FP-TREE MOTIVATED SYSTEM FOR INFORMATION RETRIEVAL USING AN ABSTRACTION PATH-BASED INVERTED INDEX

by

Richard Arthur McAllister

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

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

November, 2009

© Copyright

by

Richard Arthur McAllister

2009

All Rights Reserved

APPROVAL

of a thesis submitted by

Richard Arthur McAllister

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, 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 Fox

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfullment 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.

Richard Arthur McAllister

November, 2009

DEDICATION

To me - I rock!

ACKNOWLEDGEMENTS

I wish to thank my advisor, Dr. Rafal Angryk, for his outstanding leadership, quick wit, and persistence. As well, I wish to thank my committee for their advice, input, and mentoring. I would also like to thank Robbie Lamb, Monika Akbar, and Mahmoud Shahriar Hossain for their support, both morally and scientifically, and for their friendship. Steven Gertiser provided excellent advice, support and encouragement which was valuable in seeing this project through to completion. The group of former Montana State University computer science graduate students collectively known as the "Pigeons" provided me with an outlet full of friends all of whom understand computation and will go on to do great things in the field. Most of all, I would like to thank my family and especially my parents, Frank and Marcia, for their patience, support and encouragement. My brother Francis must not escape mention here because it was he who suggested that I get into this line of work in the first place.

Funding Acknowledgment

This work was kindly supported by RightNow Technologies and the National Aeronautics and Space Administration.

TABLE OF CONTENTS

1. INTRODUCTION 1

Motivation 1
Scope 2

2. BACKGROUND 3

Information Retrieval Background 3
Information Retrieval System Architecture 5
Document Corpus 5
Parsing Linguistics 6
Word Vectors 7
Indexers and the Inverted Index 8
User Query and the Free Text Query Parser 8
Scoring and Ranking and the Ranked Results 10
Construction of Inverted Indices 10
Information Retrieval System Evaluation 11
Frequent Pattern Mining 14
Support 15
FP-Growth Frequent Itemset Mining 16
Taxonomies and Frequent Patterns in Text Processing 21

3. APPROACH 25

Overview 25
WordNet 27
Synonomy 28
Hypernymy 28
Hyponymy 28
Polysemy 29
Abstraction Paths 29
Preprocessing of Documents 31
Word Vector Creation 32
Document Graphs 33
Discovery of Document Abstraction Paths 34
Construction of Document Graphs 35
Document Graph Construction for Document 1 35
Document Graph Construction for Document 2 41
The Master Document Graph 43
Document Abstraction Paths 44
Document Inverted Index 53

TABLE OF CONTENTS – CONTINUED

Mining the Master Document Tree 56
Minimum Support 56
Building Abstraction Path-Based Inverted Indices 65
Indexing System Architecture 66
Processing of Queries 66
The Query Tree and Query Abstraction Paths 68
Selection of Related Documents from the Inverted Index 68
Ranking 68

4. EXPERIMENTAL EVALUATION 70

Investigations 74

5. CONCLUSIONS AND FUTURE WORK 83

Future Work 83

REFERENCES CITED 86

APPENDIX A: Symbols and Abbreviations 91

LIST OF TABLES

Table Page

2.1 An Example of a Small Document Corpus 6

2.2 A Word vector vable, denoted as T, with examples of *tfidf* values constructed for the document corpus in table 2.1 (stopwords included to preserve simplicity) 9

2.3 A Database of Transactions [1] 18

3.1 Random Data. Each line represents a "document" with the "words" represented as letters from a through g. 34

3.2 Example *tfidf* table based on the random data 35

4.1 All Newsgroups Included in the 20 Newsgroups Dataset 70

4.2 10 Newsgroups Used for the Investigations 70

4.3 Queries Used for the Investigations 71

LIST OF FIGURES

Figure Page

2.1 Overview of an Information Retrieval System 6
2.2 Inverted Index Creation Process (stopwords included for simplicity) 12
2.3 Precision vs. Recall: hypothetical 14
2.4 FP-Tree After the Insertion of Transaction 1 19
2.5 FP-Tree After the Insertion of Transaction 2 20
2.6 The FP-Tree after all transactions have been inserted 20
2.7 A Prefix Paths for Item p 22
2.8 A Conditional FP-Tree for 1-itemset p 22
3.1 The Ontology Abstraction-Based Information Retrieval System 26
3.2 The types of relationships that compose WordNet 30
3.3 Time to create document graphs 44
3.4 Memory usage for creating document graphs 45
3.5 The Master Document Graph 45
3.6 Numbers of Paths and Itemsets 67
3.7 Changes to the Elementary Information Retrieval System Architecture 67
3.8 Word Vector Graphical Representation 69
4.1 Path Length Investigation Results: Precision at k 76
4.2 Path Length Investigation Results: Recall at k 77
4.3 Number of Abstraction Paths in the Inverted Index 78
4.4 Path Popularity Investigation Results: Precision at k 79
4.5 Path Popularity Investigation Results: Recall at k 80
4.6 Abs. Paths in the Inverted Index and Avg. Index Row Length 81
4.7 Precision vs. Recall: Best and Worst Results 81
4.8 Precision vs. Recall: Path Length Investigations 82
4.9 Precision vs. Recall: Path Popularity Investigations 82

ABSTRACT

Language ontologies provide an avenue for automated lexical analysis that may be used in concert with existing information retrieval methods to allow lexicographic relationships to be considered in searches. This paper presents a method of information retrieval that uses WordNet, a database of lexical ontologies, to generate paths of abstraction via lexicographic relationships, and uses these relationships as the basis for an inverted index to be used in the retrieval of documents from an indexed corpus. We present this method as a entree to a line of research in using lexical ontologies to perform graph analysis of documents, and through this process improve the precision of existing information retrieval techniques.

1

INTRODUCTION

Information retrieval techniques relying on "bag of tokens" approaches, which reduce both query and document representations to lists of words without regard to position or cardinality, have done well with regard to generalized information retrieval, and such success would indeed lead many laypersons to believe that information retrieval no longer requires consideration. Practical examples of the success of the "bag of tokens" approach are in the popular domain and, with all of their frustrating exceptions considered, are very useful. This usefulness may be a function of ease of use coupled with the flexibility of language and the multitude of ways to express a single thought. In a search, one may attempt many combinations of words expressing the same thought when trying to satisfy an information need. But in the end, domains exist in which greater specificity is demanded in the result set.

Motivation

The situations that demand greater precision may be in engineering or the sciences, where terminology often exhibits dynamic characteristics. With these considerations, a probabilistic information retrieval approach taking into account words' relationships among each other with respect to common lexical ontologies is perhaps a prudent method.

The use of lexical ontologies from computational lexicography and association rule techniques from data mining provide the building blocks to the method described in this investigation. Lexical ontologies provide a comprehensive framework for the language used in the data under consideration in the vernacular of whoever has the information need. For example, the lexical ontology *WordNet* provides a generalized

framework of the entire English language. This paper presents a method for using this ontology, along with frequent pattern mining to perform information retrieval via coupling abstract terms to terms that appear in the text.

Intersecting paths of term derivation may also be used to identify the proper meanings of terms from their contexts. Terms appearing together in context may lead to certain common abstractions having a greater degree of significance than other abstractions. This may lead to the proper definition of a term being identified.

Scope

The following presents a detailed description of this work. The next section presents a background on information retrieval that includes a general description of the process for constructing an inverted index and methods for evaluating the effectiveness of information retrieval systems. The next section describes frequent pattern mining without candidate generation. After information retrieval systems have been described generally, we then describe our system, its governing principles, and its architecture. Following the system description, a survey of our experimentation and results is given.

The main contribution of this work is a framework for an information retrieval system that is based on frequent abstraction paths as opposed to a keyword-based approach. To this end, we have created a frequent pattern-based mechanism for frequent abstraction path discovery that takes advantage of hierarchically-organized background knowledge. We present preliminary results as a "proof of concept."

3

BACKGROUND

Before detailing our approach to information retrieval indexing several important technologies should be discussed. Our approach was assembled using a combination of these technologies, including elementary information retrieval methods, frequent pattern analysis, and language ontologies.

Information Retrieval Background

Information retrieval is the process of locating information in a collection of documents (usually text) on a particular topic by using a system constructed to process queries for that corpus. Commonly, search engines such as Google and Yahoo feature familiar implementations of these systems and are used ubiquitously. Also, specialized information retrieval systems, such as the INQUERY [2] system at the University of Massachusetts are used in document control systems in corporate and research environments that organize information regarding scientific research, safety procedures, legal proceedings, and a variety of other topics.

To make a proper study of information retrieval we need to define exactly the nature of the problem on which we are working. Manning [3] defines information retrieval as:

> Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers). [3]

The term *material of an unstructured nature*, or more appropriately *unstructured data* refers to data that does not have a clear, semantically overt, easy-for-a-computer

structure [3]. Due to the free structure of the spoken and written word, text documents fall into this definition of unstructured data. Even though there are semantic and syntactic rules of grammar that govern the processes of speech and writing, these rules yield structures that are unreliable and unpredictable with respect to computation. The dynamic nature of language, the misuse of rules in speech and writing, and the sheer multiplicity of combinations that proper use of these rules may yield are just three causes of this uncertainty.

The popularity and apparent effectiveness of web search engines such as the aforementioned Google and Yahoo may lead laypersons to believe that IR is no longer a problem that requires significant effort in research and that whatever anyone needs can be found very quickly using a technology that has already been deployed in large measure by these entities. But as these entities have created products that address the general issue, namely that of web search (or information retrieval on the World Wide Web) this conclusion is flawed. A major difference between these systems and their domain-specific relatives can be explained in terms of the relative importance that web search engines put on two measures of information retrieval, namely *precision* and *recall*.

Given a query q performed on a collection of documents D, *precision* (p), the ratio of relevant documents to retrieved documents, is a measure of what fraction of the returned results D_q are relevant to the *information need* of the user [3]. An *information need* is the topic about which the user desires to know more and is codified, to the best of the user's ability, in the query [3]. *Recall* (r), the ratio of returned documents to all documents relevant in the collection, is a measure of what fraction of the documents in the corpus that are truly relevant to the user's information need were retrieved [3]. Considering the problem that web search engines have been engineered to solve, and considering the behavior of their clientele and an

operating environment consisting of a vast array of documents these systems have been engineered to produce results that concentrate on high recall rather than high precision. This is acceptable as the World Wide Web is characterized by a large number of documents and search engine users who usually issue short queries and only consider the first few pages of results [3]. Algorithms such as "Page Rank" [4] also play a role in characterizing the relevance of result sets. In other domains, such as in relatively mission-critical, domain-specific information retrieval systems this emphasis would yield a product that is not acceptable, as the returned results, being of low precision are likely to contain a lot of noise. It is the reduction of this noise, i.e. an enhancement of precision with which we are concerned.

Information Retrieval System Architecture

Figure 2.1 shows the architecture of a fully functional information retrieval system. The diagram is by no means exhaustive, since there may be some methods of machine learning included to address index formation and scoring parameters as well as mechanisms that enable the environment to check the spelling of the queries and handle other anomalies such as multiple-word phrases. The figure specifically depicts the fundamental elements with which we are concerned. The following is a description of these elements.

Document Corpus: This is the collection of documents, referred to henceforth as the corpus of documents and denoted as D, that are to be indexed for the IR system and for which queries will be processed. For our purposes this collection is finite, although some collections, such as the World Wide Web, expand without limit. An example of a tiny document corpus can be seen in table 2.1.

Figure 2.1: Overview of an Information Retrieval System

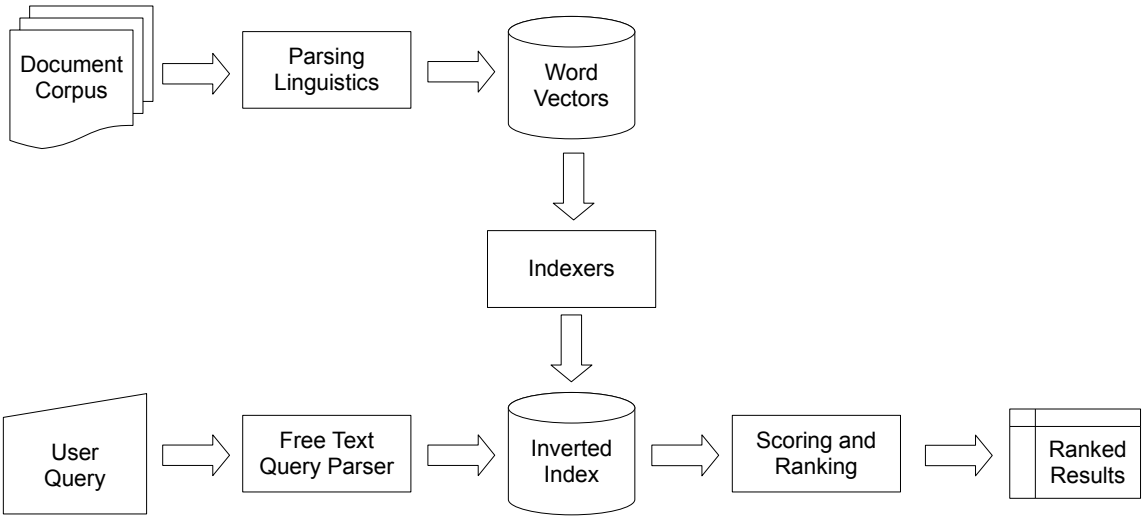


Table 2.1: An Example of a Small Document Corpus

doc	**text**
1	I did enact Julius Caesar: I was killed i the Capitol: Brutus killed me.
2	So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.
3	I told you Brutus killed the ambitious Julius Caesar.

Parsing Linguistics: This component includes the processes that parse words, word fragments, and phrases. This module may become rather complex, as it may be desired that some terms that include numbers or other non-letter characters be included in the indices. The parsing linguistics include mechanisms for discerning what types of tokens are included in these indices.

Included in these linguistics is the heuristic method for removing stopwords. During this process every document is compared with a list of known stopwords (e.g., a, an, the, his, her) that comes from an authoritative lexicographic body [5]. If a word is

found in this list then that word is discarded and is not considered in the calculation and construction of an index.

Another portion of the parsing linguistics module may include the procedure for word stemming [6]. It is common for an IR system to convert the words it indexes into as close a canonical representation of that word as is possible to determine using linguistic heuristics. For example, a properly stemmed version of the words "prevents", "prevented", "preventing", and "prevention" is the word "prevent." It is assumed that when a document uses one of the non-stemmed versions of the word "prevent" that the document is referring to the general concept embodied in the stemmed version of the word. Some stemming algorithms in popular usage are the Lovins, Porter, and Snowball stemmers [7].

Word Vectors: Word vectors, denoted as T, contain measures of importance of each word w in every document (d) in a corpus (D) to each individual document. The vectors may be a normalized measure of importance such as the Term Frequency Inverse Document Frequency (*tfidf*) measure [8]. Equations 2.1 through 2.3 depict the calculation of *tfidf* values. Term Frequency (tf) divides the number of times a word is found in a document, denoted as $a_{w,d}$, by the total number of words in that document (see equation 2.1). Inverse Document Frequency (idf) divides the number of documents in the corpus by the number of documents in the corpus that contain a particular word (see equation 2.2). idf uses the log of the aforementioned calculation for normalization, since the product of the number of documents and the number of documents containing a particular word may be greater than 1. It is important that a normalized measure be used because non-normalized term frequency measurements favor longer documents because they have a higher likelihood of having

a greater multiplicity of words. In equations 2.1 through 2.3 a_{w_d,d_j} equals the number of occurrences of term w_k in document d_j, a_{w_k,d_j} is the number of occurrences of term w_k in document d_j, $|D|$ is the number of documents in the entire corpus and $\mid \{d_j : w_k \in d_j\} \mid$ is the number of documents from that corpus in which term w_k occurs. Table 2.2 shows the word vector for the small document corpus in table 2.1.

$$\forall d_j \in D, \forall w_k \in d_j, tf_{w_k,d_j} = \frac{a_{w_k,d_j}}{\sum_{w_k \in d_j} a_{w_k,d_j}} \tag{2.1}$$

$$\forall w_k \in D, idf_{w_k} = log \frac{|D|}{\mid \{d_j \in D \wedge w_k \in d_j\} \mid} \tag{2.2}$$

$$tfidf_{w_k,d_j} = tf_{w_k,d_j} \cdot idf_{w_k} \tag{2.3}$$

Indexers and the Inverted Index: This module is charged with the construction of the inverted index, which is the mechanism that relates words to the documents in which they occur. The inverted index module contains the aforementioned inverted index against which queries will be processed.

User Query and the Free Text Query Parser: This is the point of entry for the user. It consists of an unformatted user query that is then submitted to the IR system through its user interface. The free text query is the common query processing method

Table 2.2: A Word vector vable, denoted as T, with examples of *tfidf* values constructed for the document corpus in table 2.1 (stopwords included to preserve simplicity)

term	Doc 1	Doc 2	Doc 3
ambitious	0	0.03	0.05
be	0	0.08	0
Brutus	0	0	0
capitol	0.08	0	0
Caesar	0	0	0
did	0.08	0	0
enact	0.08	0	0
hath	0.08	0	0
I	0.03	0	0.05
i'	0.08	0	0
it	0	0.08	0
Julius	0.03	0	0.05
killed	0.06	0	0.05
let	0	0.08	0
me	0.08	0	0
noble	0	0.08	0
so	0	0.08	0
the	0	0	0
told	0	0.03	0.05
you	0	0.03	0.05
was	0.03	0.03	0
with	0	0.08	0

that involves the user submitting an input that is an unstructured set of words that may include repetitions [3]. It is the query parsing method that is used in web search engines without the so-called "advanced search" option enabled. Other query engines offer the option of using regular expressions or markup characters that clarify what is being sought.

Normally a free text query is an ordered set of words that may contain stopwords. Since the IR system that is under consideration here treats the query as a *bag of words*,

or an unordered set of words without duplicates [3], the query must be handled in a way that facilitates this treatment. It is the responsibility of this module to perform this function.

Scoring and Ranking and the Ranked Results: This module contains the logic and attendant mechanisms for determining an order of precedence with regard to relevance for the documents that have been returned by the query. The ranked list of results and methods for accessing the documents in this list is returned to the user upon completion of the search process.

Construction of Inverted Indices

An effective, but horribly inefficient method of information retrieval would be to scan the entire dataset for the appearance of the words in each query every time a query is issued. Such a system would clearly be unreasonable as it would require reading each document in the entire corpus upon each query and comparing each term in each document to each term in the query. This operation would have a time complexity of $O(|D|\overline{|W|}|W_q|)$ where $|D|$ is the number of documents $\overline{|W|}$ is the average number of words in each document and $|W_q|$ is the number of words in a query. If an IR system is to be effective there must be an efficient mechanism that uses the words from the document corpus as an index to the documents in which they appear. If this mechanism is not used it will be necessary for the IR system to traverse each document upon each query, collect references as each desired word is encountered, and then to merge the results. To this end we create an *inverted index*.

An inverted index is a structure used in IR systems that maps words to the documents in which they occur. It receives the 'inverted' designation because usually, when considering a document, the document may be viewed as an index that maps

documents to words, not words to documents. An inverted index facilitates efficient keyword searches since it provides an efficient secondary access path to the documents via their constituent words. For a set of words, a multiplicity of documents in the corpus may contain the same set of words. Using this idea, commonalities among these word uses may be exploited to facilitate the retrieval of documents addressing common subject matter, or at least refer to their disparate subject matter in similar ways.

