraphs. This ensures that all the presented changes in Dice and Cosine coefficients are only outcomes of varying number of frequent subgraphs in G_1.

For experimental purpose, we also show another measure named Overlap coefficient [46] of similarity. The value of the Overlap coefficient reaches the maximum when every frequent subgraph of one document-graph appears in the set of frequent subgraphs of the other document-graph. The formula for Overlap coefficient is:

$$sim(G_1, G_2)_{Overlap} = \frac{count(FSG(G_1) \cap FSG(G_2))}{MIN(count(FSG(G_1)), count(FSG(G_2)))} \quad (4.6)$$

Therefore, it reflects inclusion property of subgraphs of one document-graph in another. In Chapter 6, we show using experimental results how inaccurate the clustering becomes using the Overlap coefficient in our system.

Chapter 6 (section title: “Clustering Accuracy with Different Similarity Measures”) shows experimental results of our clustering for different similarity measures.

Traditional Document Clustering Mechanism for Evaluation

There are direct bag-of-token based approaches for document clustering. We compared our system against the ordinary vector model representation of documents. We can construct the dissimilarity matrix for our Hierarchical Agglomerative Clustering using the classic Cosine coefficient, reflecting the angle between every pair of documents represented as vectors of tokens. The simplest way to measure the Cosine similarity between two document vectors D_i and D_j is to utilize the frequencies of the keywords without any kind of normalization. The formula is:

$$similarity(D_i, D_j)_{Cosine} = \frac{\sum_{t=1}^{T}\left(f_{it} \times f_{jt}\right)}{\sqrt{\sum_{t=1}^{T}\left(f_{it}^2\right)\sum_{t=1}^{T}\left(f_{jt}^2\right)}} \tag{4.7}$$

where f_{it} indicates the frequency of the *t*-th keyword in document D_i, and f_{jt} indicates the frequency of the *t*-th keyword in document D_j .

From the previous section, we know that Cosine coefficient is not influenced by one document being small compared to the other, yet the frequencies of the keywords in equation (4.7) do not portray any parameter to emphasize how important a keyword is to the corresponding document in essence to the entire document archive. The *term frequency-inverse document frequency (tf-idf)* is a weight used in text mining for this purpose. To get the *term frequency* tf_{it}, the count f_{it} is normalized by the total number of important keywords in document D_i to prevent bias toward longer documents to give a measure of the importance of the *t*-th keyword,

$$tf_{it} = \frac{f_{it}}{\sum_k f_{ik}} \tag{4.8}$$

The *inverse document frequency* idf_t is a measure of the general importance of the *t*-th keyword. It is usually obtained by dividing the number of all documents by the number of documents containing the keyword, and taking the natural logarithm of the quotient. Therefore, the idf_t of the *t*-th keyword is:

$$idf_t = ln\left(\frac{N}{\left|\{D_i : keyword_t \in D_i\}\right|}\right) \tag{4.9}$$

where *N* is the total number of documents in the archive and $\left|\{D_i : keyword_t \in D_i\}\right|$ indicates the number of documents where *t*-th keyword has frequency $f_{it} > 0$. *tf-idf* of the *t*-th keyword in document D_i is simply measured by multiplying the corresponding *tf* and *idf* :

$$tfidf_{it} = tf_{it} \times idf_t \tag{4.10}$$

For better results, equation (4.7) can be rearranged using *tf-idf* in the formula:

$$similarity(D_i, D_j)_{Cosine} = \frac{\sum_{t=1}^{T}\left(tfidf_{it} \times tfidf_{jt}\right)}{\sqrt{\sum_{t=1}^{T}\left(tfidf_{it}^2\right)\sum_{t=1}^{T}\left(tfidf_{jt}^2\right)}} \tag{4.11}$$

The corresponding experimental results on quality of the clustering using the classical Cosine measures are portrayed in Chapter 6.

Measuring Clustering Accuracy: Silhouette Coefficient

In many popular clustering algorithms, the number of expected clusters *n* is an input parameter provided by the user. To evaluate the quality of clustering at the given

number of clusters *n*, one can use silhouette coefficient [47, 48]. It is also possible to find out a number of clusters *n*, for which the average silhouette coefficient has the highest value *H*, by simulating the clustering with different numbers of output clusters $2 \le n \le N$, where *N* in our case is the total number of document-graphs.

Assume that the cluster to which object *i* is assigned is denoted as *A*. Let *a(i)* be the average dissimilarity of *i* to all other objects of the cluster *A*. For any cluster *C* different from *A*, let *d(i,C)* be the average dissimilarity of *i* to all objects of *C*. After computing *d(i,C)* for all available clusters (except *A*), the smallest average dissimilarity denoted as $b(i) = \min_{C \neq A} [d(i,C)]$, is selected. The silhouette coefficient of object *i*, denoted as *S(i)*, is then obtained by combining *a(i)* and *b(i)* as follows:

$$S(i) = \begin{cases} 1 - \dfrac{a(i)}{b(i)}, & if \quad a(i) < b(i) \\ 0, & if \quad a(i) = b(i) \\ \dfrac{b(i)}{a(i)} - 1, & if \quad a(i) > b(i) \end{cases} \tag{4.12}$$

Silhouette coefficient of an individual cluster is the average of silhouette coefficient *S(i)* for all the elements assigned to this particular cluster [48]. An overall measure of goodness of a clustering can be obtained by computing the average silhouette coefficient of all data points [49]. We get the most natural number of groupings by looking at the number of clusters for which there is a peak in the plot reflecting average silhouette coefficient. Experimental results on this evaluation mechanism are given in Chapter 6.

CHAPTER 5

INTERFACE IMPLEMENTATION

This chapter describes some important classes that we have designed, and some pre-existing API that help us to interface GDClust and GDClust-Plus with WordNet.

Object-Oriented Paradigm

The GDClust system has been built using Object Oriented Programming with a mechanism allowing to reduce the task of subgraph comparison to an integer comparison only. While comparing two subgraphs, GDClust does not require comparing each edge of those subgraphs; rather it requires comparing only two hashcodes. The hashcodes are generated from the list of DFS-codes of the edges of a subgraph when it is created (*see* section titled "Construction of Document-Graph and Master Document-Graph" of Chapter 2 for the generation of DFS-codes).

EdgeLabel Object

The `EdgeLabel` class is defined to maintain the basic information about an edge. An instance of this class stores information like source, target, frequency of the edge, etc. In our system, the source and the target are two different synsets of WordNet. A `LabeledEdge` of a document-graph or Master Document-Graph contains an `EdgeLabel` object. This indicates that all graphs under consideration have directed and labeled edges. Although the frequencies of the edges are not used in our approach, we kept a `frequency` field for future use.

LabeledEdge Object

Each instance of the `LabeledEdge` class represents the actual edge of a graph in GDClust system. A document-graph or the Master Document-Graph is an instance of `DirectedGraph<LabeledEdge>` object. Therefore, an edge of a `DirectedGraph` is a `LabeledEdge` and the label of that edge is an `EdgeLabel` object. The `LabeledEdge` class has several fields among which the followings are very important:

1. `boolean edgeStatus;` *// true if forward edge, false if backward edge*

2. `boolean isEdgeTraversed = false;`
   / Facilitates DFS traversal by checking the pre-traversed edges while generating DFS-Tree */*

3. `double depth = 0.0;`
   / To facilitate Gaussian minimum support strategy. This depth is dependent on the MDG, so when we use WordNet it should not be greater than 18. depth = 0 means that the edge has not been yet traversed for generating DFS-code. */*

The value of `depth` of the `LabeledEdge` informs about the abstraction level of an edge in the ontology and in our case $depth \leq 18$ as WordNet has a maximum of *18* levels in its IS-A noun taxonomy. It is hard to distinguish the accurate abstraction level of a keyword in the taxonomy because of multiple senses of the keyword. For example, in Figure 10, the abstraction level of the keyword *F* can be *2*, *3* or *4*. In our investigation, we have used the ceiling value of the average depth of *F*. So the average depth of *F* is $\frac{2+3+4}{3} = 3$. The depth of a `LabeledEdge` is determined from the average depths of the source and the target synset of this particular edge.