The creation of an inverted index proceeds as detailed in algorithm 1. The process consists of a collection phase, a sort phase and finally a collapse phase. Figure 2.2 depicts the results from each of these three steps. The document corpus, in this case, consists of the two documents at the top of the figure; Doc 1 and Doc 2. The first two columns show the collection of terms ("term" in figure 2.2) and their associated document identifications ("docID" in figure 2.2) after the indexing of both documents. The next two columns show the alphabetical ordering of the terms with duplicate terms appearing consecutively. The final two columns show the completed inverted index with each term, the term's frequency and the term's postings list. As may be seen from the example in figure 2.2 the inverted index allows a user to locate a document based upon the occurrence of a word or an assemblage of words. For example, it can be determined from the inverted index that the word "Caesar" appears in documents 1 and 2. Furthermore, all that is required to obtain results from a query containing a multiplicity of words would be to collect the references in the postings list for each term and perform a join on each of these collections.

Information Retrieval System Evaluation

A standard measurement of the quality of an IR system is the somewhat etherial concept of *user happiness* [3]. This is a concept that codifies the recognition that

Figure 2.2: Inverted Index Creation Process (stopwords included for simplicity)

Doc 1: I did enact Julius Caesar: I was killed i' the Capitol: Brutus killed me.
Doc 2: So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious.

term	**docID**
I	1
did	1
enact	1
Julius	1
Caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
Brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
Caesar	2
the	2
noble	2
Brutus	2
hath	2
told	2
you	2
Caesar	2
was	2
ambitious	2

⇒

term	**docID**
ambitious	2
be	2
Brutus	1
Brutus	2
capitol	1
Caesar	1
Caesar	2
Caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
Julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

⇒

term:freq.	→	**postings**
ambitious:1	→	2
be:1	→	2
Brutus:2	→	1 → 2
capitol:1	→	1
Caesar:2	→	1 → 2
did:1	→	1
enact:1	→	1
hath:1	→	2
I:1	→	1
i':1	→	1
it:1	→	2
Julius:1	→	1
killed:1	→	1
let:1	→	2
me:1	→	1
noble:1	→	2
so:1	→	2
the:2	→	1 → 2
told:1	→	2
you:1	→	2
was:2	→	1 → 2
with:1	→	2

an exhaustive search of a document corpus with an eye towards achieving high recall will not always return results to the user in an acceptable time frame or, depending on the search logic, will contain erroneous entries due to overfitting of the data. The user would most likely be happier with a system featuring lower precision, higher

Algorithm 1: Creating an Inverted Index [3]

input : D where d is a document in corpus D
output: Ω where ω is an inverted index entry in inverted index Ω

```
1 begin
2     forall d ∈ D do
3         Create W_d, the list of words in document d.
4         Add W_d to the term-document list L.
5     Sort L according to the alphabetical order of W, where W = ⋃_{d∈D} W_d
6     forall w ∈ L do
7         Create inverted index entry ω_w with postings list π_w a reference to all of
          the documents in which w is found
8 end
```

recall, and a short query time, relying on their instincts to perform refinements on the results once they have been returned [3]. Furthermore, sometimes the priorities of the user base favor higher recall and a willingness to wait longer for it. In sum, user happiness is highly dependent upon the behavior of the users and the size and behavior of the corpus.

Depictions of the way that precision and recall vary in relation to one another are important in order to conduct measurable, scientific investigations into the quality of an IR system. Figure 2.3 shows two plots; one where the results of a hypothetical information retrieval are perfect, and one more realistic depiction. For the perfect results, the precision vs. recall plot does not deviate from a precision of 1 for all recall levels. For the other results (Path Length <12, 12>, to be discussed in the chapter entitled "Experimental Evaluation"), it can be seen that when the results are not perfect but do exhibit a distribution of relevant documents that has more relevant documents positioned near the top of the results, for greater recall levels a corresponding decrease in precision will occur.

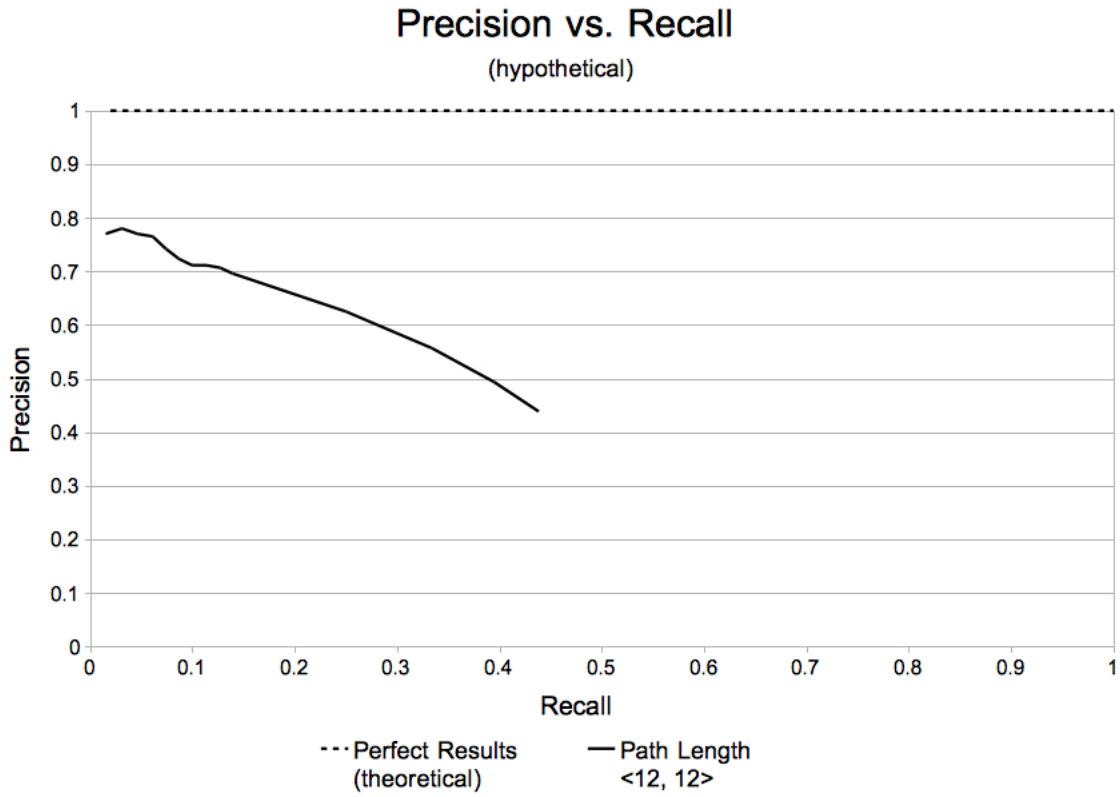


Figure 2.3: Precision vs. Recall: hypothetical

Frequent Pattern Mining

Frequent Pattern Mining has its genesis in commercial market basket analysis. In this domain commercial entities, such as stores seeking to optimize product placement and/or procurement, attempt to maximize the utility of an array of business factors by analyzing which products are sold together, i.e., appear in the same "basket." The objective of *frequent itemset generation* is to find all of the items that occur together, i.e. are purchased in the same transaction, in a frequency that satisfies a minimum support threshold [9]. Using this information, the business may be able to optimize their supply strategy, store product placement, etc.

The relationship between transactional frequent pattern mining and textual frequent pattern mining becomes readily apparent when one considers a document to be analogous to a commercial transaction and the words in the document to be the items within this transaction. Hierarchical clustering for text documents using frequent sets of co-occurring terms was introduced into data mining literature by Fung, et al. [10]. Their method reduces the dimensionality of the word space in which documents are represented by performing the clustering based on frequent itemsets discovered in the documents, instead of using the space consisting of all individual words.

An intuitive way of obtaining frequent patterns would be to generate singular candidate items, check them against the minimum support threshold to see if they can be added to the list of frequent itemsets as 1-itemsets, and from these results generate their 2-combinations and again check the dataset to see if they can be added to the list of frequent itemsets as 2-itemsets, repeating this process of candidate generation and verification until the frequent combinations are exhausted. This is called the Apriori method [9], and is generally referenced as frequent itemset generation with candidate generation, since a set of candidates is generated at each step. Since text corpora tend to be very large, with large numbers of frequent itemsets, this method is very expensive since it requires a complete search through the corpus at each iteration. Instead we prefer a method that does not require this time consuming process of candidate generation and stepwise verification. The method we have chosen is called FP-Growth (Frequent Pattern Growth) and will be discussed in the section entitled "FP-Growth Frequent Itemset Mining."

Support

Before presenting FP-Growth we want to introduce *support*, which is a measure that is commonly used in frequent pattern mining. Since we are concerned with text,

this could translate to the certainty that any two words appear together. *Support* is the probability that a transaction in the set of transactions contains the union of items A and B. Equation 2.4 shows that the support of the implication $A \Rightarrow B$ (the presence of A implying the presence of B) equals the probability of the appearance of the union of items A and B in a transaction. To be designated frequent, the individual items have to have been present in a frequency that exceeds a pre-specified threshold. We call this the *minimum support threshold*.

$$support(A, B) = \frac{count(A \cap B)}{total\ number\ of\ transactions} \tag{2.4}$$

FP-Growth Frequent Itemset Mining

FP-Growth [1] is a method of mining frequent patterns in a dataset without generating candidates, as is done in the popular Apriori method [9] [1]. For example, consider the test data in table 2.3. The first column represents the ID number of the transaction. The second column represents the items in the transaction. The third column represents the items in that transaction that have been found to be frequent, for a support threshold of 3, in an aggregation of the transactions in an order based on their popularity among the buyers. This total ordering is obtained by traversing the entire dataset and tallying the occurrence of each item, then sorting this list by the aggregate item frequency.

Algorithm 2 shows the pseudocode for the construction of an FP-Tree. It makes use of algorithm 3 to perform insertion of frequent items into the proper locations in the tree. The FP-Tree consists of several structure types. The first are the vertices, which each consist of a label and a support value. The labels correspond to the frequent items under consideration along with the support levels of the items in the

Algorithm 2: Construction of an FP-Tree [1]

input : Δ, the database of transactions.
output: An FP-Tree Ξ depicting the aforementioned frequent items accordingly.

```
1 begin
2     Scan Δ collecting the set δ.
3     Sort δ in support count descending order as L, the list of frequent items.
4     Create the root Ξ and label it "null."
5     for each transaction δ in Δ do
6         let P be the sorted list of elements in δ.
7         for each p in P do
8             call insert_tree(P, Ξ)
9 end
```

Algorithm 3: Insert a frequent item into the tree (**insert_tree**) [1]

input : P, the sorted transaction database entry containing only frequent items and Ξ, an FP-Tree
output: The FP-Tree Ξ with each element p in P inserted

```
1 begin
2     forall p in P do
3         if Ξ has a child N such that N.label = p.label then
4             increment N's count by 1
5         else
6             create a new node N
7             set N's count to 1
8             link N to the previous node evaluated
9 end
```

Table 2.3: A Database of Transactions [1]

TID	Items Bought	(Ordered) frequent items
100	f,a,c,d,g,i,m,p	f,c,a,m,p
200	a,b,c,f,l,m,o	f,c,a,b,m
300	b,f,h,j,o	f,b
400	b,c,k,s,p	c,b,p
500	a,f,c,e,l,p,m,n	f,c,a,m,p

dataset when arranged in accordance with the construction algorithm. The second, and most obvious structure is the tree itself. The tree is a monotonic structure, which means that as any leaf is traced to the root of the tree, the supports of the nodes are strictly non-decreasing. This is a property that is vital to the mining of the FP-Tree the reason for which will become clear when we discuss mining the FP-Tree. The third structure is the header table. This table includes all of the unique labels in the tree along with their aggregate supports. It is sorted by these aggregate supports. This allows the algorithm to locate the leaves of the tree without having to traverse the entire tree structure.

Figure 2.4 depicts the construction of the FP-Tree after the first transaction has been processed. It is here where the total ordering becomes important. If this total ordering of frequent items is not used for the sorting of the items in each transaction before the insertion into the FP-Tree then the algorithm may not construct a tree that exhibits a property of monotonicity. The items in the transaction are added to the tree in this order with each successive member of the transaction attached to its antecedent item.

Figure 2.5 depicts the construction of the FP-Tree after the second transaction has been processed. Since items f, c, and a are common to both the first and second transactions, the construction proceeds down the same path, increasing the support

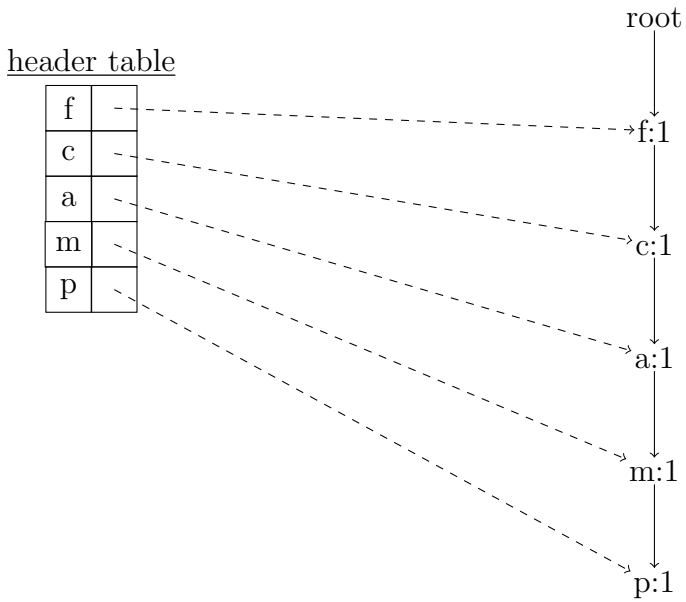


Figure 2.4: FP-Tree After the Insertion of Transaction 1

of each by 1 until there is a difference in transactions. At this point a new branch is formed and the construction proceeds as an extension of this branch. Also, links are added between vertices with the same label on different branches of the tree to facilitate linkage from the corresponding item in the header table through all vertices with that label.

Figure 2.6 depicts the completed FP-Tree for this example. Notice that the tree is monotonic in the supports of each individual node. This is an important feature of this structure and indeed facilitates the next step: mining the FP-Tree.

The purpose of algorithm 4 is to mine the FP-Tree, generating the frequent patterns. Simply, this proceeds by successively mutating the tree into two alternating forms; the prefix paths and the conditional FP-Tree. *Prefix paths* are the paths that contain a specific item at the end of their paths. They are arranged as a tree that is generated by collecting all of the paths in the tree that end with a specific item. Figure 2.7 depicts the prefix paths for item p. If an item is found to be frequent in this

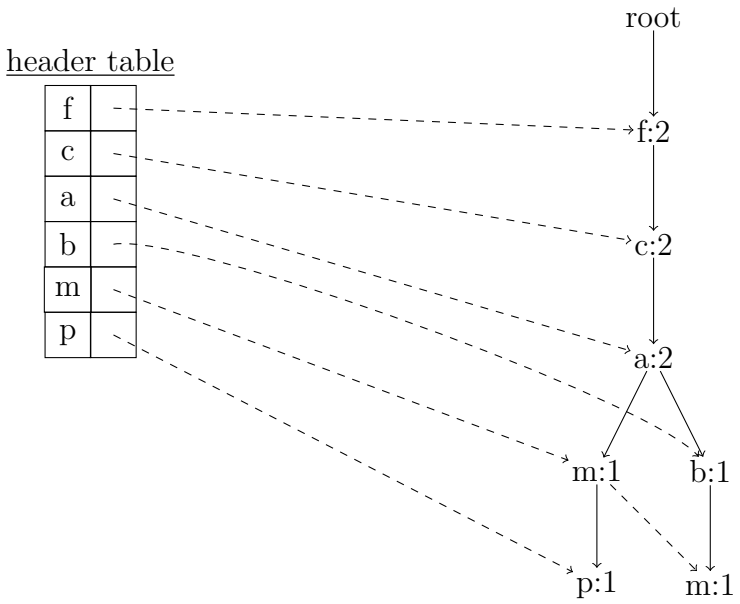


Figure 2.5: FP-Tree After the Insertion of Transaction 2

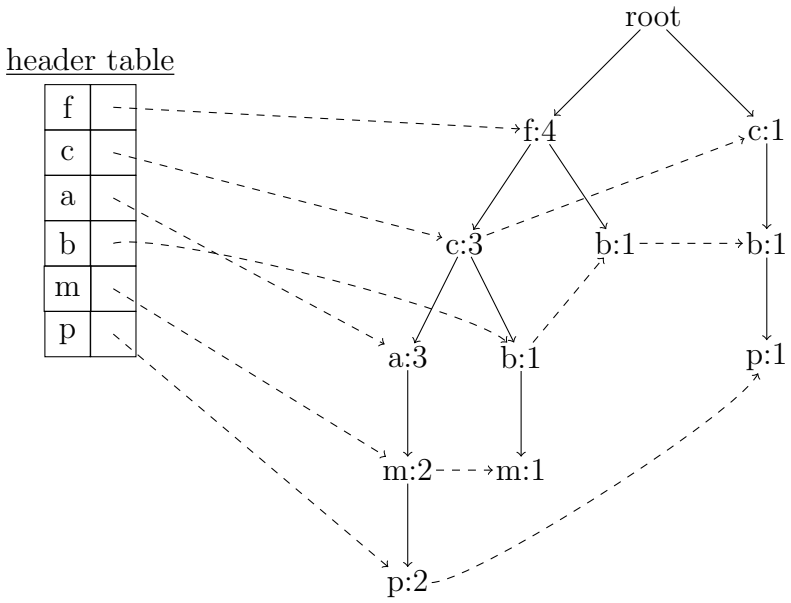


Figure 2.6: The FP-Tree after all transactions have been inserted

configuration, then it is added to the list of frequent items. The *conditional FP-Tree* depicts the prefix paths with the item characterizing the prefix paths removed and the

weights adjusted. The construction of the conditional FP-Tree starts with the prefix paths and adjusts it for the items that are conditional upon the itemset for which the prefix paths were obtained. If an item is found to be infrequent in this configuration, it is deleted. Figure 2.8 depicts the conditional FP-Tree for the 1-itemset p.

Algorithm 4: Mining of an FP-Tree [1]

input : Ξ, an FP-Tree and $\phi_{condition}$, an itemset upon which Ξ is conditioned.
output: $[\phi \mid \Phi]$ where ϕ is a frequent pattern in the set of frequent patterns Φ

```
1  begin
2      if Ξ contains a single path P then
3          forall combinations β of the nodes in path P do
4              Generate pattern β ∪ β_condition with
               support_count = minimum support count of nodes in β
5      else
6          Generate pattern φ = β_condition ∪ β_condition with
           support_count = β_condition.support_count
7          Construct β's conditional pattern base and then β's conditional
           FP-Tree Ξ_β
8          if Tree_β ≠ ∅ then
9              call FP-growth(Ξ_β, β)
10 end
```

Taxonomies and Frequent Patterns in Text Processing

Pedersen, Banerjee, and Patwardhan [11] presented a method of word-sense disambiguation based on assigning a target word the sense that is most related to the senses of its neighboring words. Their methodology was based on finding paths in concept networks composed of information derived from the corpus and word definitions.

Wan and Angryk [12] proposed using WordNet to create context vectors to judge relationships between semantic concepts. Their measure involves creating a geometric

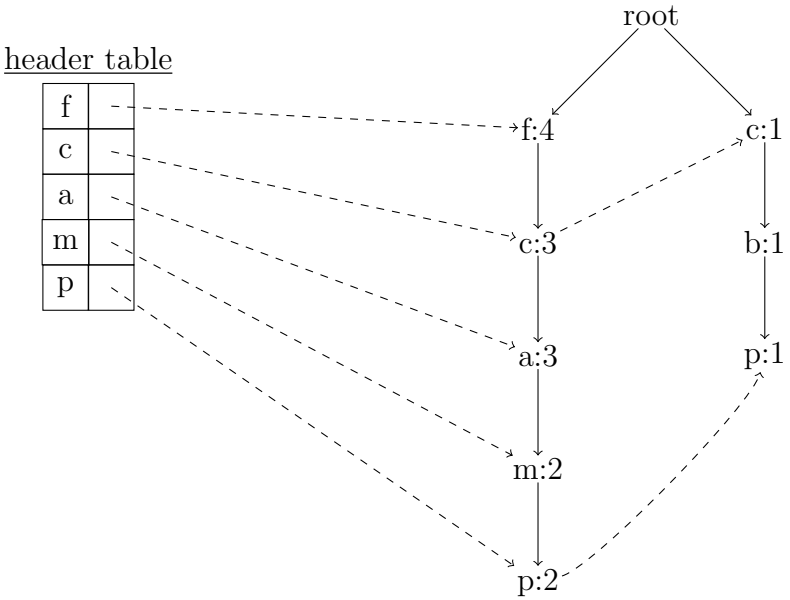


Figure 2.7: A Prefix Paths for Item p

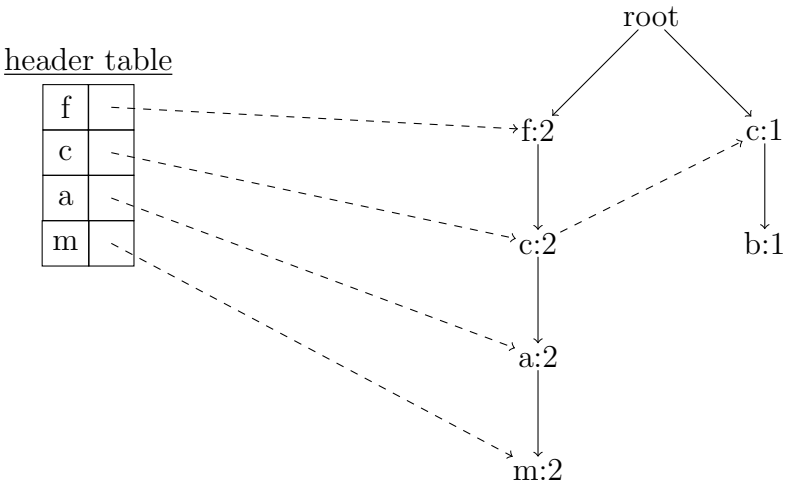


Figure 2.8: A Conditional FP-Tree for 1-itemset p

vector representation of words and their constituent concepts and using the cosine of the angle between concepts to measure the similarity between them.

Perhaps one of the most relevant ideas comes from Resnik [13] who created a measure of semantic similarity in an IS-A taxonomy based on shared information

content. The relevance comes from the IS-A taxonomy idea, since this betrays the use of subclass/superclass relationships within the measure.

Measuring semantic distance between network nodes for word-sense disambiguation was addressed by Sussna [14], who also clarified the perils of inaccuracy in keyword search.

Jiang and Conrath [15] combined lexical taxonomy structures with statistical information hewn from the corpus. In doing so they were not reliant on either of the methods for a measure of semantic similarity, but rather both. An essential idea to them was that words used in a subject-specific corpus would be more likely to mean some things based on how they are normally used in relation to the subject address in the corpus.

Lin [16] proposed a measure of similarity of words based on the probability that they contain common independent features.

Widdows and Dorow [17] presented a graph method for recognizing polysemy. Word-sense disambiguation provided motivation for the technique, which is based on creating graphs of words, using the words' equivalence relations as the connections.

Hossain and Angryk [18] created a system (GDClust) to perform document clustering based on frequent senses. Their work used the Apriori paradigm to identify frequent subgraphs of document graphs created using WordNet in a manner similar to the way we use it. Akbar and Angryk [19] used the FP-Growth algorithm to organize documents in a corpus hierarchically.

Qian et. al [20] developed an algorithm for text categorization using hyperclique patterns, which are association patterns that contain items that are "highly affiliated" with each other. "Highly affiliated" in this context means that the presence of any item in a transaction "strongly implies the presence of every other item that belongs to the same hyperclique pattern." [20]

Document clustering using frequent itemsets was explored by Fung, Wang, and Ester [10]. Their work aimed to address specific problems in text document clustering, including high dimensionality, high volume, and the facilitation of use. They used frequent itemset mining to discover these itemsets and use them to create a hierarchical topic tree.

APPROACH

Our approach required the construction of a complete information retrieval system. This chapter provides a description of the principles of operation of the constituent pieces of this system along with a description of the system's operation.

Overview

Our approach is based on the concept of an *abstraction path*, which is a path of term derivation, from most general to most specific, that corresponds to a taxonomy of terms. For our purposes, the WordNet hypernym/hyponym relationship of nouns form this taxonomy. We restricted our analysis to nouns to simplify the relationships in order to save processing time and storage space.

Figure 3.1 shows the relationships of the artifacts of the entire system. All of the items in this figure will be described in this section.

The following list is an overview of the important systems and data structures that make up our approach:

- Graphing Module: It is the responsibility of this module to accept word vector input for any number of documents, create *document graphs* which are depictions of the taxonomic relationships among words in the document, create the corresponding *document trees* which are tree-representations of the aforementioned document graphs, and extract the *abstraction paths*, the unique paths that make up a document tree, from the document trees.

- Master Document Tree Miner: This module performs the mining of the *master document tree*, which represents a combination of all of the document trees,

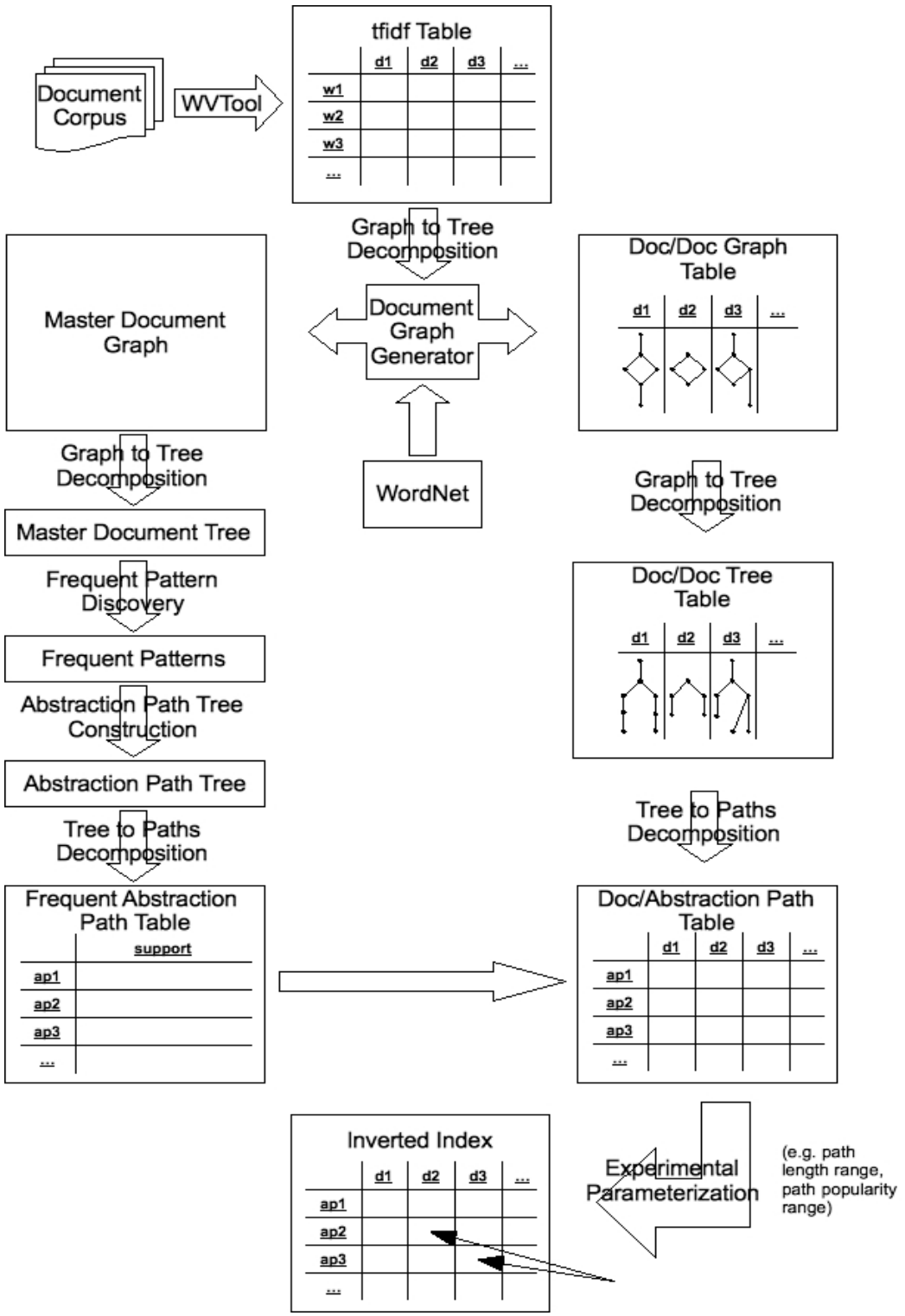


Figure 3.1: The Ontology Abstraction-Based Information Retrieval System

and produces *connected frequent paths*, which are connected paths found to be frequent through frequent pattern analysis.

- Frequent Abstraction Path Table: *Frequent abstraction paths*, which are connected frequent paths that are extended to the root vertex, from the graphing module are converted into a frequent abstraction path hash table, which is a data structure that is compatible with the document abstraction path inverted index for the purposes of taking an intersection of both.

- Document Abstraction Path Table: This is an inverted index that is created from the abstraction paths taken from each individual document tree.

- Inverted Index: This inverted index is the intersection of the index based upon frequent abstraction paths and the index based upon the document abstraction paths.

WordNet

WordNet is an on-line lexical reference system created by a group of lexicographers [21]. The creators of this system intended to create an exhaustive ontology of the English language that depicts words in their environment of mutually defined relationships. With respect to the relationships with which we are concerned, WordNet forms a directed acyclic graph (a hierarchy) of relationships among words.

In WordNet, words are related as a collection of *synsets*, which are groups of words that all carry a meaning that is similar enough to render each word in the synset interchangable to some degree, i.e., they are synonyms. The relationships among these synsets are then identified and the synsets are placed in the hierarchies and graphs that depict broader relationships. These relationships include *synonomy*, *meronymy*, *hypernymy*, *hyponymy*, *polysemy* and others of greater complexity. We

have chosen to exploit the following subset of all of the WordNet relationships for our approach.

Synonomy: Synonymy is conveniently tracked through the use of the aforementioned synsets. Items in the same synset have meanings that are the same or are very close to each other. When a word is profiled in WordNet it essentially loses its identity as a lexeme and exists in the system in its corresponding synsets. Since words sharing a common synset have the same meaning, the rest of the dynamics exhibited by any word in the synset hold for any other word in the synset.

Hypernymy: A hypernym of a word is a superordinate form of that word. Being like a parent object, in object-oriented parlance, the word bestows upon all of its children its own attributes. An example of this is the relation ship between the words "cat" and "animal." "Animal" being the hypernym of "cat." Since there could conceivably exist repeating cycles of tautologies, WordNet limits itself to 18 levels of hypernymy, so there is never a distance greater than 18 from any synset and the synset that represents the word "entity." Being the most general and abstract hypernym, the word "entity" does not convey much information at all besides being a convenient moniker for anything in existence. However, a hypernym may be used in place of any of its children with perhaps some loss of specificity but without loss of accuracy.

Hyponymy: Hyponymy is the opposite of hypernymy. Exhibiting a greater degree of specificity, a hyponym carries more information than its hypernym. In reference to the previous example, "cat" is a hyponym of "animal." Unlike a word's

hypernym, a hyponym may not be substituted for its hypernym without the loss of accuracy due to the additional constraints that are imposed by the higher degree of specificity. Also, with respect to the word 'entity' as the most general hypernym, there is clearly a multiplicity of terms in the category of "most specific."

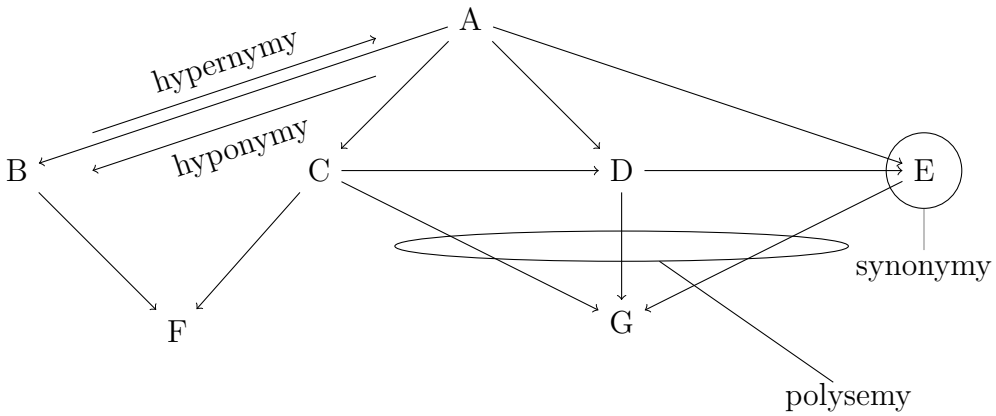
Polysemy: If a word appears in more than one synset, that word is said to be *polysemous*. By definition, polysemous words have more than one meaning, and therefore belong in a multiplicity of synsets. For example, the word 'box' is polysemous in that it describes a type of container as well as the pugilistic sport of 'boxing.' This is a source of great ambiguity, and the consequent categorization difficulties, in the computational processing of language.

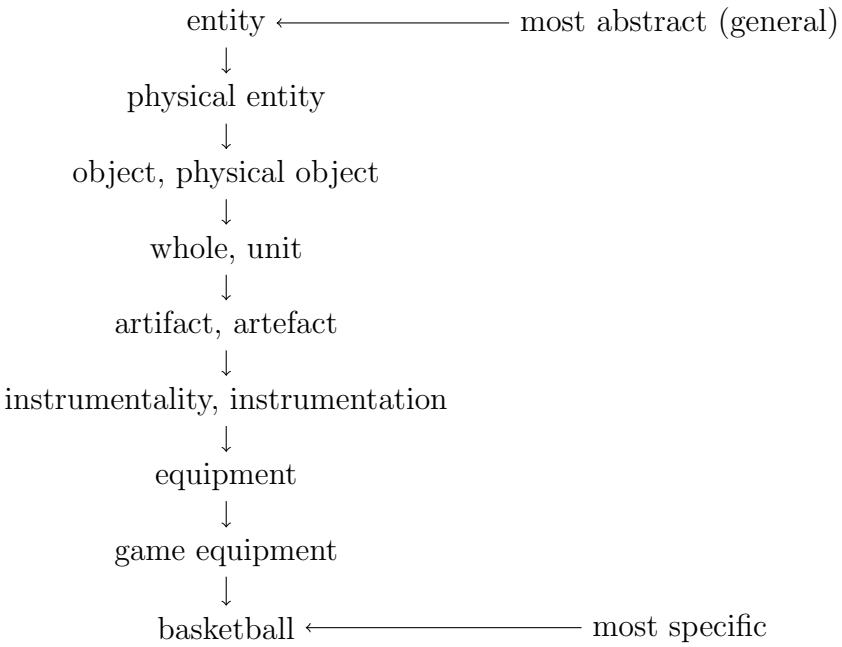
Consider the hierarchy in figure 3.2 as an example of the aforementioned relationships as they exist in WordNet. Vertex A represents the most abstract concept in the hierarchy, and is located at the top. This figure captures all of the important types of relationships with which we are concerned including polysemy (e.g. node F's relationships to nodes B and C), hypernymy (e.g. node A's relationship to node C), and hyponymy (e.g. node C's relationship to node A). Node A is analogous to the most abstract term in WordNet ('Entity'). We present this hierarchy as a representation of an ontology for a vocabulary over the tokens $\{A, B, C, D, E, F, G\}$.

Abstraction Paths

To illustrate the idea of an abstraction path, the WordNet-derived abstraction path corresponding to a sense of the word 'basketball' is:

Figure 3.2: The types of relationships that compose WordNet





The term 'entity' is the most general noun in WordNet, therefore all abstraction paths include this word.

Nouns in WordNet have a subclass/superclass relationship which may be exploited to create these relationships. Therefore, when an assemblage of these paths, in tree form, is created, the resultant data structure is available for association rule discovery

via the FP-Growth algorithm. The mining of this structure will produce frequent paths which, when converted to their corresponding abstraction paths, may be used to create an inverted index for information retrieval from an indexed corpus. This approach involves the use of several established mechanisms of text analysis.

Preprocessing of Documents

It is the responsibility of the preprocessing subsystem to condition each document for processing by the main system. The structure suitable for processing is the word-vector, which is a depiction of how important each term is to each document. The conditioning includes:

1. Removing stopwords.
2. Checking for the existence of each remaining word as a noun in WordNet and discarding the words that are not identified as being such.
3. Creating the word vectors.

Step 2, detailed in algorithm 5, represents a departure from the modus operandi of information retrieval systems. This is normally where word stemming would occur. However, since we use WordNet to resolve word forms, we do not stem the words, but instead rely on WordNet to resolve word forms into synsets. Furthermore, in early experimentation, stemming proved detrimental to our procedure because the use of the extant stemming algorithms produced word lemmas that were unusable for querying WordNet. For example, the word 'swimming' would be stemmed to the pseudo-word 'swimm' which does not exist in WordNet but may be a word that is critical to identifying the character of a document. In contrast, WordNet's synsets

contain the majority of the words in their un-stemmed form and therefore offers a convenient alternative to stemming. In this example, the gerund 'swimming' is stored in the same synset as its canonical lexeme 'swim' or, if appropriate, retained in its gerund form.

Performing lexeme to synset resolution via WordNet is not the ideal situation, as, for example, there are several forms of the word 'swim' that can be treated as nouns. For example, 'going for a swim,' 'your swimming is excellent,' and 'I took several swims' are all noun forms of the word 'swim' that would perhaps lead in different directions where the WordNet hierarchy is concerned. Ideally, these would all resolve to the same noun, but as of now we have no way to guarantee this.

Algorithm 5: Preprocessing a Document Corpus

input : $d \mid D$ where d is a document in document corpus D.
output: $t \mid T$ where t is a file containing word vector entries t_w where w is a word in document d.

```
1 begin
2     forall d ∈ D do
3         Remove the stopwords.
4         Check for the existence of a corresponding noun in WordNet.
5         Calculate the Term Frequency (tf) information.
6     Calculate the Term Frequency Inverse Document Frequency (tfidf)
      information
7     forall d ∈ D do
8         Read the word vector output into t_d.
9 end
```

Word Vector Creation

A word vector is a structure that maps each word to the documents in which each word appears, assigning an importance measure to this word/document pairing. Intuitively, the cardinality of a word with respect to a document would be the indicator

of the importance of that word to that document. But this simple measure fails to account for documents that have a higher total number of words or how important any word is to an entire corpus. Larger documents may contain a greater diversity of words, thereby skewing the results of this simple term frequency measure in their favor. To mitigate this effect we use a normalized measure of term importance, namely the *tfidf* measure.

The *tfidf*-based word vector excels as an importance measure because, in addition to being normalized with respect to the number of words in the file, the importance of a word is determined as an inverse to its prevalence in the entire corpus. In other words, to be weighted high, a word must have a high normalized term frequency (tf) value and a high normalized document frequency (df) value, meaning that its non-inverse document frequency is high and its corresponding inverse document frequency (idf) value is low. As an example, the existence of stopwords provides a good analogy because the word "the" is present in a very high proportion of documents and will therefore have a very low idf value while also having a very high tf value. These properties of the word "the" will combine to produce a very low *tfidf* value. In contrast, if there exists a word that is only contained in one document out of the entire corpus, the tf value will be high for that document, indicating a high importance of this word for the document, but the idf value will be very low, increasing the importance of this rare word to the entire corpus.