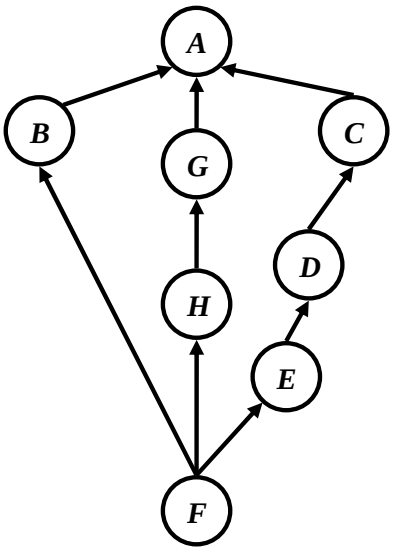


Figure 10: Hierarchy of a keyword-synset *F*.

Subgraph Object

The `Subgraph` class represents a subgraph in GDClust system. Most of the methods of the `Subgraph` class are used to maintain and retrieve two of its private fields. The fields are:

1. `HashSet subgraphEdgeIDsHash` and
2. `HashSet documentSuppHash`.

`subgraphEdgeIDsHash` is a simple `HashSet` that stores the DFS-codes of the edges of `this` subgraph. Since the DFS-codes of the edges of each subgraph are stored in a `HashSet`, comparison between two subgraphs can be even minimized to just one integer. We implemented such mechanism, so we do not have to even check each edge of the two subgraphs. For this purpose, we take the advantage of object-oriented facility and override the `hashCode()` method. In general, Java Virtual Machine (JVM) imposes a unique hashcode for each object running under it. JVM imposes this hashcode only if the `hashCode()` method is not overridden. Therefore, for each `Subgraph` object, if we

do not override the `hashCode()` method, we have a hashcode provided by the virtual machine for each subgraph object. Now, even if we have the same list of edges in two subgraphs, due to any other temporary facilitating fields of the `subgraph` class, JVM would impose different hashcodes for these two subgraphs. To avoid this, we needed to override the `hashCode()` method inside `Subgraph` class. We did it in such a way that the hashcode of the subgraph is dependent on the `HashSet` of the DFS-codes of the edges only. Therefore the equality comparison between two subgraphs hash been reduced to the comparison of two hashchodes of the two corresponding `HashSet`s. The overridden `hashCode()` method of our `Subgraph` class is as follows:

```
public int hashCode(){
     return subgraphEdgeIDsHash.hashCode();
}
```

The `equals()` method of `Subgraph` class uses the `hashCode()` in the following way:

```
public boolean equals(Object o){
     Subgraph s = (Subgraph) o;
     HashSet hs= s.getSubgraphEdgeIDsHash();
     return
        subgraphEdgeIDsHash.hashCode()==hs.hashCode();
}
```

A subgraph object of GDClust-Plus maintains another important `HashSet` named `neighbouringEdgeIDsHash`. This `HashSet` contains important information about the neighborhood edges of a subgraph in Master Document-Graph to facilitate the Subgraph-Extension mining mechanism.

Interface with WordNet

Many researchers over the whole world are working on different projects for data mining and considering WordNet as their reference of lexical database when required. As a result, a vast number of interfaces and APIs were developed in the past years to retrieve data from the WordNet dictionary. Among them .NET, COM, Java, Haskell, Lisp, OCaml, Palm, Perl, PHP, Prolog, Python and Ruby interfaces and APIs are being mentioned most frequently. WordNet was developed by Cognitive Science Laboratory of Princeton University [2], they provide necessary libraries (library functions are described in Section 3 of [50]) and the API to access the WordNet dictionary using C++. The interface header and the library functions are available as `wn.h` and `wn.lib` with the distribution package. Besides, there are also Java APIs like JWNL [51] and JWordnet [52], which can be used to retrieve our WordNet ontology from WordNet lexical reference system.

Android Technologies, Inc. provides a MySQL version of the WordNet converted from the Prolog files of the WordNet database. They took the Prolog database format of WordNet 2.0 files and converted them to MySQL batch script format. So researchers can import the files in MySQL and use them as needed. Android Technologies also provide the translation documentation for Prolog format to MySQL table format [53]. The MySQL version of WordNet raised interest among many programmers due to the simplicity of retrieval of needed ontologies directly from the dictionary database.

WNSQLBUILDER [54] is a Java tool from Sourceforge project to build the SQL database from the WordNet releases. The project also provides MySQL and PostGreSQL ready-to-use versions of the WordNet databases.

The latest Windows version of WordNet is WordNet2.1. In GDClust, we used a low level interface named WordNet Java Native interface (WNJN [55]) to communicate with the WordNet dictionary. WNJN is able to communicate with the latest Win32 version of WordNet. Moreover, since the WNJN uses low level platform dependent C++ codes, the interface is fast. Another advantage of WNJN is that it uses the original data of WordNet without any modification. Therefore, we have chosen WNJN as a bridge between GDClust and WordNet.

JGraphT

JGraphT is a Java graph library that provides mathematical graph-theory objects [56]. JGraphT supports various types of graphs including:

1. directed and undirected graphs
2. graphs with weighted, unweighted, labeled or any user-defined edges
3. various edge multiplicity options including: simple-graphs, multigraphs, pseudographs
4. unmodifiable graphs, which allow modules to provide “read-only” access to internal graphs
5. listenable graphs, which allow external listeners to track modification events
6. all compositions of the above mentioned graphs.

Although powerful, JGraphT is designed to be simple. For example, graph vertices and edges can be of any objects. We took this facility and incorporated `LabeledEdge` in the graphs for our document-graphs and Master Document-Graph. A JGraphT graph takes up very little room in the memory, and it is possible to create stream of graphs by overriding some of the original JGraphT methods. So, one can handle graphs even with a few million vertices and edges, and the graph objects can also be stored on disk, since they can be serialized. GDClust only utilizes the data structure of JGraphT for document-graphs and the Master Document-Graph. Thus, although GDClust handles thousand of document-graphs with thousands of edges, the memory and disk usage is very efficient.