Document Graphs

Using WordNet we create *Document Graphs*, which are hierarchical representations of each document's relevant subgraph, which is the subgraph of the WordNet hierarchy that contains the synsets whose words are found in the document and the

hypernyms of those words. The words that are found in the documents are called *keywords* and the hypernyms of those words are called *implicit words*. The construction of the document graphs for all of the documents in the corpus culminates in the creation of the *Master Document Graph* [18], which is the subgraph of the WordNet hierarchy that contains all of the synsets and synset relationships found in the entire corpus.

To create a document graph, for each word in the document, WordNet's hypernym paths are traced through the hierarchy to the top-most hypernym, which is the word 'entity'. By design, each of these path will eventually intersect. This intersection may occur anywhere between the term and the term 'entity.' Each document graph emerges as the "fingerprint" of the document in relation to the language ontology, meaning that it is a somewhat unique representation of each document's profile in relation to the WordNet ontology. Furthermore, the master document graph represents the fingerprint of the entire corpus in relation to the ontology.

Discovery of Document Abstraction Paths

Consider the example in table 3.1. In this data each letter may be considered as a unique word in an example document. We proceed in creating a master document graph as follows:

Table 3.1: Random Data. Each line represents a "document" with the "words" represented as letters from a through g.

Doc	Items
d_1	b f f b f f f e e g g b g
d_2	f b f b f b b
d_3	g b e b e f f e b g e f b e g e e e e b f b f g g b b b b b
d_4	f g b g f g f f g f g g g

1. Create a word-vector representation for each of the documents as each of their components of the vector space model using each letter's *tfidf* value (table 3.2).

2. Construct the document graphs and the master document graph.

Table 3.2: Example *tfidf* table based on the random data

doc \ word	b	e	f	g
d_1	0.231	0.154	0.385	0.231
d_2	0.571	0.0	0.429	0.0
d_3	0.367	0.300	0.167	0.167
d_4	0.077	0.0	0.385	0.538
$\sum_{i=1}^{4}$	1.246	0.454	1.364	0.936

Construction of Document Graphs

Document graphs contain the synset correlates of the words that are found in the documents. Again, we call these synsets the *keywords* and the synsets that are discovered via the document graph creation process are called *implicit words*. Initially, the synsets found in the document, along with their weights, are placed into an empty graph representation of the taxonomy. The document graph is then created by propagating the words' weights up to the root vertex. Using the graph in the following example as an analogy to the WordNet hierarchy, the construction of individual document graphs proceeds as detailed in algorithm 6.

Document Graph Construction for Document 1: The construction of the document graph for document 1 proceeds as follows:

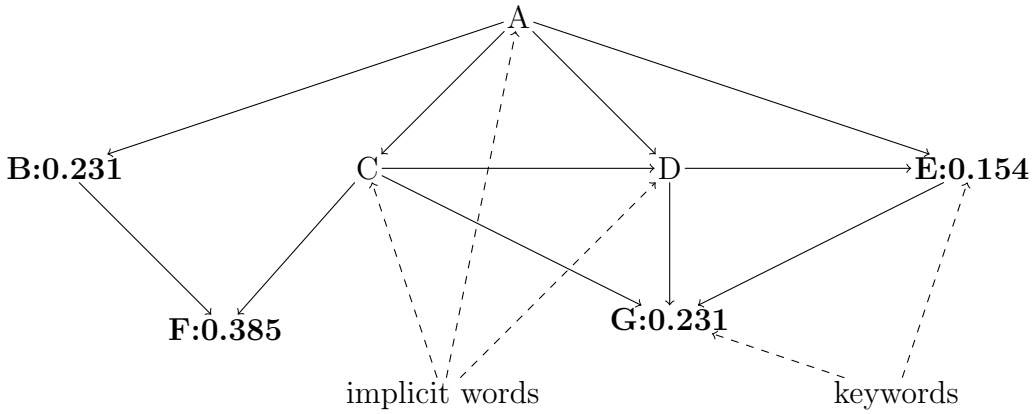
Algorithm 6: Creation of Document Graphs

input : Set of vectors $T = \{t_1 \dots t_{|D|}\}$, where $|D|$ is the number of documents and t_i denotes the vector representing the i^{th} document, and t_{ij} contains the *tfidf* value for the j^{th} word in the i^{th} document vector. ($i = 1 \dots |D|$ and $j = 1 \dots |W|$, where $|W|$ is the number of words in the corpus.)

output: A collection of document graphs $DG = \{dg_{d_1} \dots dg_{d_{|D|}}\}$ where $dg_{d_i}(V, E)$ is a document graph corresponding to document i.

```
1  begin
2      let DG = {∅}
3      for i = 1 to i = |D| do
4          let dg_{d_i} = a new, initially empty, document graph corresponding to
           document d_i
5          for j = 1 to j = |W| do
6              if t_ij > 0 then
7                  Fetch the S, the set of synsets corresponding to t_ij from
                   WordNet
8              while S ≠ {∅} do
9                  Process Synset Hypernyms (see algorithm 7)
10         DG = DG ∪ dg_{d_i}
11 end
```

1. Assign a weight to each node in the hierarchy whose corresponding token is also found in document 1's word-vector, based on the TF value. In the case of document 1 the document contains only B, F, G and E. The new weights can be seen in bold:



Algorithm 7: Process Synset Hypernyms

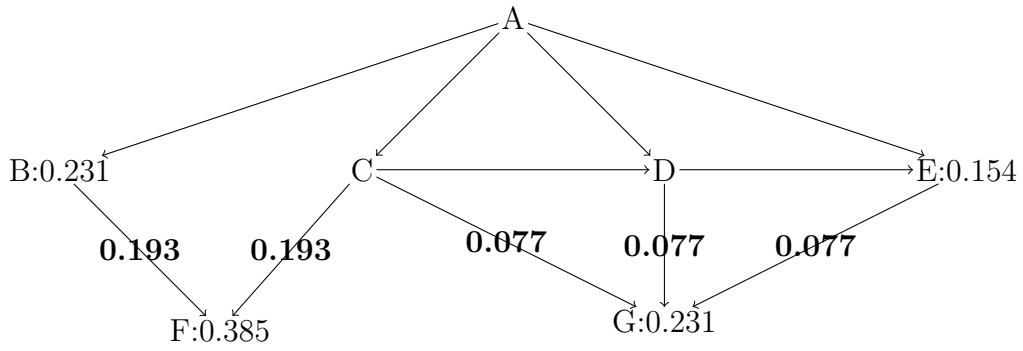
input : A document graph $dg = \{V, E\}$, where V is a set of vertices that contains pairs of elements: { *synset id*, *vertex weight* } and E is a set of directed edges, where each edge is a triple: { *hypernym*, *weight*, *hyponym* } a synset s and a word vector t where t_s is the vector entry for synset s in t.

output: A document graph dg with all of the partial *tfidf* values assigned.

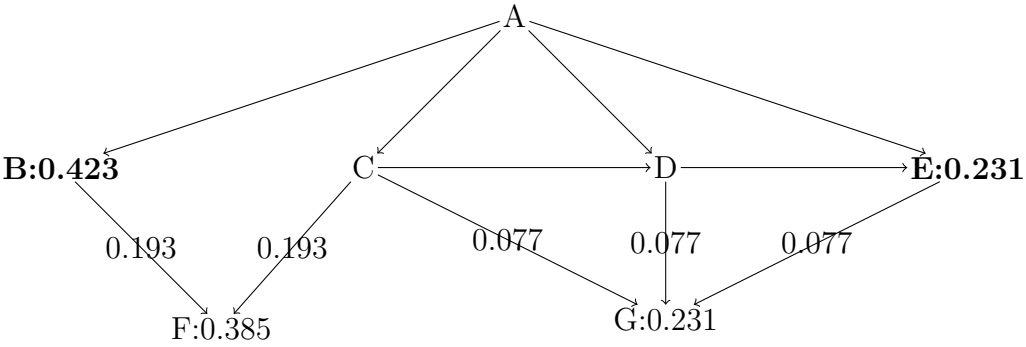
```
1  begin
2      let v_s be a vertex corresponding to synset s.
3      if v_s ∉ V then
4          add vertex v_s to dg and set its weight to t_s
5          fetch S', the set of hypernyms of s from WordNet
6          forall s' ∈ S' do
7              if v_s' ∉ V then
8                  add vertex v_s' to dg and set its weight to t_s'.
9                  add an edge from v_s to v_s'.
10                 increase v_s' by t_s / |S'|.
11         S = S ∪ S'
12     else
13         increase the weight of v_s by t_s.
14         increase the weight of all hypernyms of v_s by propagating its fractional
           weight upwards through the graph.
15     remove s from S
16 end
```

2. Proceeding from the bottom-most leaf vertices, propagate the value of each vertex upward to the edge connecting it to it's parent, weighting the incoming connections by using equation 3.1 to distribute the vertex's weight among these connections. The bold values represent the newly calculated connection strengths:

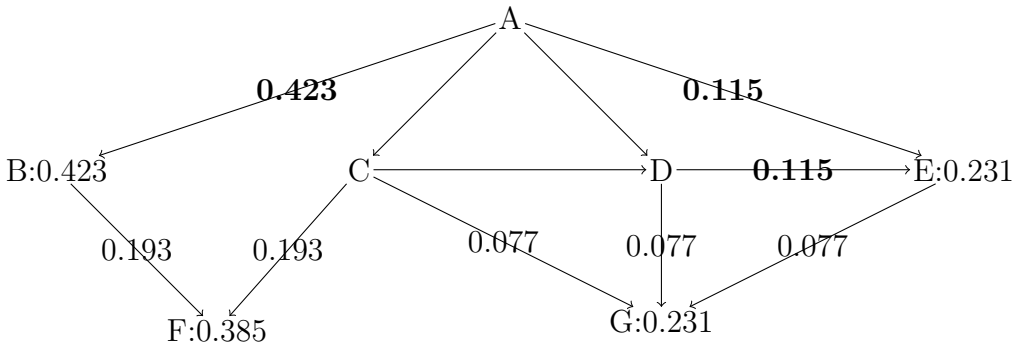
$$\frac{\textit{item support}}{\textit{degree of incoming connections}} \tag{3.1}$$



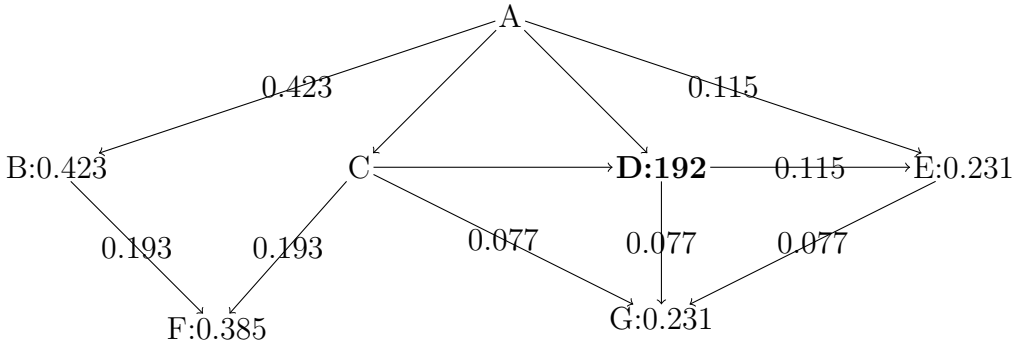
3. Proceeding from the bottom-most leaf nodes, propagate the value of each node upward to the edge connecting it to it's parent. Since all relevant outgoing connections out of nodes B and E have been assigned weights, we may increase the weights of these nodes with the sum of the weights of the edges incident to B and E respectively. No other vertices have this condition satisfied and so may not be updated at this time:



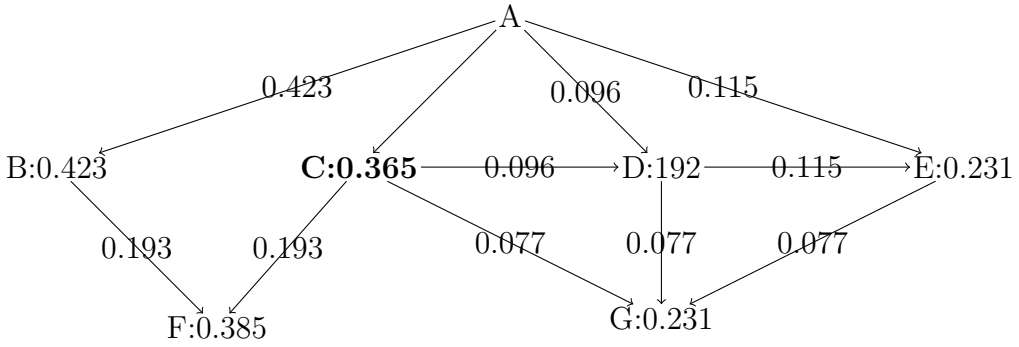
4. Proceeding from the bottom-most leaf nodes, propagate the value of each node upward to the edge connecting it to it's parent. Nodes B and E now propagate their weights to all of their incoming edges. Since B only has one incoming edge, this edge will be assigned the full value of B's weight. E has two incoming edges so therefore has its weight divided between these edges:



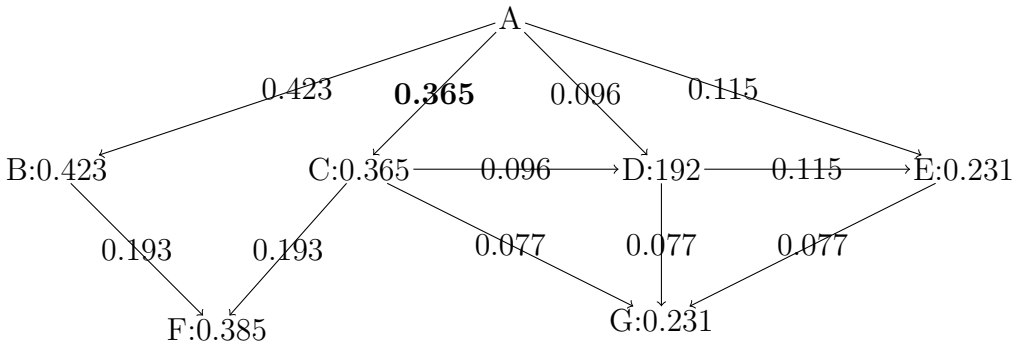
5. Proceeding from the bottom-most leaf nodes, propagate the value of each node upward to the edge connecting it to it's parent. In this case D may now be assigned a weight. Interestingly D was not originally in the document but is now in the abstraction profile of the document since one or more hyponyms of D are in the document:



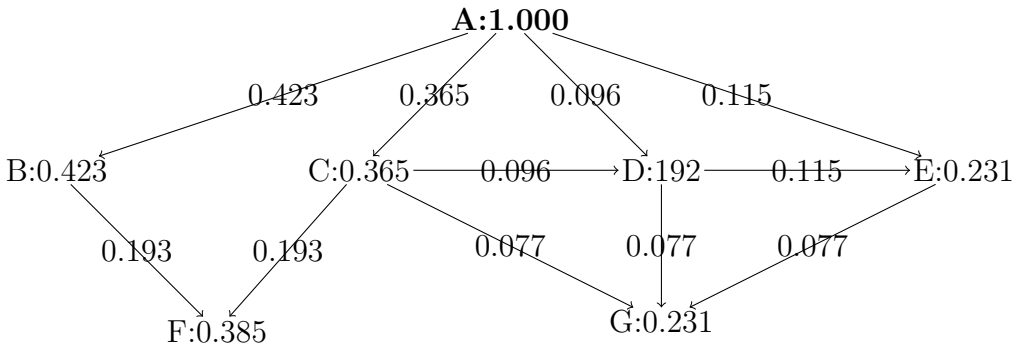
6. Proceeding from the bottom-most leaf nodes, propagate the value of each node upward to the edge connecting it to it's parent. C now gets its weight from its hyponyms D, G and F:



7. Proceeding from the bottom-most leaf nodes, propagate the value of each node upward to the edge connecting it to it's parent. C propagates its weight to its incoming edge, completing all weight prerequisites for the super-hypernym, A:



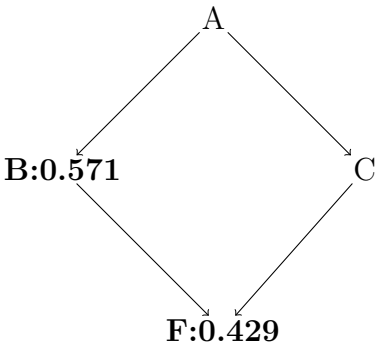
8. Proceeding from the bottom-most leaf nodes, propagate the value of each node upward to the edge connecting it to it's parent. The value of A is now equal to the sum of all TFxIDF values assigned at the beginning of the analysis:



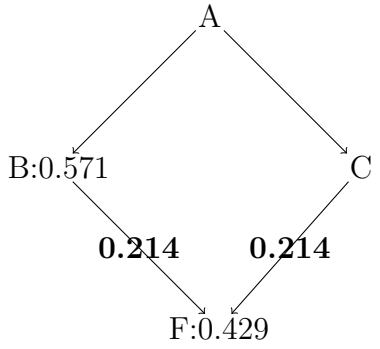
Note that only values from the original keywords need to be propagated. Since the abstract terms that do not belong to the set of keywords appearing in the *tfidf* table do not have their own *tfidf* values, propagation of *tfidf* values ceases when the most abstract implicit word in the hierarchy has been updated. As a result, the document graphs end up representing a very small subset of the entire ontology.

Document Graph Construction for Document 2: The graph construction for document 2 is different from that of document 1 in that it involves less of the entire ontology. Since this is the case, we create the associated document graph using only the relevant vertices and not the entire hierarchy. This example illustrates the property of this system that it only uses the relevant portion of the entire WordNet hypernym/hyponym hierarchy for document graphing and not the entire hierarchy. Such a parsimonious use of the entirety of WordNet is important to the conservation of memory and processor resources.

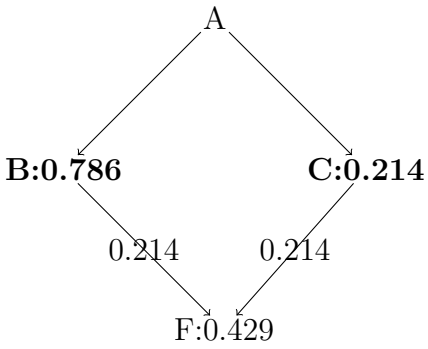
1. Assign a weight to each node in the hierarchy based on the TF value:



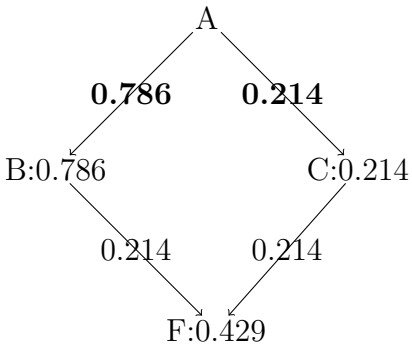
2. Edges C,F and B,F now both receive half of F's node weight, since they are the only two incoming connections:



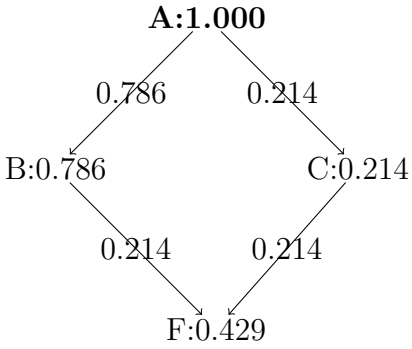
3. Since there is only one outgoing edge for both nodes B and C, and in both cases this edge has been assigned a weight, their respective weights are increased by the value of the weights of these outgoing edges:



4. B and C, having only one incoming edge each propagate their full node weights to these incoming edges:



5. Finally we have the nodes' full TFxIDF value at A:



To test the resource usage of the document graph creation process we collected time and memory usage numbers for the procedure running from 1000 to 20,000 documents, with an interval of 1000. Figures 3.3 and 3.4 show the time and memory usage for creating document graphs. These charts show that as the number of document graphs to create increases, the time to create the graphs has a tendency to increase in a polynomial fashion and the memory used in their creation increases in a linear fashion. The polynomial nature of the time increase is slight, but still exhibits polynomial characteristics. One may surmise that this may be due to the idiosyncrasies of the implementation. These efficiency properties are important because they show that document graph construction is a process that does not grow in its resource usage in a way that introduces intractability issues, rendering the system viable in terms of computational resources.

The Master Document Graph

An empty master document graph (MDG) is created at the beginning of the DG creation process and is updated with new information (vertices, weights, and connections) during the creation of each document graph. This is because, in order to enforce universally unique labeling for the vertices, the labels are created first in the MDG and given a sequential numerical label. Then the corresponding vertex labeling is used on the DG being created. Such system-wide unique vertex identifiers allow coherence within the entire system as they enforce unity in synset reference. This will become important in the extraction of frequent abstraction paths.

During the construction of each document graph the Master Document Graph (MDG) is incrementally updated. This results in the weighted hierarchies below. The MDG will represent the fingerprint of an entire corpus in relation to the ontology. Figure 3.5 shows the completed document graph for the running example.

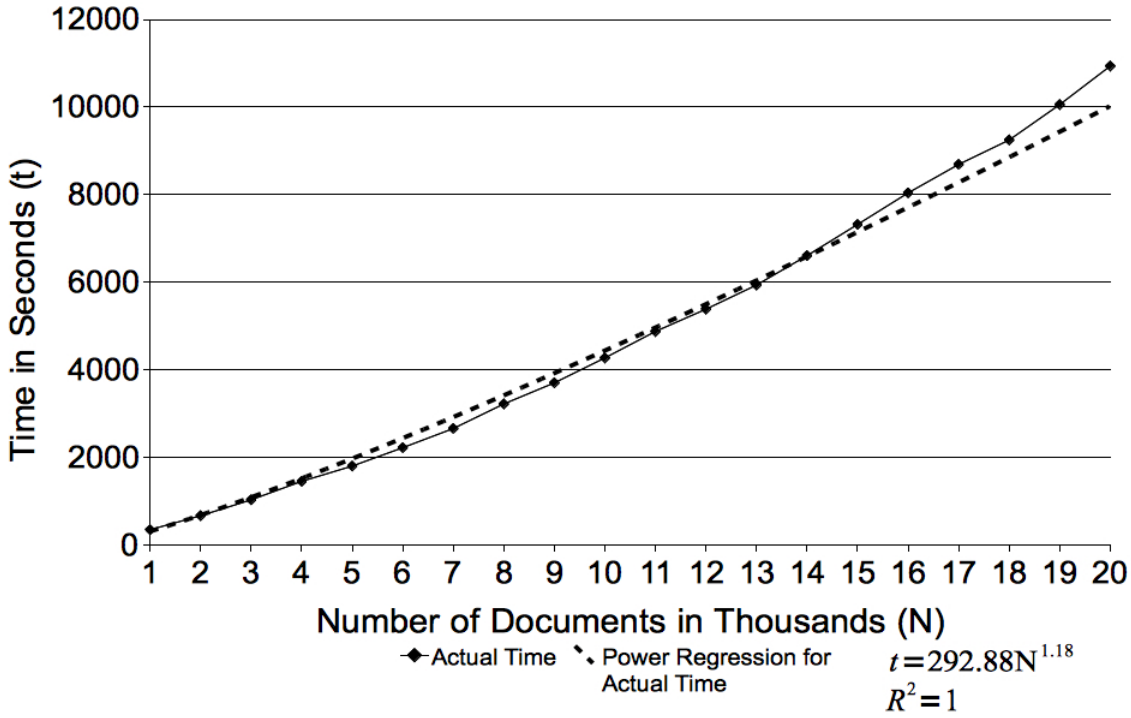


Figure 3.3: Time to create document graphs

Document Abstraction Paths

Upon creating the document graphs, the *document trees* (DTs), which are derived from each document graph are created. These structures facilitate matching which abstraction paths correspond to which document. These abstraction paths are the main entity upon which our information retrieval system is created.

It is important to note that order of vertex splitting is important to our process, since an error in this order may result in the misrepresentation of the weights of some vertices in the resulting tree. For our purposes, we use a variation on a depth-first strategy that favors splitting vertices with an out-degree of zero before all others. The next priority is to split vertices whose children have already been split. Algorithm 8 shows the pseudocode for this tree creation process.

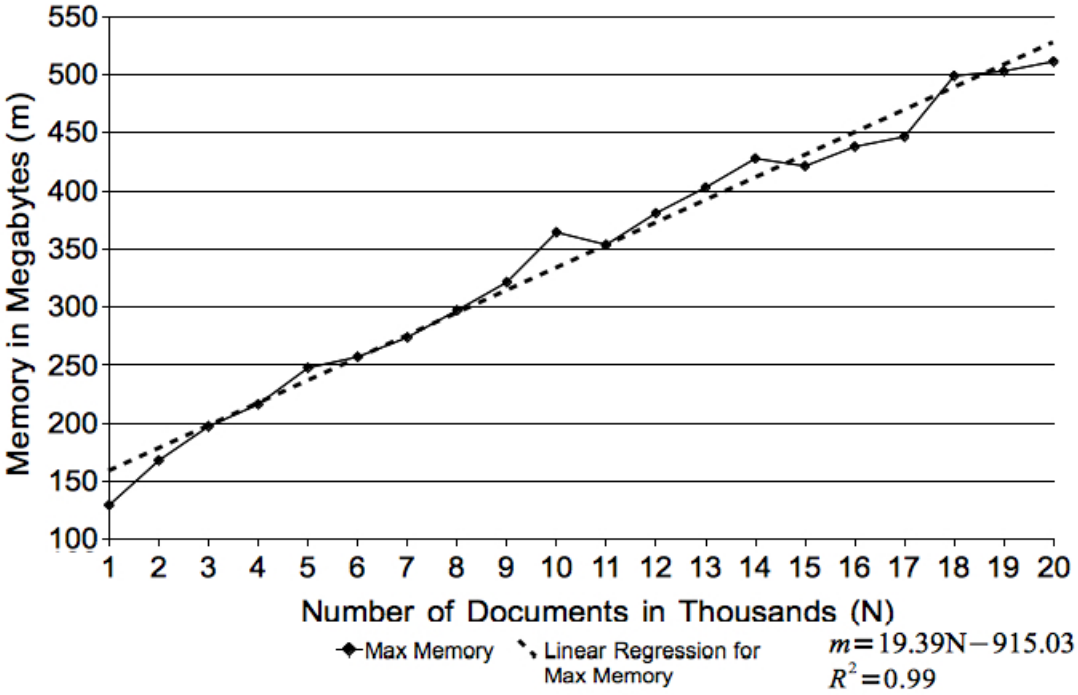


Figure 3.4: Memory usage for creating document graphs

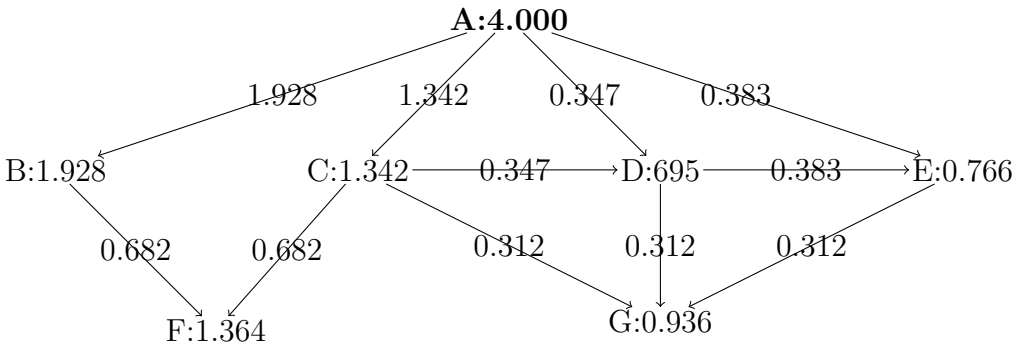


Figure 3.5: The Master Document Graph

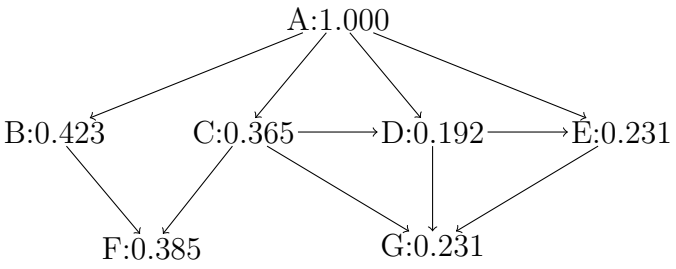
The following example shows the document tree creation algorithm (algorithm 8) at work on our running example. The illustrations corresponding to each step show the hierarchy's transformation (the graph), the vertex currently being evaluated ("Pop") and the state of the stack after the current iteration ("Stack"):

Algorithm 8: Create a Document Tree

input : A document graph $dg = \{V, E\}$.
output: A document tree DT.

```
1  begin
2      let STACK_v be a stack of vertices from dg
3      push the root vertex of dg on to the stack
4      while STACK_v is not empty do
5          pop v from STACK_v
6          if v has unmarked children then
7              push v on to STACK_v
8              push the unmarked children of v on to STACK_v
9          else if in-degree(v) > 1 then
10             split v into in-degree(v) vertices
11             mark all new vertices
12             clear the marks of all of the children of v
13             push all of the children of v on to STACK_v
14         else
15             mark v
16 end
```

1. The root of the hierarchy is pushed on to the stack, only to be popped at the first iteration and evaluated. Since none of its children are marked it is pushed back on to the stack, followed by each of its children (vertices B, C, D, and E).



Pop: A

Stack

E

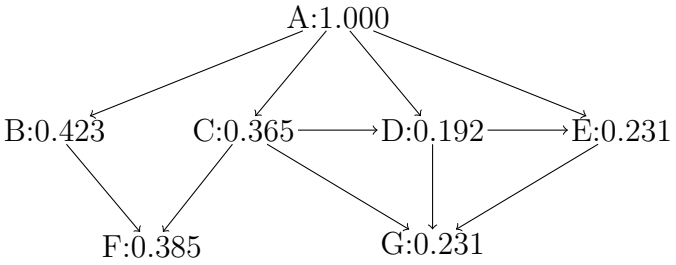
D

C

B

A

2. Vertex E is popped from the stack and found to have multiple incoming edges. However it also has an unmarked child, so it is pushed back on to the stack, followed by its child vertex G.

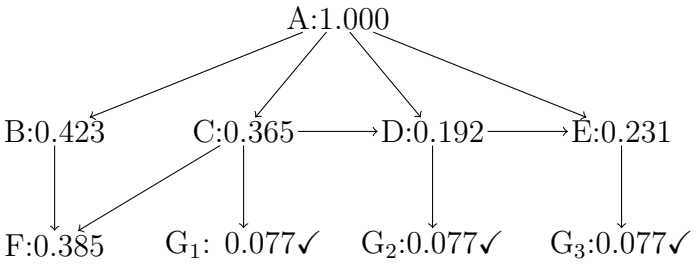


Pop: E

Stack

G
E
D
C
B
A

3. G is popped from the stack and found to have both multiple incoming edges and no children and consequently is split into three vertices, each having one third of the original weight of vertex G. Each of these vertices is marked.

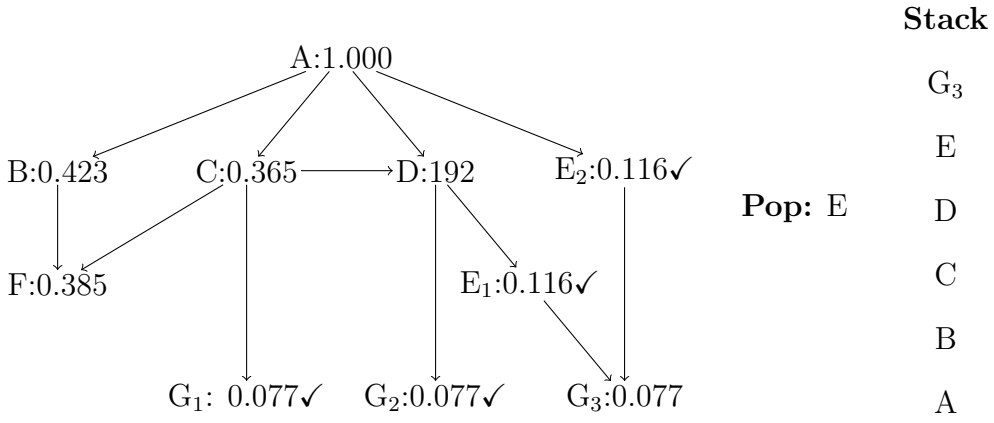


Pop: G

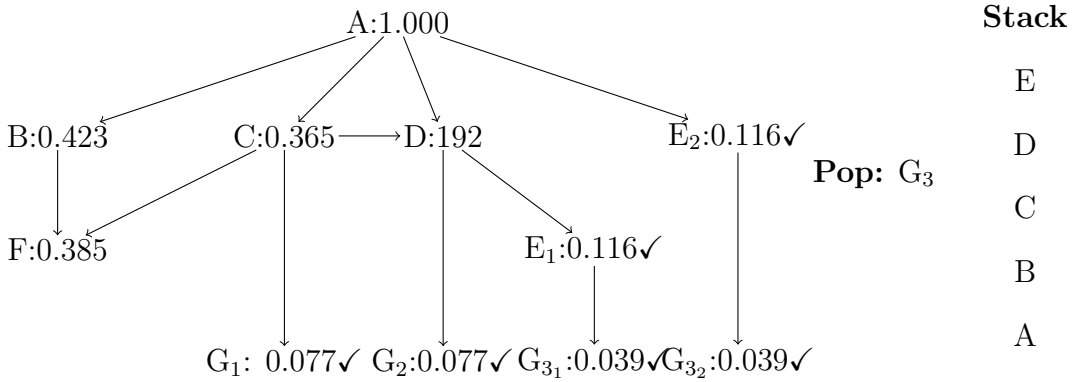
Stack

E
D
C
B
A

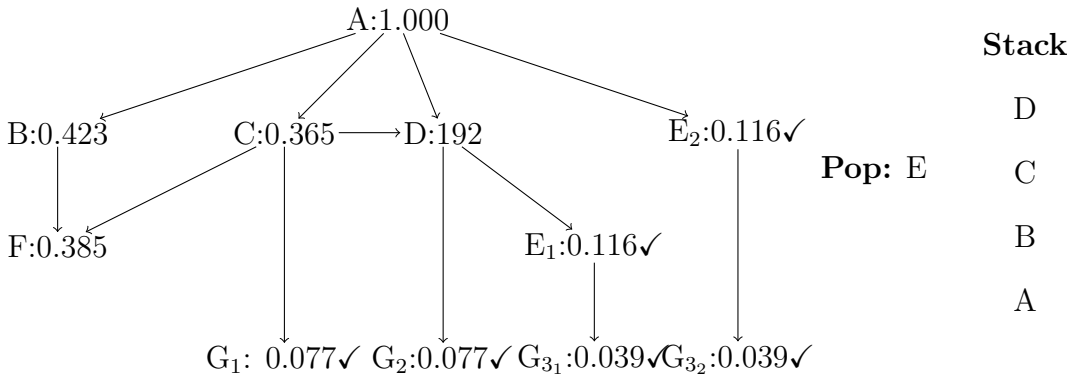
4. Vertex E is now popped from the stack. Since its only child (G_3) is marked, E is now split. G_3, being the child of a split vertex is pushed on to the stack.



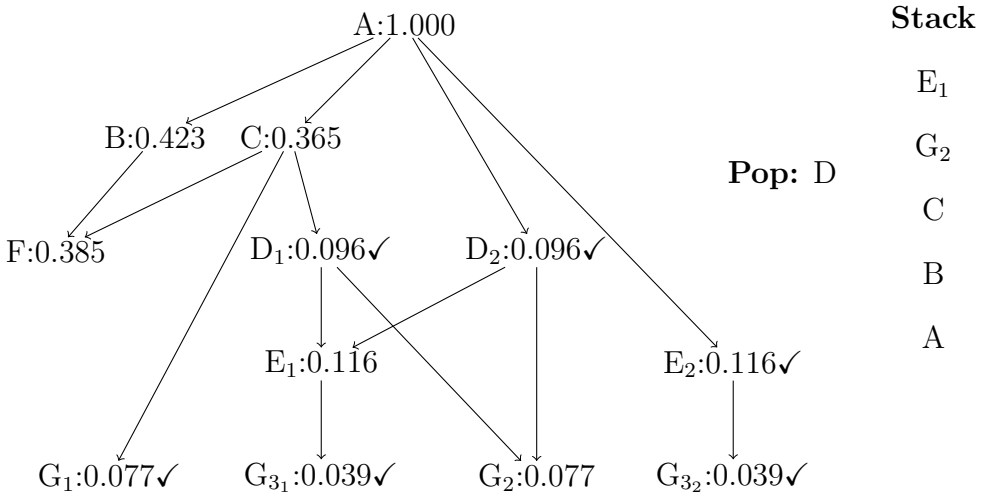
5. Being childless, G_3 is immediately split and the resulting new vertices are marked.



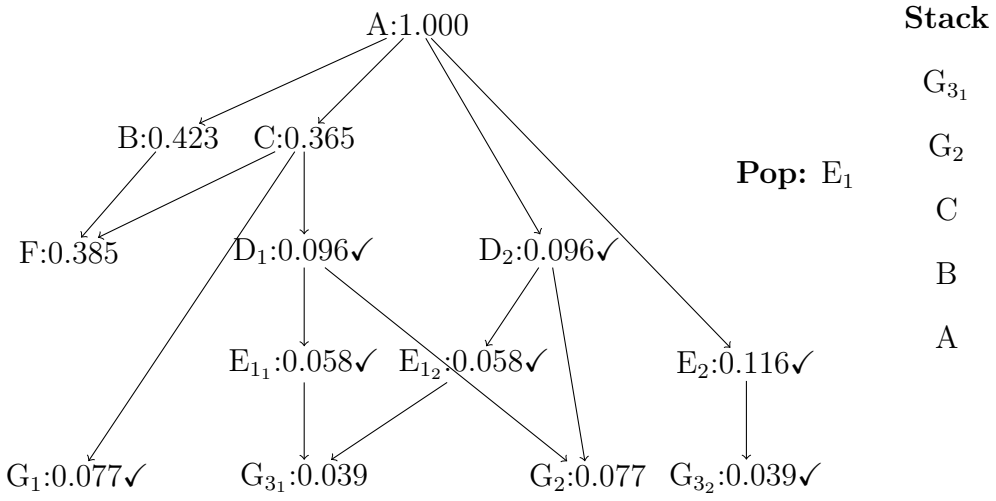
6. E is then popped from the stack. Nothing happens at this point because it no longer exists.