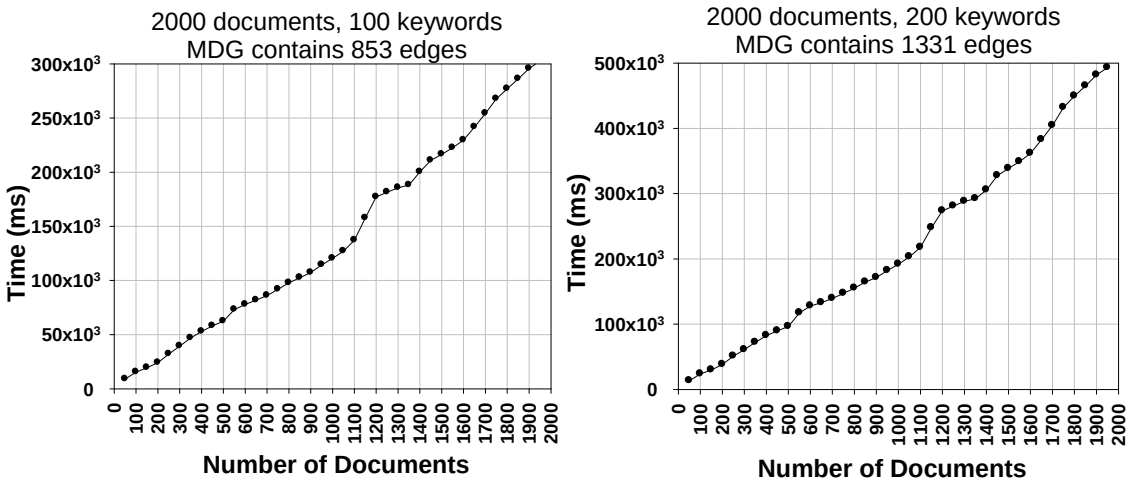
CHAPTER 6

EXPERIMENTAL RESULTS

The majority of our experimental results can be found in this chapter. The machine we used to execute all our experiments had an Intel Pentium 4 CPU (*2.0* GHz) and 1GB of RAM running under Windows XP. All our experiments are conducted on the *20* News Groups dataset [37], which is regarded as a benchmark collection of data for natural language processing.

Document-Graph Construction

Figure 11 shows the scalability of our document-graph construction algorithm (which is depicted in Table 1). It shows that the algorithm performs its execution in linear fashion with an increasing number of documents. In the experiment shown in Figure 11(a), a total of *100* unique keywords with highest information gain were selected from a



(a) *100* Keywords **(b) *200* Keywords**

Figure 11: Document-graph construction time for *2000* documents.

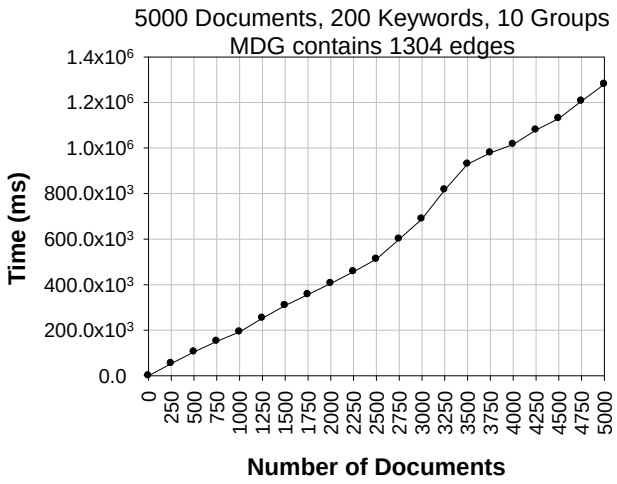


Figure 12: Document-graph construction time for *5000* documents.

maximum of *2000* documents. Figure 11(b) shows the experiment with *200* keywords from the same set of *2000* document. Both experiments show that the behavior is linear. The corresponding MDG of Figure 11(a) contained *853* unique edges whereas the MDG related to Figure 11(b) contained *1331* edges. Database *D* of the Apriori algorithm of Table 3 contains the generated *2000* document-graphs.

For most of the experiments in this chapter, we used a subset of the *20* News Groups dataset with *5000* documents from only *10* groups. The graph for the scalability test with these *5000* documents is drawn in Figure 12. The Master Document-Graph of the generated *5000* document-graphs contained *1304* unique edges.

Performance of GDClust

This section provides experimental results of GDClust using the FSG approach. All of the *1304* edges of the MDG were *1*-edge candidates before calling the `find_frequent_1-edge_subgraphs` procedure of Table 3. This procedure

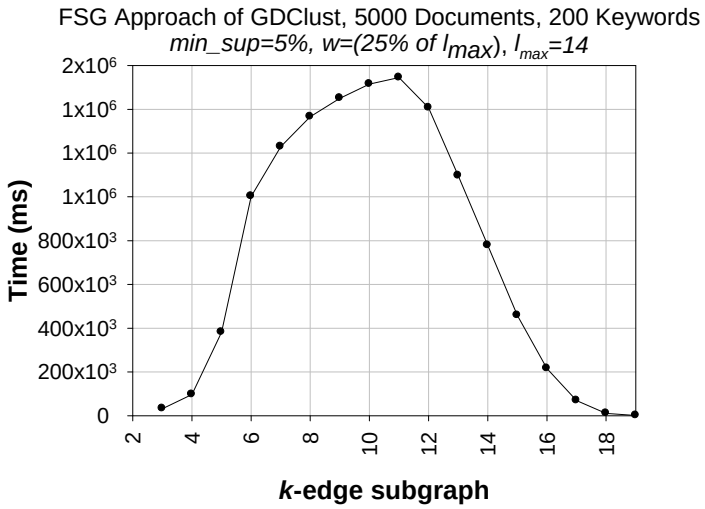


Figure 13: *k*-edge subgraph discovery time using FSG Approach of GDClust.

utilizes the Gaussian minimum support strategy to prune *1*-edge subgraphs from L_1. After this mechanism is applied, a total of *123* edges were left in L_1. In this experiment, the Apriori paradigm discovered the largest frequent subgraph with *19* edges. Figure 13 shows the Apriori algorithm's execution time to discover *k*-edge subgraphs using the FSG approach.

Table 6 shows the number of detected *k*-edge subgraphs and the number of attempts to combine *(k-1)*-edge subgraphs at each iteration of the Apriori paradigm. It shows that *10*-edge subgraphs are most frequent in our document-graph archive. A total of *713* different *10*-edge subgraphs passed the minimum support threshold *min_sup*. Since *10*-edge subgraphs are the most frequent ones, obviously the number of attempts to construct *11*-edge subgraphs from *10*-edge subgraphs reaches the maximum (marked in bold in Table 6). The execution time for generating *k*-edge subgraphs depends on the number of *(k-1)*-edge subgraphs generated in the previous iteration of the Apriori

Table 6: Number of *k*-edge subgraphs and attempts to construct *k*-edge subgraphs. Information about the experiment: *5000* documents, *200* Keywords, *min_sup=5%, w=(25%of l_{max}), 123 1*-edge subgraphs.

k	***Number of k-edge subgraphs***	**Number of attempts to construct *k*-edge subgraph**
2	107	123X123
3	177	11342
4	309	31152
5	439	95172
6	537	192282
7	614	287832
8	677	376382
9	708	457652
10	***713***	500556
11	694	***507656***
12	656	480942
13	596	429680
14	520	354620
15	416	269880
16	263	172640
17	98	68906
18	15	9506
19	1	210

paradigm. Since the number of *10*-edge subgraphs is the maximum, the peak of the line in Figure 13 is at *11*-edges.

For this simulation, *min_sup* is set to *5%* (allowing the Gaussian minimum supports to be in the range *[5, 100]*, resulting the amplitude of the Gaussian function, *A=95*), and the *c* value of equation (3.1) is derived with *w=(25% of l_{max})* in equation (3.2). We found $l_{max}=14$ from the Master Document-Graph of these *5000*-document graphs. The motivation of setting *w* to *(25% of l_{max})* appeared from the experimental result shown in Figure 3(d). It shows that the keyword distribution fits the Gaussian shape with *w=(25% of 16)* where l_{max} was *16* for that specific dataset.

Table 7: Impact of Gaussian minimum support on number of *1*-edge subgraphs.

Information about the experiment: varying *min_sup, w=(25% of* l_{max}*), 1304* edges in MDG.

min_sup (%)	**Number of *1*-edge subgraphs after Gaussian minimum support strategy**
1	290
2	245
3	190
4	149
5	123
6	107
7	100
8	80
9	67
10	61

To show the impact of Gaussian minimum support, we collected the number of selected *1*-edge subgraphs from the candidate list of *1304* edges with different *min_sup* and placed them in Table 7. It shows that the lower the *min_sup* (small *min_sup* indicates high amplitude *A* of equation 3.1), the higher the number of *1*-edge subgraphs after Gaussian filtration. Although Table 7 does not show it, sometimes, if *min_sup* is very small, the reduction of the *min_sup* value may not result in further inclusion of *1*-edge subgraphs if all edges in the mid-levels are already included by a higher *min_sup*. In that case, *w* can be increased to include additional *1*-edges subgraphs, if necessary.

Edges can be pruned even with fixed *min_sup,* but varying width denoted by *w*, of the Gaussian minimum support curve. A narrower Gaussian curve (smaller *w*) would result in fewer subgraphs, whereas a broader Gaussian curve (larger *w*) will generate more *1*-edge subgraphs. This behavior is reflected in Table 8. It shows that with a fixed *min_sup,* the number of selected *1*-edge subgraphs increases with increasing values of *w*

Table 8: Impact of *w* on number of *1*-edge subgraphs.
Information about the experiment: varying *w*, *min_sup=5%*.

x ***(w =x % of l_{max})***	**Number of *1*-edge subgraphs after Gaussian minimum support strategy**
1	98
12.5	98
25	123
37.5	155
50	201
67.5	259
75	295
87.5	327
100	341

(i.e., widening the Gaussian curve). Therefore, one can fine tune the parameters of the Gaussian function for expected accuracy in a specific domain.

One can also move the center of the Gaussian curve's peak by controlling the *b* value of the equation (3.1), skewing the curve in any direction (i.e., toward more or less abstract levels of MDG). In all of our experiments, we kept the curve symmetric because most of the important senses of the document-graphs are represented by the midlevel of the ontology (i.e., MDG) and similar document-graphs have the tendency to start overlapping at midlevel (discussed in Chapter 3).

Performance of GDClust-Plus: Subgraph-Extension mining

In this section, we use the same dataset with *5000* documents and *200* keywords that we used for the previous section. *min_sup* has been set to *5%* and *w=(25% of l_{max})*. Figure 14 shows the runtime required to detect certain *k*-edge subgraphs. Table 9 shows information about the quantity of *k*-edge subgraphs generated and the number of attempts made to construct these subgraphs from *(k-1)*-edge ones. The table shows that Subgraph-

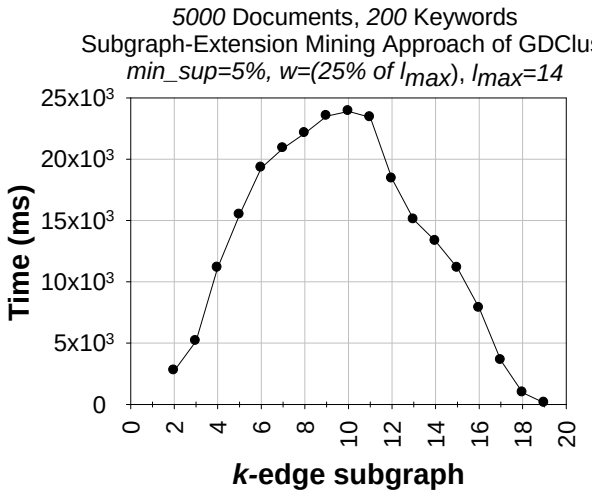


Figure 14: *k*-edge subgraph discovery time using Subgraph Extension Mining.

Table 9: Number of *k*-edge subgraphs and corresponding number of attempts.

Information about the experiment: *5000* documents, *200* Keywords, *min_sup=5%, w=(25%of l_{max})*) using Subgraph Extension Mining.

k	***Number of k-edge subgraphs***	**Number of attempts to construct *k*-edge subgraph**
2	107	224
3	177	655
4	309	1666
5	439	3639
6	537	5814
7	614	7519
8	677	9083
9	708	10450
10	713	11259
11	694	11681
12	656	11691
13	596	11293
14	520	10465
15	416	9248
16	263	7454
17	98	4726
18	15	1763
19	1	98

Extension mining found the largest subgraph with *19* edges during this subgraph discovery process.

The numbers of subgraphs of Table 9 perfectly matches the numbers of subgraphs in Table 6, confirming that our Subgraph-Extension mining approach performs the discovery of subgraphs accurately. Once again, among all *k*-edge subgraphs, *10*-edge subgraphs are the most frequent. In our Subgraph-Extension mining process, the number of attempts to generate *k*-edge subgraphs from *(k-1)*-edge subgraphs depends on the MDG-driven neighborhood lists of those *(k-1)*-edge subgraphs. As the result, the number of attempts is far lower than with the FSG approach of GDClust (because we avoid unsuccessful attempts to generate candidate subgraphs).

The Appendix of this thesis contains an experiment on Subgraph-Extension mining with a large dataset of *15000* documents. We compare the performance of Subgraph-Extension mining with the FSG approach of GDClust in the following section.

GDClust vs. GDClust-Plus

GDClust-Plus outperforms GDClust by a high magnitude due to our novel Subgraph-Extension mining technique. Figure 15 shows the difference between the runtime of these two approaches by combining Figure 13 and Figure 14 from the previous two sections. The gray line indicates the time necessary to discover *k*-edge subgraphs using GDClust's FSG approach. The black line indicates the performance of the Subgraph-Extension mining approach of GDClust-Plus. Due to the significant speed of the Subgraph-Extension mining, the black line looks linear and almost flat when it is

compared with the gray line of the FSG approach, although the actual behavior of Subgraph-Extension mining is not really linear. We made the boundary of the scale of the *Y*-axis of Figure 15 smaller and redraw it in Figure 16 just for an illustration. Both the curves have their peaks near to the maximum number of *k*-edge subgraphs (in this case,

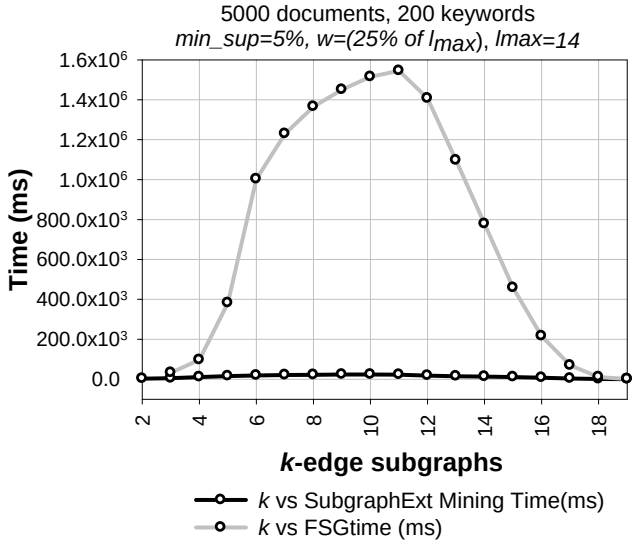


Figure 15: Comparison between FSG and Subgraph-Extension mining approach.