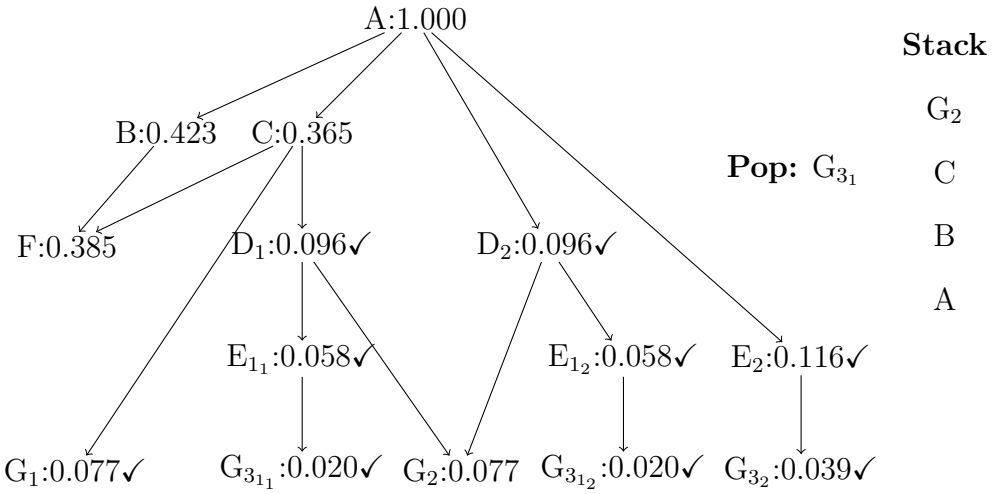
7. D is now popped and split, resulting in the marks of vertices E_1 and G_2 being cleared. As a result E_1 and G_2 are both pushed on to the stack.



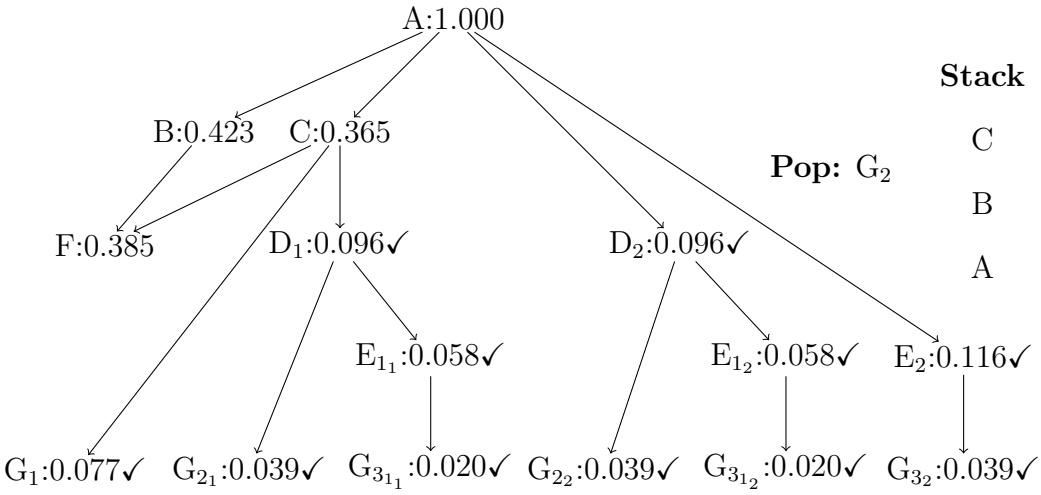
8. E_1 is now popped and split, resulting in the marks of vertices G_{3_1} and G_2 being cleared. As a result G_{3_1} and G_2 are both pushed on to the stack.



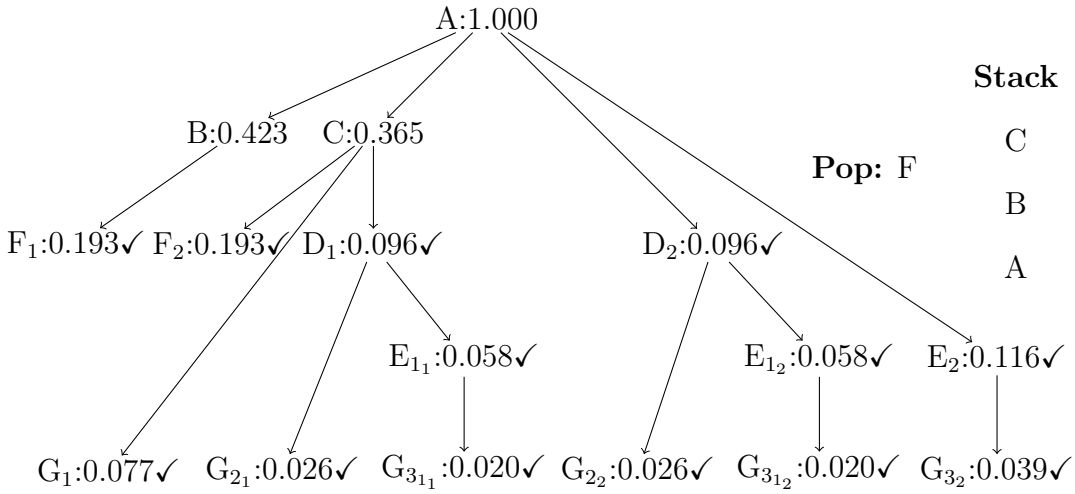
9. G_{3_1} is now popped off of the stack and split.



10. G_2 is now popped off of the stack and split.



11. The next diagram shows that when C is popped off of the stack and evaluated the algorithm finds that one of its children, F, is not marked. As a result C is pushed back on to the stack, followed by F. When F is popped it is split.

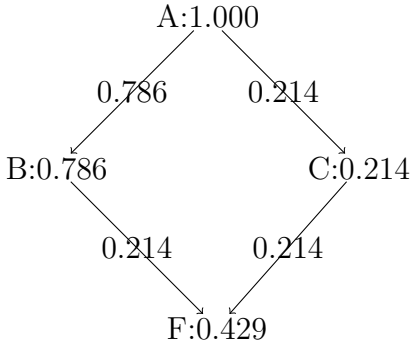


12. The algorithm now pops the rest of the stack and, finding no condition to push anything back on to the stack, finishes.

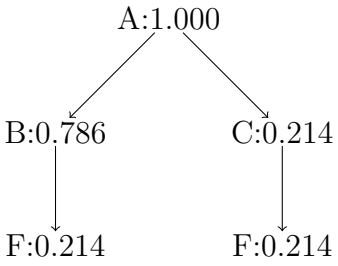
For each of the leaf nodes a path is traced from the leaf to the root node. This path forms the abstraction path. The abstraction paths produced for this DT are:

abstraction paths	weight
A, B, F	0.193
A, B	0.423
A, C, F	0.193
A, C, G	0.077
A, C, D, G	0.026
A, C, D, E, G	0.020
A, C, D, E	0.058
A, C, D	0.096
A, C	0.365
A, D, G	0.026
A, D, E, G	0.020
A, D, E	0.058
A, D	0.096
A, E, G	0.039
A, E	0.116
A	1.000

The situation is much simpler for document 2:



...which becomes the DT:



The abstraction paths produced for this DT are:

itemsets	support
A, B, F	0.214
A, C, F	0.214
A, B	0.786
A	1.000

The reason path A, B is included and path A, C is not is that there is no change in the weight values between A, C, F and A, C. Therefore, no new information is gained by including the abstraction path for A, C in the set of abstraction paths for document 2.

Document Inverted Index

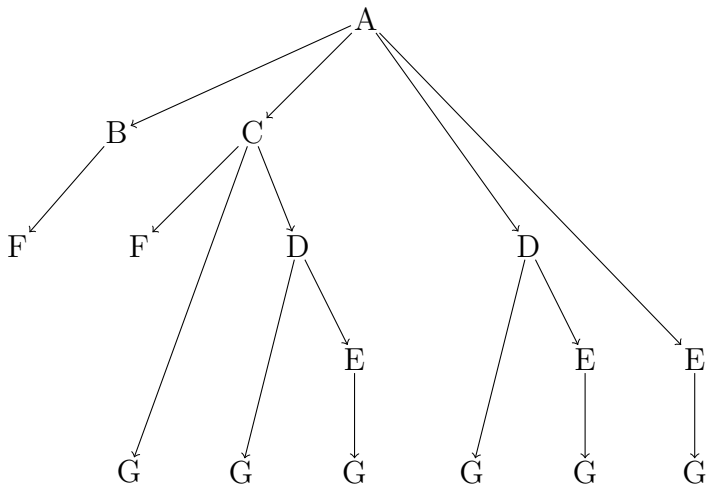
To create the document inverted index, which is a primary index and is not used for information retrieval, each of these abstraction paths is related back to the document from which it came. Using this method we obtain abstraction paths for each synset that is represented in each document and a reference from the abstractions back to the document in which it occurs. The uniqueness of these abstraction paths in their database structures is maintained through the use of a standard hashing function (algorithm 9). The order of the paths is enforced by the hypernym order in WordNet.

1. Extract a tree from the DG:

Algorithm 9: Compute Abstraction Path Hash Code

input : an abstraction path
output: a hash code

```
1 begin
2 |   let hash = 1
3 |   let synsetSum = 0
4 |   forall synset offset ∈ abstraction path do
5 |   |   let synsetSum = synsetSum + synset offset
6 |   let hash = hash × 31 + synsetSum
7 |   let hash = hash × 31 + end synset offset
8 end
```



The abstraction paths created by this DT are:

A, B, F

A, C, F

A, C, G

A, C, D, G

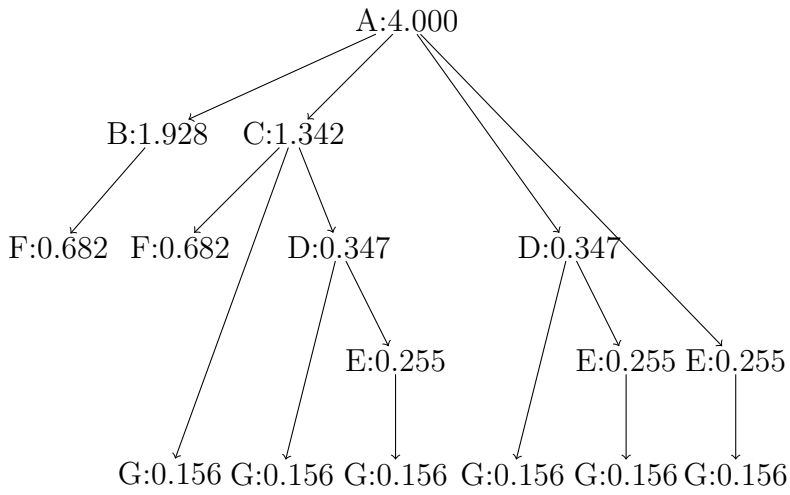
A, C, D, E, G

A, D, G

A, D, E, G

A, E, G

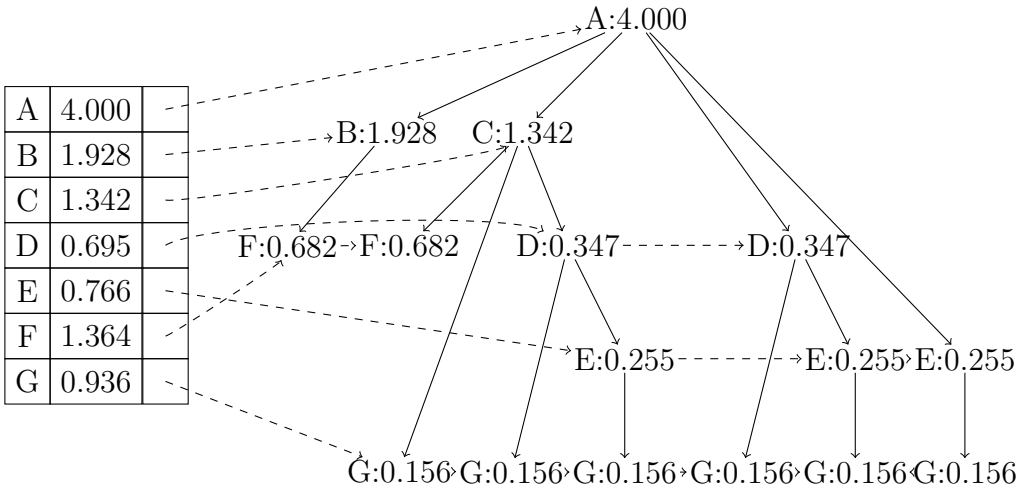
2. Assign to each node in the tree the weight corresponding to it's node sum divided by the cardinality of that node's label:



The weighted abstraction paths now become:

Path	support
A, B, F	0.682
A, C, F	0.682
A, C, G	0.156
A, C, D, G	0.156
A, C, D, E, G	0.156
A, D, G	0.156
A, D, E, G	0.156
A, E, G	0.156

3. Create a master document tree:



Mining the Master Document Tree

Minimum Support: In frequent pattern mining, minimum support is a necessary dimension to which attention must be paid. The significance of minimum support in frequent pattern mining is that it is the threshold above which items are considered frequent. In this case, minimum support corresponds to the degree of scrutiny that is applied to the admission of a path to frequent status, and hence that path's inclusion in the inverted index. An adjustment of this value upwards, for example, would have the effect of shrinking the inverted index.

The MDTis mined using the procedure detailed in algorithm 10.

The changes to the original FP-Growth algorithm are all made so that the result achieved from mining the MDT is a set of frequent abstraction paths that are *connected* with respect to the WordNet ontology. The steps taken to this end include:

1. only generating prefix path trees for the header table entries corresponding to vertices that have an outdegree of 0. This is an effective pruning step because, it can be claimed, the vertices in a conditional tree that have an outdegree of

Algorithm 10: Mining of a master document tree

input : An FP-Tree Ξ, α
output: $[\phi \mid \Phi]$ where ϕ is a frequent pattern in the set of frequent patterns Φ

```
1  begin
2      let Ψ equal a stack of prefix path trees
3      forall header table entries r in header table R do
4          generate a prefix path tree for r
5          push the prefix path tree on to Ψ
6      while Ψ is not empty do
7          pop Ξ_r off of Ψ
8          if Ξ_r contains a single path P then
9              if path P is connected then
10                 generate pattern β ∪ α with
                   support_count = minimum support count of nodes inβ
11         else
12             if path α_i ∪ α is connected then
13                 generate pattern β = α_i ∪ α with
                   support_count = a_i.support_count
14                 convert Ξ_r into its conditional FP-Tree
15                 forall λ in Λ with an outdegree of 0 do
16                     generate a prefix path tree corresponding to the header table
                       entry for λ
17                     push the prefix path tree on to Ψ
18 end
```

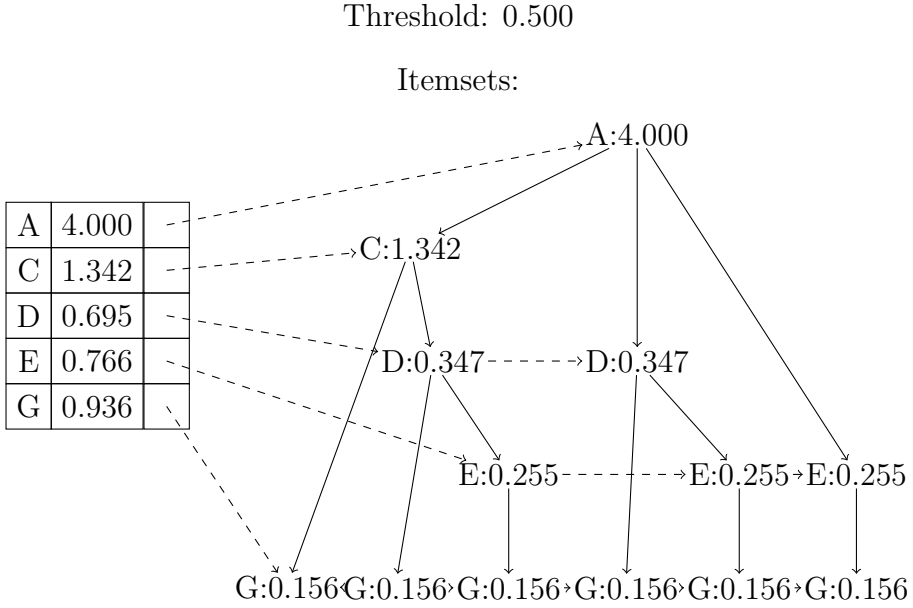
0 can be said to have direct connections in at least one instance to the items upon which the conditional tree is conditioned.

2. checking for connectivity when the frequent paths are accumulated.

Frequent path discovery for the running example are generated in the following example.

1. G:

   (a) Gather all prefix paths containing node G:

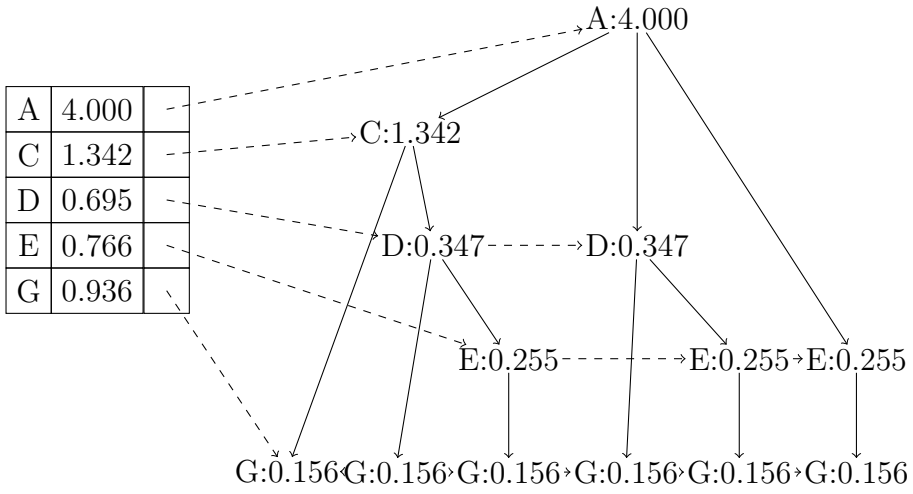


A	4.000	
C	1.342	
D	0.695	
E	0.766	
G	0.936	

   (b) Derive the support count by adding the support counts associated with nodes labeled with G. If this support count is above the minimum support threshold, declare the item G-conditionally frequent:

Threshold: 0.500

Itemsets: {G:0.936}

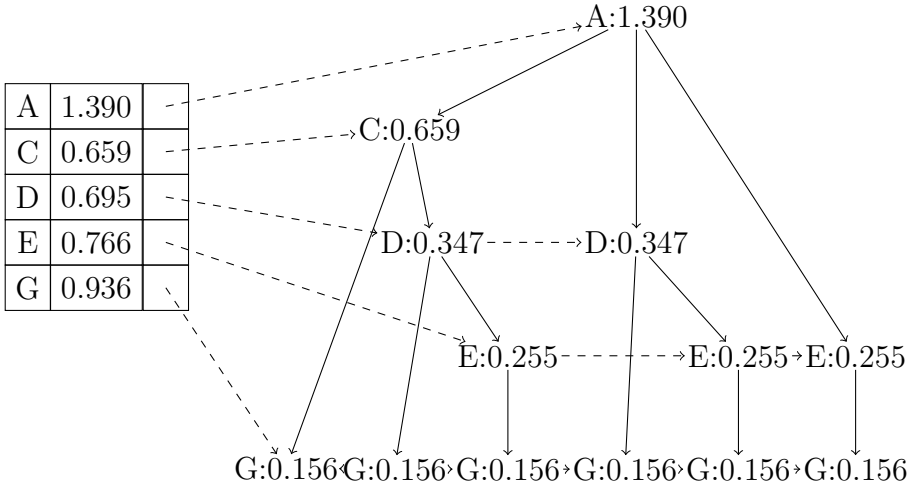


(c) Convert the prefix paths into a conditional prefix tree:

i. Update the support counts, taking into account that some of the tallied transactions do not include the item G.