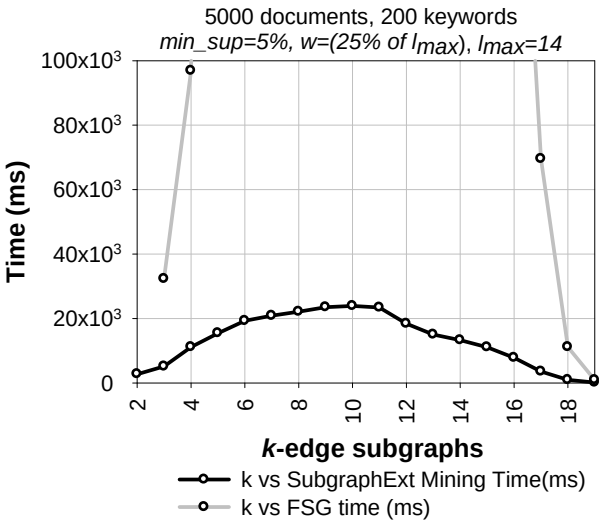


Figure 16: Representation of Figure 15 with shorter boundary of the scale.

k=10). Although, they have the similar tendency, GDClust-Plus's Subgraph-Extension mining approach performs around *60* times better than GDClust's FSG approach when detecting *10*-edge subgraphs. This number is not static and can vary depending on the document-archive size, the number of *1*-edge candidates generated using Gaussian dynamic minimum support strategy and the number of subgraphs generated at lower values of *k* (which is also dependent on the character of the document corpora).

The difference is the most visible at *k=11* where the blind generation of FSG reached its peak caused by the maximum number of *10*-edge subgraphs. For generating

Table 10: Comparison between GDClust and GDClust-Plus.
Information about the experiment: *5000* documents, *200* Keywords, *min_sup=5%, w=(25%of* l_{max}*)*) using Subgraph Extension Mining.

k	**Number of *k*-edge subgraphs**	**Number of attempts to construct *k*-edge subgraph**		**Saved attempts (in %)**
		FSG strategy of GDClust	**Subgraph Extension Mining of GDClust-Plus**	
2	107	123X123	224	98.5
3	177	11342	655	94.2
4	309	31152	1666	94.7
5	439	95172	3639	96.2
6	537	192282	5814	97.0
7	614	287832	7519	97.4
8	677	376382	9083	97.6
9	708	457652	10450	97.7
10	***713***	500556	11259	97.8
11	694	***507656***	11681	97.7
12	656	480942	***11691***	97.6
13	596	429680	11293	97.4
14	520	354620	10465	97.0
15	416	269880	9248	96.6
16	263	172640	7454	95.7
17	98	68906	4726	93.1
18	15	9506	1763	81.5
19	1	210	98	53.3

11-edge candidate subgraphs, the overhead was effectively reduced by our Subgraph-Extension mining approach from *507656* to *11681* attempts (saving *97.7%* attempts). GDClust-Plus does not try to combine every *(k-1)*-edge subgraph to generate *k*-edge subgraphs. Rather, it makes this attempts only when provided with the evidence of neighborhood from the MDG. As a result, it will perform same or better than FSG approach. If the MDG composes to a star then the Subgraph-Extension mining approach would perform the same as FSG approach. In practice, it is very unlikely that the MDG would form a star. So, the chance that the Subgraph-Extension mining approach would perform better than FSG approach is very high. Table 10 combines the number of attempts to construct *k*-edge subgraphs from *(k-1)*-edge subgraphs using both the approaches. It shows a significant differences in the numbers of attempts between these two algorithms and illustrates why GDClust-Plus dominates over the original GDClust.

Clustering Accuracy Measurement: Silhouette Coefficient

The frequent subgraphs discovered using either the FSG approach of GDClust or Subgraph-Extension mining approach of GDClust-Plus are used to cluster the *20* News Groups corpora. For our experiments, *5000* documents were chosen from *10* different news groups. Figure 17 shows the average silhouette coefficients for different numbers of clusters generated by our hierarchical agglomerative clustering (HAC). As the tendency of the curve is downward after certain number of clusters, we displayed silhouettes only up to *50* clusters in Figure 17. The graph shows that the maximum average silhouette coefficient (i.e., the best clustering) is found when the number of clusters is *8*. The result

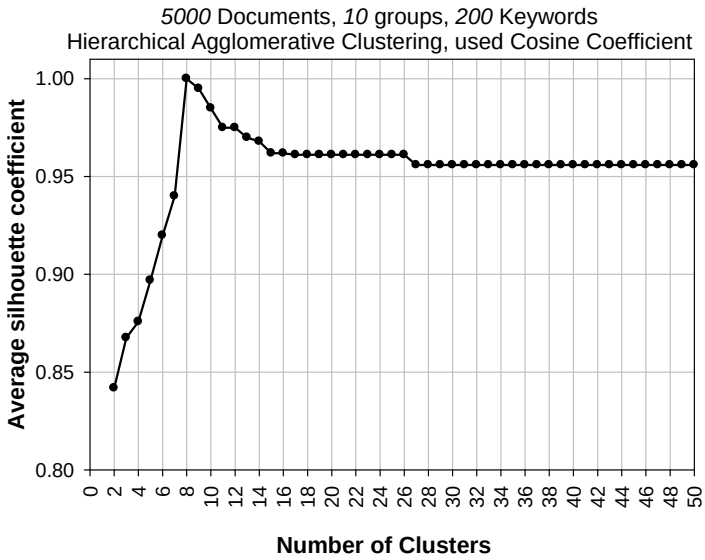


Figure 17: Average silhouette coefficient calculated for different numbers of clusters.

of 0.99 is very good, as average silhouette coefficient falls into *[-1, 1]*. So, our best result is really close to the number of groups in the input documents (our *5000* documents had *10* predefined groups). It needs to be noted that all the average silhouettes displayed in Figure 17 are greater than *0.8* which is particularly good. This means that average silhouette coefficient remains high in the neighborhood of pre-labeled number of clusters (i.e. *10*) and gradually falls downward in our a plot. This demonstrates a close match of cluster numbers with the number of predefined groups of the dataset.

Clustering Accuracy with Different Similarity Measures

We discussed different graph similarity measures in Chapter 4. In this section we show results of our analysis using those similarity measures. We used the same subset of the *20* News Groups dataset (*5000* documents from *10* groups and *200* keywords) for clustering.

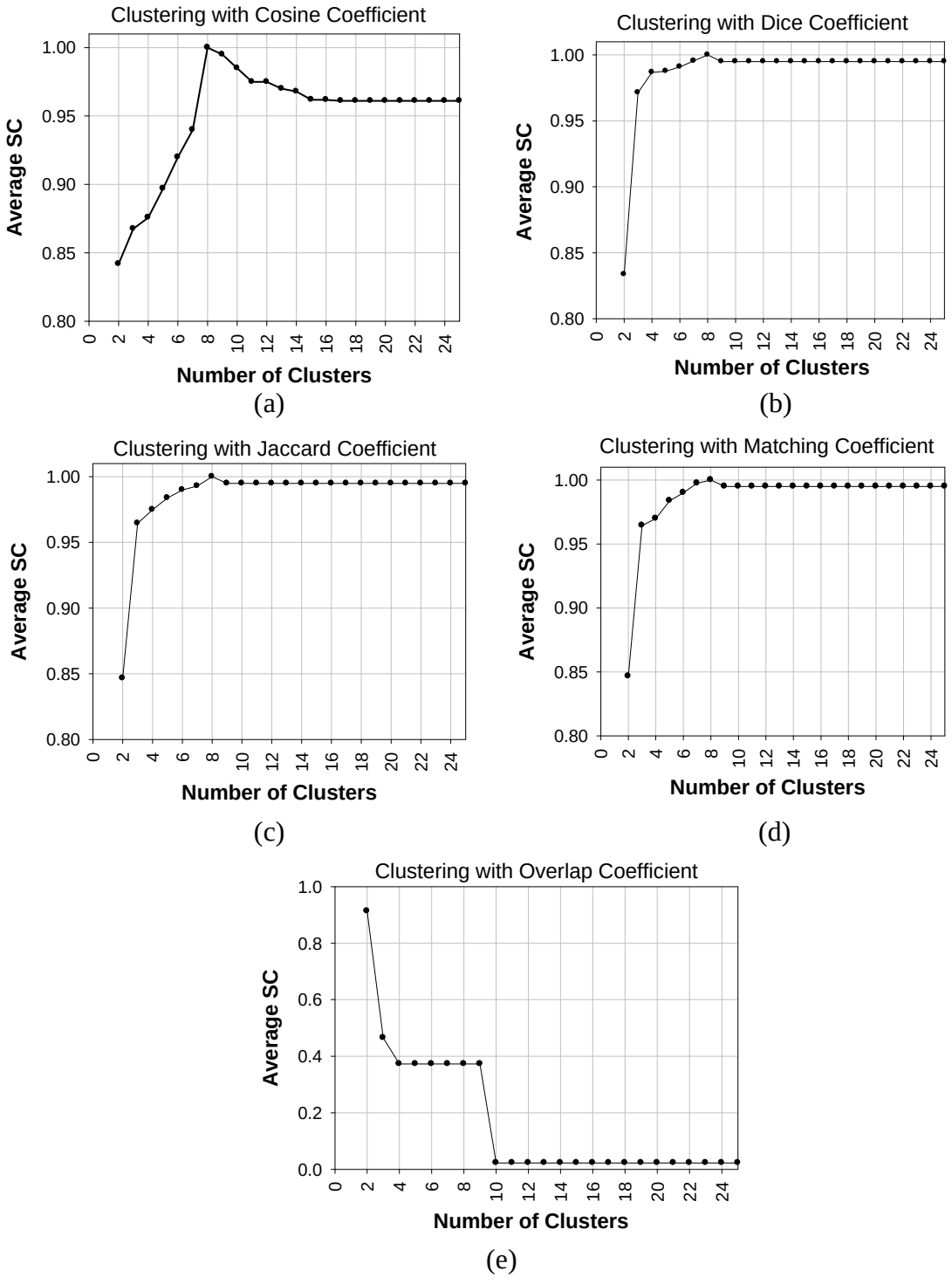


Figure 18: Clustering accuracy with different kinds of similarity measures.

Figure 18 shows our analysis with different similarity measures. It shows that we always get the highest average silhouette coefficient at *8* clusters using any similarity measure, except for the Overlap coefficient of Figure 18(e). The value of the Overlap coefficient is the maximum when every frequent subgraph of one document-graph appears in the set of frequent subgraphs of the other document-graph. This rarely happens in our document-graphs. As we remove very common *1*-edge subgraphs near the top levels of the hierarchy, it is unlikely that a lot of subgraphs will match between two document-graphs. Figure 18(e) shows that average silhouette is less than *0.8* for every number of clusters except for *2*. This illustrates that the Overlap coefficient does not meet our purpose.

Although the Dice, Jaccard and Matching coefficients show that the best clustering is found when there are *8* groups in the dataset, they all keep showing high accuracy when the number of clusters increases. From this perspective, Figure 18(a) shows better variation on the average silhouette coefficient for a different number of clusters confirming well-known opinion that Cosine coefficient is more appropriate in our case of documents' clustering. This is especially interesting, in the context of the fact that Cosine coefficient became popular to balance similarities between documents of significantly different size.

Comparison of GDClust-Plus with Traditional System

We discussed the traditional bag-of-tokens strategy for document clustering in Chapter 4. In this section, we compare the accuracies of the traditional document

clustering and the sense-based clustering mechanism of GDClust. To make sure our results can be easily compared, we used the same archive of *5000* documents from *10* groups with *200* keywords that we discussed before. As for traditional approaches, we directly used equation (4.7) and (4.11) to construct the distance matrix for Hierarchical Agglomerative Clustering. We plot average silhouette coefficients for different numbers of clusters and compare them with the results achieved for GDClust (Figure 17). Results of the clustering with three mechanisms: (1) GDClust (or GDClust-Plus), (2) traditional frequency-based document clustering and (3) traditional tf-idf based document clustering are depicted in Figure 19. The dashed line of Figure 19 is a copy of results reported in Figure 17. The gray line indicates results from the traditional (i.e., un-normalized) vector representation of documents with cosine distance measurement using frequency of terms only, where the solid black line is an outcome of similar vector representation of documents with utilized tf-idf (i.e., counts of terms' frequencies are properly normalized, reducing differences between long and short documents).

Figure 19 shows that GDClust detects best clustering with *8* clusters. It also shows a very high average silhouette with *10* clusters indicating highly accurate clustering. In contrary, the vector representation of documents cannot show satisfactory accuracy with any number of clusters, except for *2*, which is not desired for *5000* documents of *10* groups. Also with large numbers of clusters, both frequency-based and tf-idf based clustering mechanisms quickly reaches negative average silhouettes. The frequency-based traditional approach results in the negative average silhouette coefficient after *17* clusters, and the tf-idf based approach generates negative values after *11* clusters.

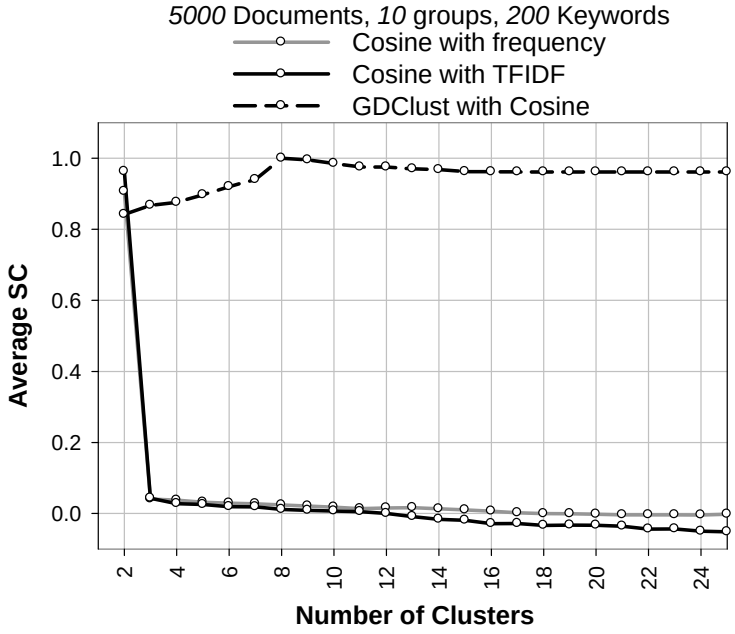


Figure 19: Comparison of Subgraph Extension Mining with traditional systems.