Threshold: 0.500

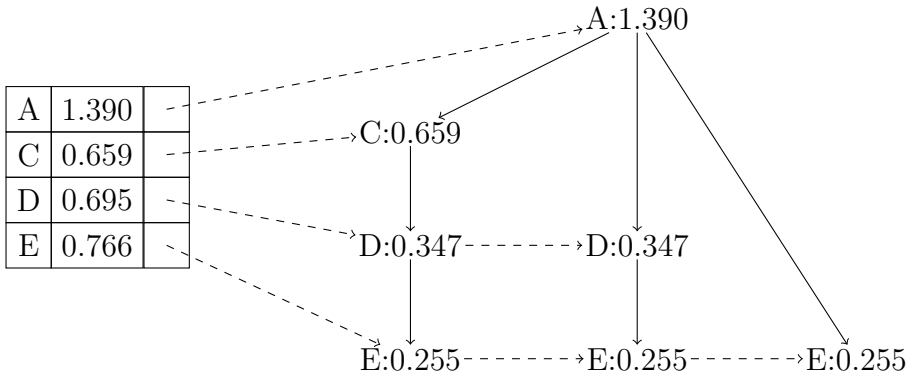
Itemsets: {G:0.936}



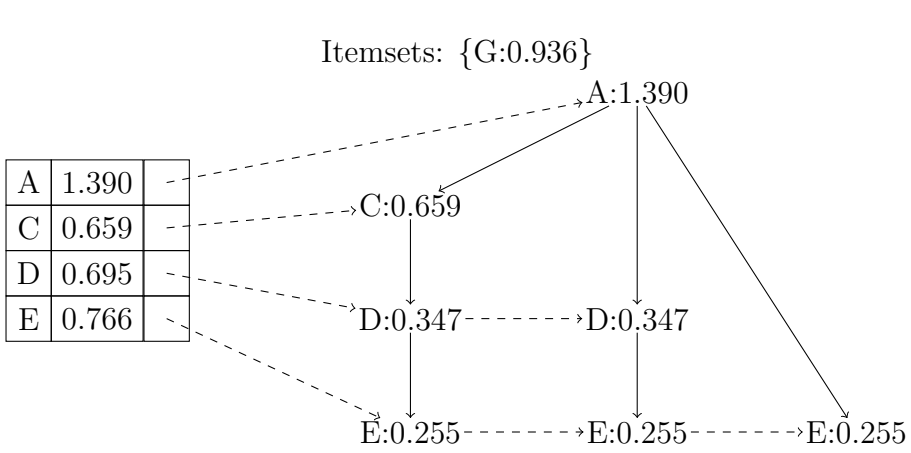
ii. Remove the nodes for G:

Threshold: 0.500

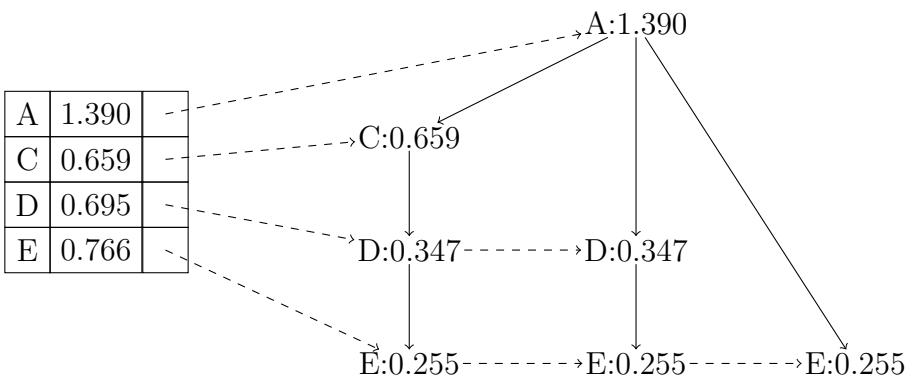
Itemsets: {G:0.936}



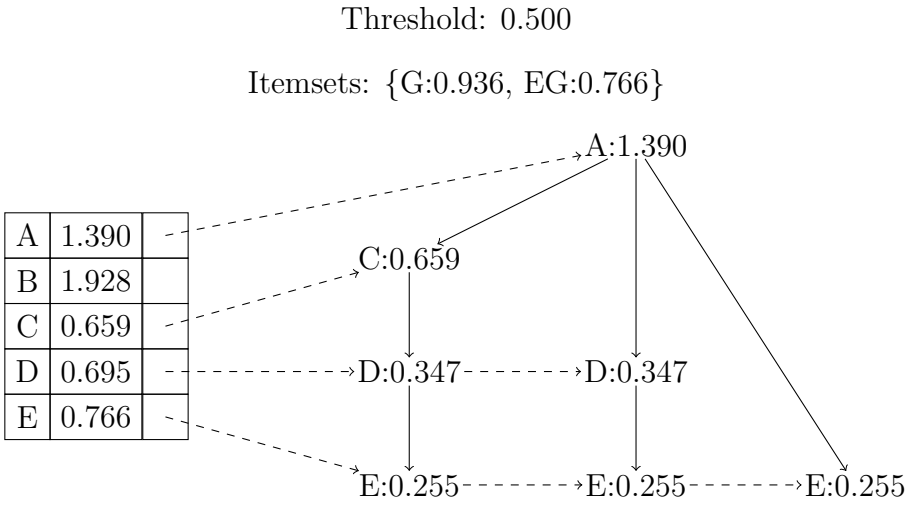
iii. Remove the items that are no longer frequent after the previous two updates:



(d) Gather all prefix paths containing nodes EG:

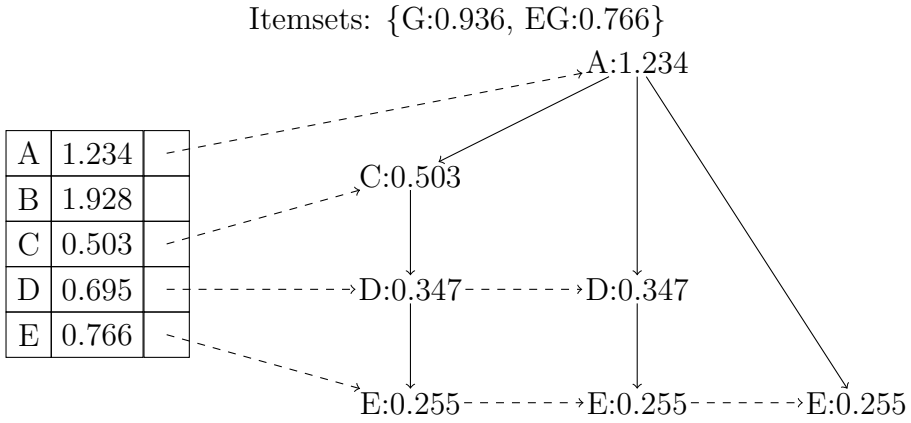


(e) Derive the support counts as before with EG:



(f) Convert the prefix paths into a conditional prefix tree:

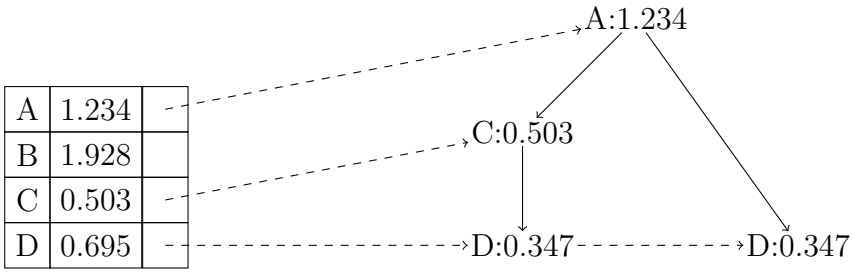
i. Update the support counts:



ii. Remove the E nodes:

Threshold: 0.500

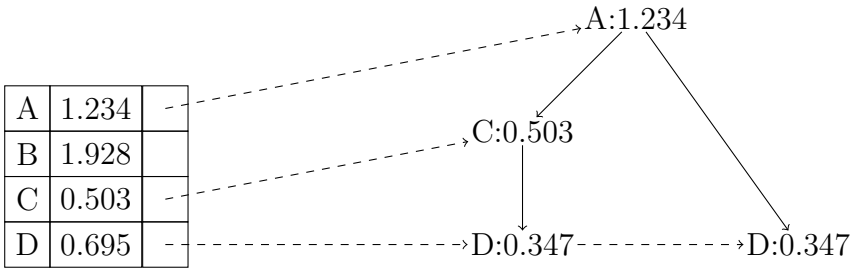
Itemsets: {G:0.936, EG:0.766}



iii. Remove items that are no longer frequent:

Threshold: 0.500

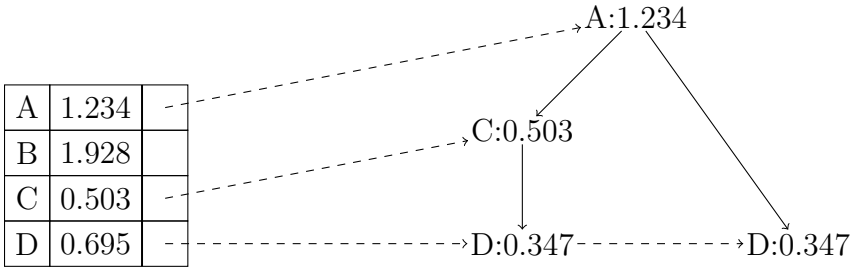
Itemsets: {G:0.936, EG:0.766}



(g) Gather all prefix paths containing nodes DEG:

Threshold: 0.500

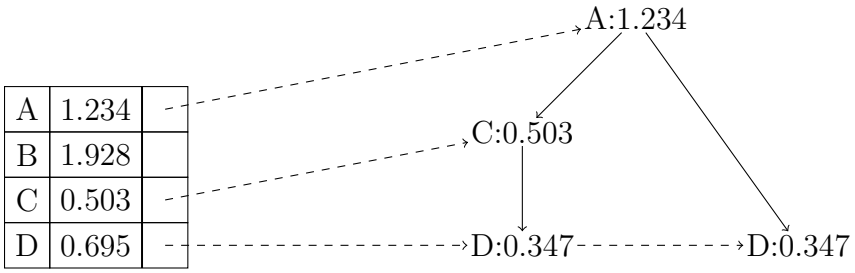
Itemsets: {G:0.936, EG:0.766}



(h) Derive the support counts as before with DEG:

Threshold: 0.500

Itemsets: {G:0.936, EG:0.766, DEG:0.695}

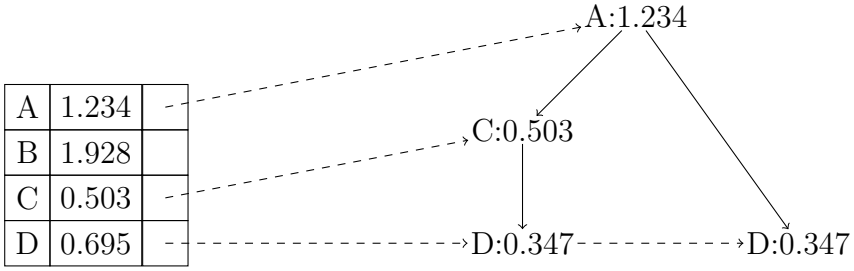


(i) Convert the prefix paths into a conditional prefix tree:

i. Update the support counts:

Threshold: 0.500

Itemsets: {G:0.936, EG:0.766, DEG:0.695}



ii. Remove the D nodes:

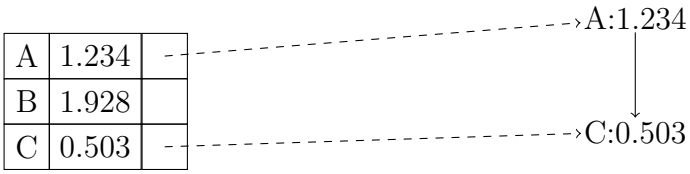
Threshold: 0.500

Itemsets: {G:0.936, EG:0.766, DEG:0.695}

iii. Remove items that are no longer frequent:

Threshold: 0.500

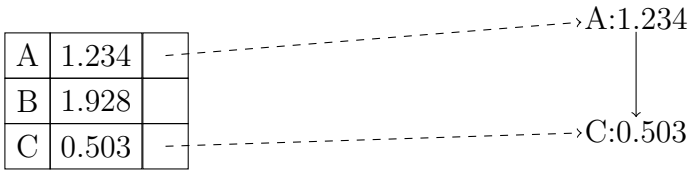
Itemsets: {G:0.936, EG:0.766, DEG:0.695}



(j) Gather all prefix paths containing nodes CDEG:

Threshold: 0.500

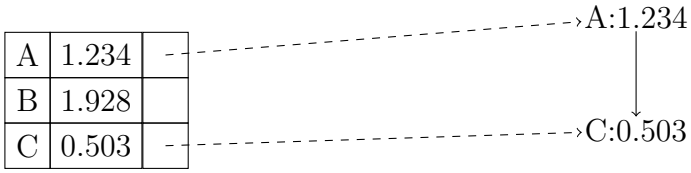
Itemsets: {G:0.936, EG:0.766, DEG:0.695}



(k) Derive the support counts as before with CDEG:

Threshold: 0.500

Itemsets: {G:0.936, EG:0.766, DEG:0.695, CDEG}

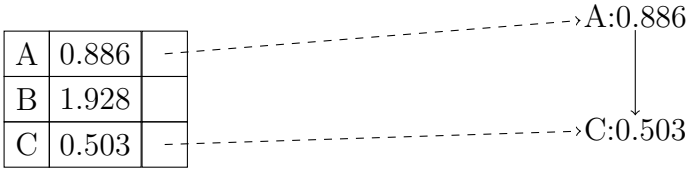


(l) Convert the prefix paths into a conditional prefix tree:

i. Update the support counts:

Threshold: 0.500

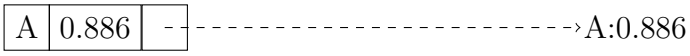
Itemsets: {G:0.936, EG:0.766, DEG:0.695, CDEG}



ii. Remove C:

Threshold: 0.500

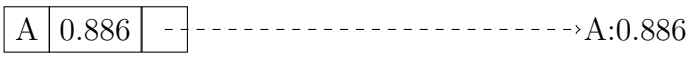
Itemsets: {G:0.936, EG:0.766, DEG:0.695, CDEG}



iii. Remove items that are no longer frequent:

Threshold: 0.500

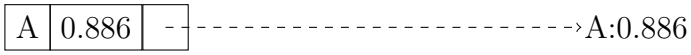
Itemsets: {G:0.936, EG:0.766, DEG:0.695, CDEG}



(m) Gather all prefix paths containing nodes ACDEG:

Threshold: 0.500

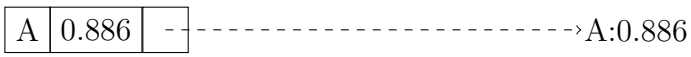
Itemsets: {G:0.936, EG:0.766, DEG:0.695, CDEG}



(n) Gather all prefix paths containing nodes ACDEG:

Threshold: 0.500

Itemsets: {G:0.936, EG:0.766, DEG:0.695, CDEG, CDEG:0.886}



Building Abstraction Path-Based Inverted Indices

With the information gathered, the construction of an inverted index proceeds in a way that has a more complicated structure than standard inverted indices, using entire abstraction paths instead of keywords. Algorithm 11 shows the pseudocode for this inverted index construction. It shows the process of taking the intersection of the frequent abstraction paths and the document abstraction paths and using the resulting abstraction paths' document abstraction path records as the inverted index.

Figure 3.6 shows the difference between the number of itemsets and the number of abstraction paths. From this data it is clear that using abstraction paths will result

Algorithm 11: Create Abstraction Path-Based Inverted Index

input : $AP_{frequent}$: the frequent abstraction paths, $AP_{document}$: the document abstraction paths, and $\Omega_{document}$: the document inverted index
output: $\Omega_{ap_{frequent}}$, an inverted index based on $AP_{frequent} \cup AP_{document}$

```
1 Let Ω_{ap_frequent} be a new inverted index
2 begin
3     forall ap_frequent ∈ AP_frequent do
4         if ap_frequent ∈ AP_document then
5             Let ω_document equal the document inverted index entry
              corresponding to ap_frequent
6             Add ω_document to Ω_{ap_frequent}
7 end
```

in a data space that has dimensionality far smaller than one based on frequent all itemsets.

Indexing System Architecture

Figure 3.7 shows the changes we have made to the elementary retrieval system architecture (see 2.1). The three main areas in which these changes were implemented were in the indexers, the free text query parser, and the inverted index.

Processing of Queries

Upon acceptance of a plain-text query, several actions proceed in sequence:

1. The query undergoes the same preprocessing as the documents.
   (a) A query graph and query tree are constructed.

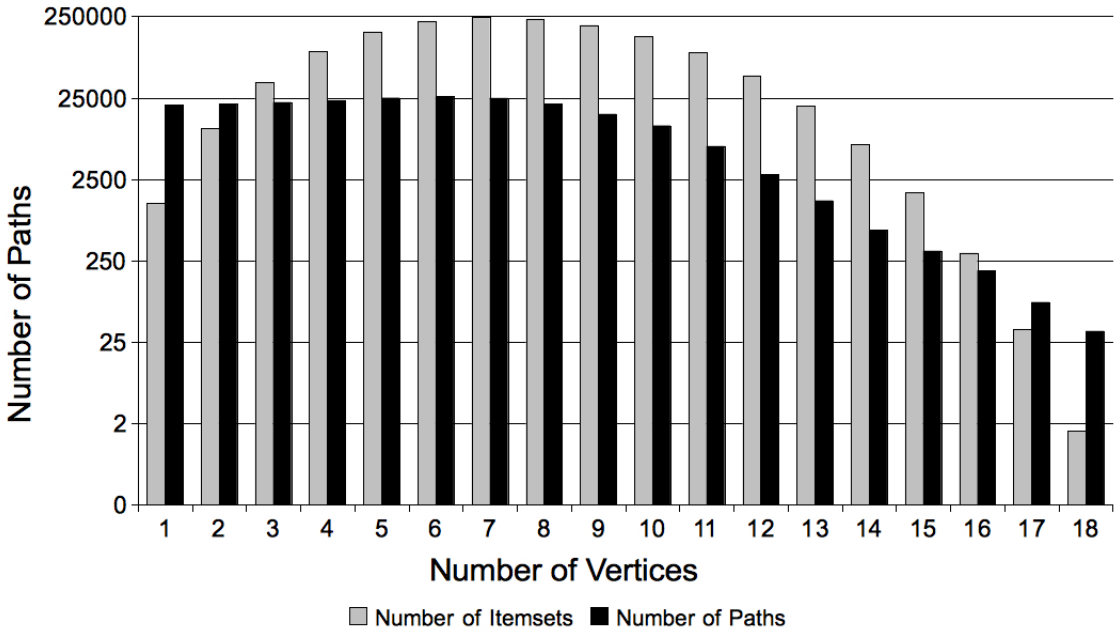
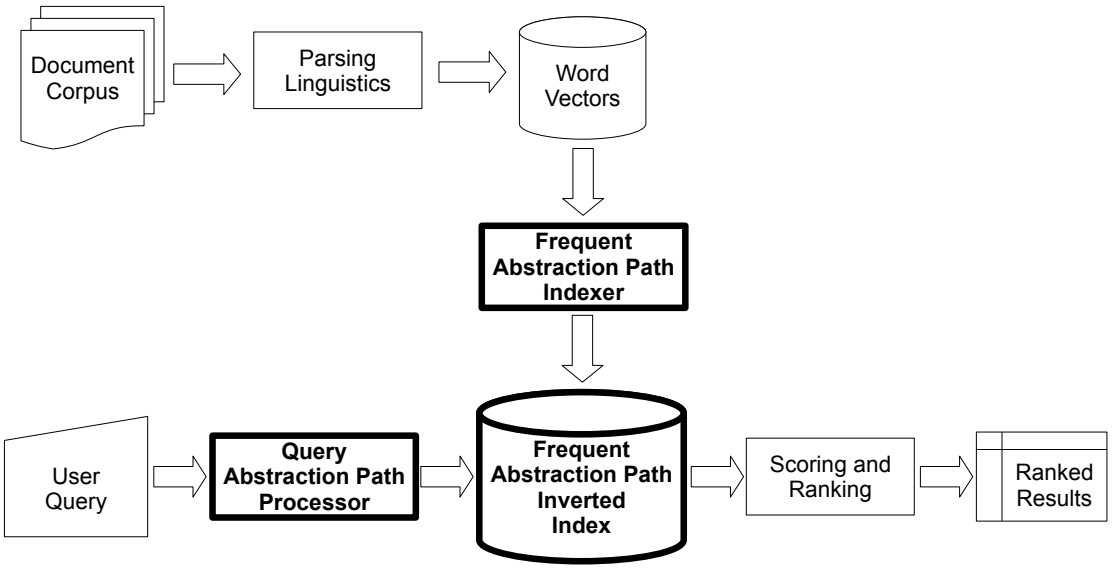


Figure 3.6: Numbers of Paths and Numbers of Itemsets for 20,000 Document Set for Support = 0.01 and Number of Original Keywords = 25083

Figure 3.7: Changes to the Elementary Information Retrieval System Architecture



(b) Query abstraction paths are harvested.