Although average silhouettes can be in the range *[-1, 1]*, a negative value is definitely undesirable because this corresponds to a case in which the average dissimilarity of documents in the cluster is greater than the minimum average dissimilarity of documents in other clusters [49]. Therefore, a negative average silhouette coefficient at any number of clusters indicates strongly inaccurate clustering at that number.

Because GDClust is a subgraph-based clustering mechanism, although there are only *200* keywords, it discovers enough frequent subgraphs for each of the documents to cluster properly. It results in positive average silhouettes at every number of clusters. Therefore, we can conclude that our GDClust system is capable of providing proper sense-based clusters, even with a small number of keywords.

CHAPTER 7

CONCLUSION

GDClust presents a new technique for clustering text documents based on the co-occurrence of frequent senses in the documents. The developed, novel approach offers an interesting, sense-based alternative to the commonly used bag-of-tokens technique for clustering text documents. Unlike traditional systems, GDClust harnesses its clustering capability from the frequent senses discovered in the documents. It uses graph-based mining technology to discover frequent senses. The novelty of our work lies beneath two new approaches introduced in this report: Dynamic minimum support strategy and Subgraph-Extension mining technique. Subgraph-Extension mining technique outperforms FSG strategy by high magnitudes. Besides, we have shown that GDClust performs more accurately than traditional systems. GDClust is an automated system and requires minimal user interaction for its operations.

Limitations

To keep things simple, we used only noun keywords in all our experiments. We believe however that our techniques will work well with all parts of speech that are provided with a hierarchical ontology. Our claim is based on the observation that all our approaches are graph-based and do not focus on the comparison of the actual keywords. The document-graph construction algorithm (Table 2) could be modified to incorporate all the parts of speech if necessary. Indeed, based on our experiments, we started to

believe that the noun keywords from the text documents are enough to cluster documents accurately.

Future Works

In the future, we want to develop an intelligent system for the Dynamic minimum support strategy. In our system, since the *20* News Groups dataset follows Gaussian trend, we have utilized Gaussian minimum support strategy for generating Dynamic minimum support thresholds. In Chapter 3, we describe that the domain behavior for the keywords can be different than the Gaussian trend, depending on the document archive. This requires an intelligent system to determine the shape of the Dynamic minimum support curve if the domain is unknown.

REFERENCES

1. M. S. Hossain and R. Angryk, "GDClust: A Graph-Based Document Clustering Technique", *2007 IEEE International Conference on Data Mining (ICDM'07), IEEE ICDM Workshop on Mining Graphs and Complex Structures*, IEEE Press, Omaha, NE, USA, October 28-31, 2007, pp. 417-422.

2. "WordNet: A Lexical Database for the English Language", *Cognitive Science Laboratory Princeton University*, http://wordnet.princeton.edu/

3. F. Sebastiani, "Machine learning in automated text categorization", *ACM Computing Surveys*, v. 34 (1), 2002, pp. 1-47.

4. C. D. Manning and H. Schutze, "Foundations of Natural Language Processing", *MIT Press*, 1999.

5. C. Cleverdon, "Optimizing convenient online access to bibliographic databases", *Information Survey and Use*, v. 4 (1), 1984, pp. 37-47.

6. G. Miller, R. Beckwith, C. FellBaum, D. Gross, K. Miller, and R. Tengi, "Five papers on WordNet", *Princeton University*, August 1993.

7. R. Agrawal, and R. Srikant, "Fast Algorithms for Mining Association Rules", *Proceedings of International Conference on Very Large Data Bases (VLDB'94)*, Santiago, Chile, September 1994, pp. 487–499.

8. R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases", *Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, Washington, D.C., USA, 1993, pp. 207–216.

9. T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An Efficient Data Clustering Method for Very Large Databases", *Proceedings of ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, Montreal, Quebec, Canada, June 4-6, 1996, pp. 103–114.

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

11. S. T. Dumais, G. W. Furnas, T. K. Landauer, and S. Deerwester, “Using latent semantic analysis to improve information retrieval”, *Proceedings of Conference on Human Factors in Computing*, New York, 1998, pp. 281–285.

12. M. Kuramochi and G. Karypis, “An efficient algorithm for discovering frequent subgraphs”, *IEEE Transactions on Knowledge and Data Engineering*, v. 16(9), September 2004, pp.1038–1051.

13. R. N. Chittimoori, L. B. Holder, and D. J. Cook, “Applying the SUBDUE substructure discovery system to the chemical toxicity domain”, *Proceedings of the 12th International Florida AI Research Society Conference*, 1999, pp. 90–94.

14. L. Dehaspe, H. Toivonen, and R. D. King. “Finding frequent substructures in chemical compounds”, *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 30–36.

15. A. Srinivasan, R. D. King, S. Muggleton, and M. J. E. Sternberg, “Carcinogenesis predictions using ILP”, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, v. 1297, 1997, pp. 273–287.

16. A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg, “The predictive toxicology evaluation challenge”, *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI)*, 1997, pp. 1–6.

17. H. Kalviainen, and E. Oja. “Comparisons of attributed graph matching algorithms for computer vision”, *Proceedings of STEP-90, Finnish Artificial Intelligence Symposium*, Oulu, Finland, June 1990, pp. 354–368.

18. D. A. L. Piriyakumar, and P. Levi, “An efficient A* based algorithm for optimal graph matching applied to computer vision”, *GRWSIA-98*, Munich, 1998.

19. D. Dupplaw and P. H. Lewis, “Content-based image retrieval with scale-spaced object trees”, *Proceeding of SPIE: Storage and Retrieval for Media Databases*, v. 3972, 2000, pp. 253–261.

20. M. E. J. Newman, “The structure and function of complex networks”, *Society for Industrial and Applied Mathematics (SIAM) Review*, v. 45(2), 2003, pp. 167–256.

21. C. W. K. Chen and D. Y. Y. Yun, “Unifying graph-matching problem with a practical solution”, *Proceeding of International Conference on Systems, Signals, Control, Computers*, September 1998.

22. L. Holder, D. Cook, and S. Djoko, “Substructure discovery in the SUBDUE system”, *Proceedings of the Workshop on Knowledge Discovery in Databases*, 1994, pp. 169–180.

23. K. Yoshida and H. Motoda, “CLIP: Concept learning from inference patterns”, *Artificial Intelligence*, v. 75(1), 1995, pp. 63–92.

24. R. Agrawal, M. Mehta, J. Shafer, R. Srikant, A. Arning, and T. Bollinger, “The Quest Data Mining System”, *Proceeding on 2nd International Conference on Data Mining and Knowledge Discovery (KDD'96)*, Portland, OR, USA, August 1996, pp. 244–249.

25. H. Mannila, H. Toivonen, and I. Verkamo, “Efficient Algorithms for Discovering Association Rules”, *Proceedings of AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, Seattle, Washington, USA, July 1994, pp. 181–192.

26. J. S. Park, M. S. Chen, and P. S. Yu, “An effective hash-based algorithm for mining association rules”, *Proceedings of the ACM SIGMOD international conference on Management of data (SIGMOD'95)*, San Jose, CA, USA, May 1995, pp. 175–186.

27. J. Han, and Y. Fu, “Discovery of Multiple-Level Association Rules from Large Databases”, *Proceedings of the 21th International Conference on Very Large Data Bases (VLDB’95)*, Zurick, Switzerland, September 1995. pp. 420–431.

28. M. Kuramochi, and G. Karypis, “Frequent subgraph discovery”, *Proceedings of IEEE International Conference on Data Mining*, 29 November-2 December. 2001, pp. 313–320.

29. X. Yan, and J. Han, “gSpan: graph-based substructure pattern mining”, *Proceedings of IEEE International Conference on Data Mining*, December 2002, pp. 721–724.

30. C. Moti, and G. Ehud, “Diagonally Subgraphs Pattern Mining”, *Proceedings of the 9th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, June 2004, pp. 51–58.

31. M. J. Zaki, “Efficiently mining frequent trees in a forest”, *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'02*, 2002, pp. 71–80.

32. X. Yan, P. S. Yu, and J. Han, “Substructure Similarity Search in Graph Databases”, *Proceedings of the 2005 ACM SIGMOD international conference on Management of data, SIGMOD'05*, 2005, pp. 766–777.

33. X. Yan, F. Zhu, J. Han, and P. S. Yu, “Searching Substructures with Superimposed Distance”, *Proceedings of the 22nd International Conference on Data Engineering , ICDE'06,* 2006, pp. 88–97.

34. N. Ketkar, L. Holder, D. Cook, R. Shah, and J. Coble, “Subdue: Compression-based Frequent Pattern Discovery in Graph Data”, *Proceedings of the ACM KDD Workshop on Open-Source Data Mining,* August 2005, pp. 71–76.

35. J. Tomita, and G. Kikui, “Interactive Web search by graphical query refinement”, *Proceedings of the 10th international World Wide Web conference (WWW01)*, 2001, pp. 190–191.

36. A. McCallum, “Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering”, *http://www.cs.cmu.edu/~mccallum/bow/*

37. “*20* News Groups dataset”, *http://people.csail.mit.edu/jrennie/20Newsgroups/*

38. T. R. Gruber, “A translation approach to portable ontology specifications”, *Knowledge Acquisition*, v. 5(2), pp. 199–220, 1993.

39. M. S. Hossain, M. Akbar, and R. Angryk, “Sense Based Organization of Descriptive Data”, *2007 IEEE International Conference on Systems, Man and Cybernetics*, IEEE Press, Montreal, Quebec, Canada, October 7-10, 2007. pp. 468-473.

40. T. Cormen, C. Leiserson, R. Rivest and C. Stein, “Introduction to Algorithms”, Second Edition, MIT Press, McGraw-Hill Book Company, Section 22.3, pp. 540–549.

41. E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. D. Ullman, and C. Yang, “Finding interesting associations without support pruning”, *IEEE Transactions on Knowledge and Data Engineering*, v. 13(1), January/February 2001, pp. 64–78.

42. D. Gibson, R. Kumar, and A. Tomkins, “Discovering large dense subgraphs in massive graphs”, *Proceedings of the 31st international conference on Very large data bases (VLDB'05)*, Trondheim, Norway, 2005, pp. 721 - 732.

43. P. Ganesan, H. Garcia-Molina, and J. Widom, “Exploiting hierarchical domain structure to compute similarity”, *ACM Transactions on Information Systems (TOIS)*, v. 21(1), January 2003, pp. 64 - 93.

44. D. Lin, “An information-theoretic definition of similarity”, *Proceedings of the 15th International Conf. on Machine Learning*, Morgan Kaufmann, San Francisco, CA, (1998), pp. 296–304.

45. H. Chen, M. Lin and Y. Wei, "Novel association measures using web search with double checking", *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, Sydney, Australia, 2006, pp. 1009 - 1016.

46. R. White, J. Jose, "A study of topic similarity measures", *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, Sheffield, United Kingdom, July 2004, pp. 520 - 521.

47. F. Lin, C. M. Hsueh, "Knowledge map creation and maintenance for virtual communities of practice", *International Journal of Information Processing and Management, ACM*, v. 42(2), 2006, pp. 551–568.

48. N. R. Adam, V. P. Janeja, and V. Atluri, "Neighborhood Based Detection of Anomalies in High Dimensional Spatio-temporal Sensor Datasets", *Proceedings of ACM Symposium on Applied Computing*, March 2004, pp. 576–583.

49. Tan P. N., Steinbachm M., Kumar V., "Introduction to data mining", *Addison-Wesley*, ISBN: 0321321367, April 2005, pp. 539–547.

50. "Library Functions for WordNet", http://wordnet.princeton.edu/doc

51. "JWNL (Java WordNet Library)", http://sourceforge.net/projects/jwordnet

52. "JWordNet", http://sourceforge.net/projects/jwn/

53. "WordNet 2.0 data files in MySQL format", Android Technologies Inc,. http://www.androidtech.com/html/wordnet-mysql-20.php

54. "WordNet SQL Builder", http://wnsqlbuilder.sourceforge.net/

55. "WordNet JNI Java Native Support", http://sourceforge.net/projects/wnjn/

56. “JGraphT”, http://jgrapht.sourceforge.net/

57. J. Han and M. Kamber, “Data Mining: Concepts and Techniques”, 2nd Edition, *Morgan Kaufmann Pub.*, ISBN: 1558609016, March 2006, pp. 234–242.

APPENDIX A

SUBGRAPH-EXTENSION MINING FOR LARGE DATASET

From the experimental results of Chapter 6, we know that the GDClust's FSG approach is less efficient than the Subgraph-Extension mining approach. This is why it is very time inefficient to conduct subgraph discovery on large datasets using FSG approach of GDClust. GDClust-Plus's Subgraph-Extension mining approach provides faster execution even with large datasets. Figure 20 shows an example with *15000* documents and *200* keywords. Table 11 contains the corresponding information.

Table 11: Information about Subgraph Extension Mining of Figure 20.

k	***N(k)***	**Attempts (*k*)**
2	65	132
3	101	317
4	181	726
5	306	1610
6	464	3144
7	621	5302
8	758	7757
9	830	10163
10	832	11748
11	769	12305
12	650	11797
13	479	10272
14	293	7755
15	140	4835
16	47	2343
17	10	795
18	1	170

N(k)=Number of *k*-edge subgraphs
Attempts(k) = Number of attempts to construct *k*-edge subgraphs

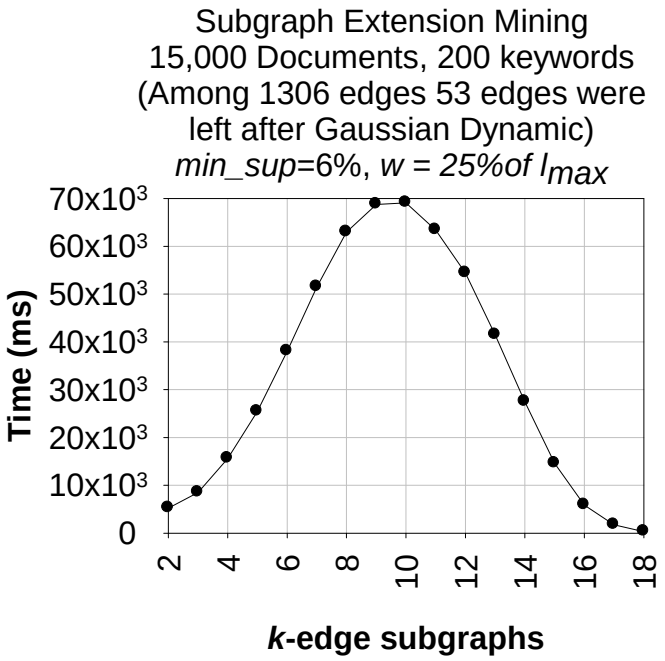


Figure 20: Time elapsed to detect different *k*-edge subgraphs.