2. Selection from of relevant documents is performed using the inverted index and the query abstraction paths.

3. Ranking of the documents, selected above, is performed using the cosine similarity measure.

The Query Tree and Query Abstraction Paths: A *query tree* is generated in the same way as a document tree. It allows us to perform standard comparisons between queries and documents as if we were comparing only documents. To facilitate the search through the collection of abstraction paths that came from the document analysis, the same structures are obtained from the query tree.

Selection of Related Documents from the Inverted Index: The hash codes of the query's abstraction paths are used to query the inverted index for relevant postings. Postings relevant to any of the query's abstraction paths are considered as relevant documents.

Ranking: The ranking procedure uses the cosine similarity measure [3]. Each document in the result set, generated above, is compared against the query using this measure. The results are then sorted by descending order of cosine similarity and the results are presented to the user. Equation 3.2 depicts the calculation of the cosine similarity between the query and document d_1 in figure 3.8. For this figure, the cosine similarity will be calculated between the query and all three documents. This will dictate the order of the results.

$$sim(query, d_1) = \frac{\vec{V}(query) \bullet \vec{V}(d_1)}{|\vec{V}(query)||\vec{V}(d_1)|} \tag{3.2}$$

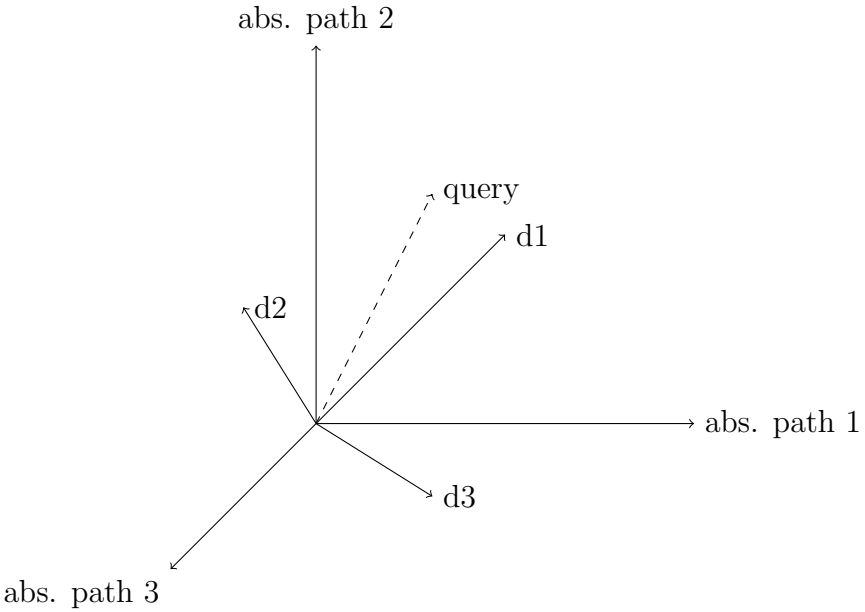


Figure 3.8: Word Vector Graphical Representation

EXPERIMENTAL EVALUATION

For our experimentation we used the 20 Newsgroups dataset [22]. This dataset contains almost 20,000 newsgroup documents from 20 different newsgroups, about 1000 per newsgroup. The newsgroups included are:

comp.graphics	sci.electronics
comp.os.ms-windows.misc	sci.med
comp.sys.ibm.pc.hardware	sci.space
comp.sys.mac.hardware	misc.forsale
comp.windows.x	talk.politics.misc
rec.autos	talk.politics.guns
rec.motorcycles	talk.politics.mideast
rec.sport.baseball	talk.religion.misc
rec.sport.hockey	alt.atheism
sci.crypt	soc.religion.christian

Table 4.1: All Newsgroups Included in the 20 Newsgroups Dataset

A template for our experimentation was taken from the work of Ido Cohn and Amit Gruber [23]. Following these authors, we used a subset of the 20 Newsgroups dataset. The purpose of this truncation was to pick the newsgroups that were most distinct from each other:

comp.sys.mac.hardware	sci.med
rec.autos	sci.space
rec.sport.hockey	misc.forsale
sci.crypt	talk.politics.guns
sci.electronics	alt.atheism

Table 4.2: 10 Newsgroups Used for the Investigations

Each newsgroup in this subset has all of its original 1000 documents, making a total of 9580 documents for the entire subset when duplicates and documents that contain only header information are removed. The newsgroup subset was selected for

their mutually distinctive subject character according to Slonim and Tishby [24]. The queries we used were the queries published by Cohn and Gruber [23] in their survey of information retrieval techniques. Table 4.3 is a list of these queries along with each query's corresponding target newsgroup.

Table 4.3: Queries Used for the Investigations

	Query	**Target Group**
1	device board video signal window RGB driver machine cache software processor scanner scsi port powerbook cable apple mac modem laptop client hardware memory buffer chip chips simm pcb dram monitor drive disk vram ram	comp.sys.mac.hardware
2	auto automobile engine wheel steering steer trunk truck speed drive road tire crash front speedometer gear shift brake brakes clutch stereo window condition door model engines flat ticket highway cop tires chassis steering liter liters throttle clutch automatic manual	rec.autos
3	league hockey player trophy ice skate goal keeper mask puck trade goal goals offense offensive defense defensive team score speed injury zone assist assists rookie rookies game playoff playoffs pass coach crowd goaltender goaltenders draft center	rec.sport.hockey

Continued on Next Page...

Table 4.3 – Continued

	Query	Target Group
4	cryptography algebra boolean group code password hack hacker crack encrypt encrypted encryption public safety decode decoding military espionage security nsa crypto cryptosystem cryptology clipper chip device transmit protocol key value secret sequence agent bit bits function hash algorithm protect	sci.crypt
5	antenna circuit design component components electronic electric electronics video audio intel memory output pcb net component filter chip mixer mixers current socket tape wire wires analog digital signal	sci.electronics
6	psychology fatal adult child patient doctor pain operation paralyze blood pressure antibiotics clinic clinical advice advise paralyze paralyzed medical oral condition symptom symptoms health healthy disorder surgery medicine drug syndrome cure phobia hospital hospitals allergy trauma mental test sleep	sci.med

Continued on Next Page...

Table 4.3 – Continued

	Query	Target Group
7	space planets planet shuttle mission star astronaut ship rocket speed light gravity satellite earth orbit nasa mars saturn venus moon world pressure thrust lunar probe spacecraft radio launch system life support navigate project missle missles temperature atmosphere earth	sci.space
8	trade internet web complete buying buy wanted afford stereo information includes sell sale sales benefit price need needed call mail bought refund advise lease available interested quality shipping include including model useful pricing	misc.forsale
9	revolver crime crimes illegal amendment gun guns sniper snipers fire firearm firearms regulation regulations regulate regulated firepower weapon weapons violence possession law laws handgun handguns pistol pistols kill killed posses	talk.politics.guns
10	religious flyer spirituality separation church preacher atheism atheist christian christians bible saint faith religion christ god secular muslim islam fanatic fanatics theist theists deity	alt.atheism

Each of the queries was run against the entire set of 10,000 documents (see table 4). If a document from a particular target group was returned in response to the related query (see table 4.3), it was interpreted as a match. If it was a document from any other group (e.g. non-target groups), then we interpretd such a response as non-matching. This experimental scenario follows that of Cohn and Gruber [23]. In keeping with this scenario, we also used the measures of *precision* and *recall*. Like the Cohn and Gruber experiments our results were truncated to the top 500 documents.

Investigations

The results from our procedure are influenced by several parameters that can be manipulated to adjust the efficacy of our approach. These are:

1. *Path length lower limit*: The minimal length of a path (i.e. the distance from the most abstract level to the terminating vertex). For example, a *path length lower limit* of 5 would only allow abstraction paths with a length of 5 and higher to be included in the inverted index.

2. *Path length upper limit*: The maximal length of the paths to be included in the inverted index. By constraining the length of the paths to shorter ones the inverted index will include only the abstraction paths that correspond to the more abstract words in the WordNet hierarchy.

3. *Path length range*: Specifies both minimal and maximal limits for path lengths included in the inverted index.

4. *Path popularity lower limit*: The minimal number of documents in which an abstraction path must be found in order to be included in the inverted index.

5. *Path popularity upper limit*: The maximal number of documents in which an abstraction path can be found in order to be included in the inverted index.

6. *Path popularity range*: Specifies both minimal and maximal limits on the number of documents in which an abstraction path can be found in order to be included in the inverted index.

We ran several experiments using several different scenarios. Figures 4.1 through 4.5 depict the results of the best of these configurations with respect to precision and recall. Each point represents the precision (figures 4.1 and 4.4) and recall (figures 4.2 and 4.5) for a specific number of documents retrieved. Specifically, using a path length ranges of 1 through 9, 9 through 18, 0 through 18, and using single path lengths of 8 and 12. Also using the path popularities of 10 through 100, 100 through 1000, 1000 through 2000, 1 through 5000, 7000 through 9000, and 9000 through 10,000.

Figure 4.1 depicts the precision at several document retrieval intervals for the path length investigations. The chart shows, for example, at a level of 10 documents retrieved the path length configuration of <12, 12> (only abstraction paths 12 vertices long were included in the inverted index), the precision achieved was 0.77. At a level of 20 documents the same configuration produced a winning precision of about 0.79. Included in this chart are depictions of a sample of the path length configurations that represent a variety of abstraction levels. Since an abstraction path length is always measured from the root of the WordNet hierarchy (i.e. the word "entity"), a longer abstraction path represents words of greater specificity and, conversely, a shorter abstraction path represents words of greater generality. As can be seen, the use of longer path lengths in the inverted index, corresponding to paths of greater specificity, resulted in better precision levels than those of low specificity (see plot <9, 18> vs. plot <0, 9> in figure 4.1). The best results for the path length investigations, and

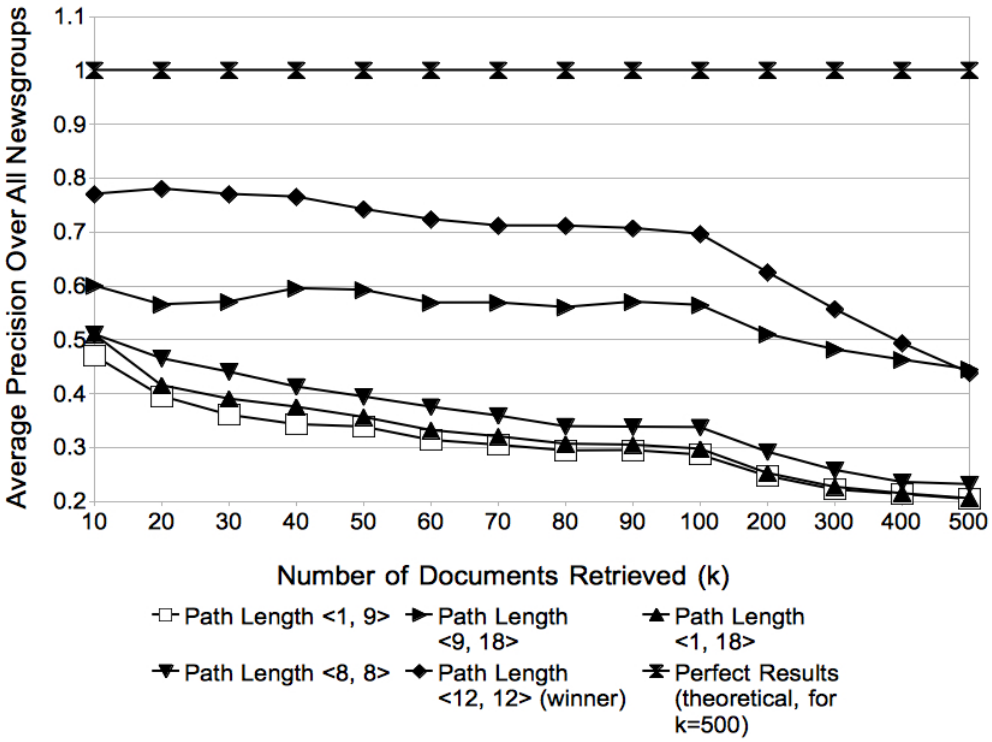


Figure 4.1: Path Length Investigation Results: Precision at k

indeed the entire set of all investigations, were achieved using a path length of 12 for all paths. For $k = 10$ the precision for this path length being 0.77 means that about 77% of all documents returned at that level were relevant.

The corresponding recall levels for the path length investigations are shown in figure 4.2. In the figure, the plot for the path length of <12, 12> shows a recall level of less than 0.01 for $k = 10$. This means that when the first 10 documents retrieved are evaluated, the search obtained greater than 9% of all relevant documents in the corpus.

Figure 4.3 shows the sizes of the inverted indices for each of the path length investigation configurations using a logarithmic scale. As can be seen, the average number of abstraction paths for each individual document profile in the inverted

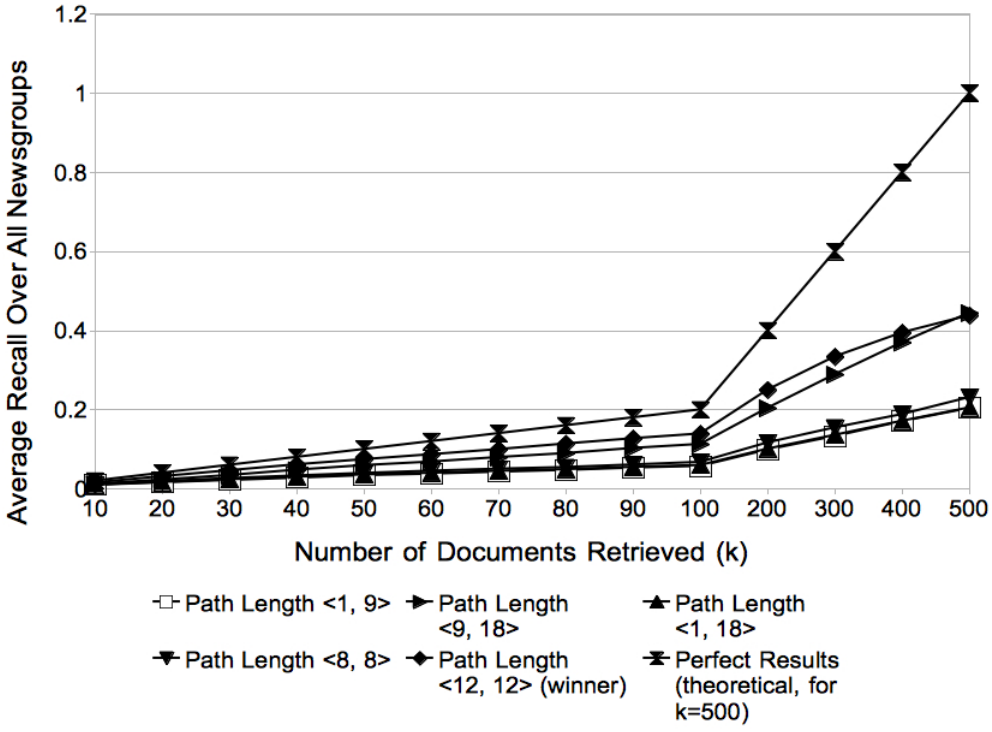


Figure 4.2: Path Length Investigation Results: Recall at k

index is the smallest for the path length <12, 12> configuration at fewer than 10 abstraction paths per document profile. Also, in this configuration there were only about 100000 total records in the inverted index.

Figure 4.4 depicts the precision at several document retrieval intervals for the path popularity investigations. These investigations represented an attempt to reduce the length of the inverted index by imposing a limit on how many documents in which any abstraction path may be present. The idea that generated this line of reasoning is that using words that are less common to the entire corpus may expose deviations in the lexica of the various groups therein. As can be seen by comparison to figure 4.1 the results did not appear as favorable as the results that were obtained using the path length limitations, though the dynamic property of the average precision with

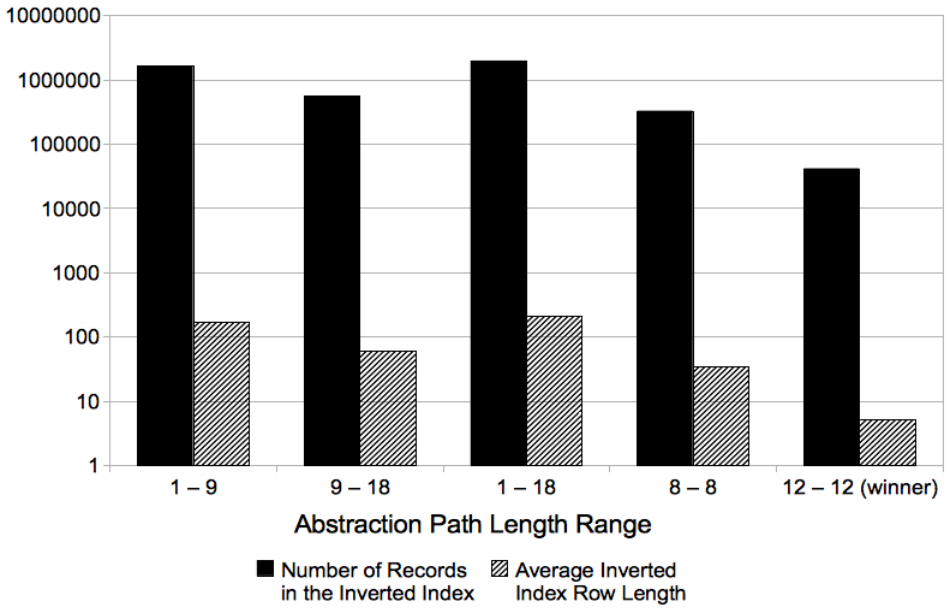


Figure 4.3: Number of Abstraction Paths in the Inverted Index and the Average Inverted Index Row Length for Path Length Investigations

respect to path popularity was clearly demonstrated. That we discovered the greatest precision level for $k = 10$ at a document popularity of <9000, 10000> is interesting because at this level of popularity the abstraction paths contained in the inverted index appear in nearly all of the documents. The reason for this remains undiscovered, but one may surmise that it is because at this popularity level, there is more of a chance for the cosine similarity to calculate relevant similarities since the values for most abstraction paths in the inverted index are non-zero. The corresponding recall levels for the path popularity investigations are shown in figure 4.5.

Figure 4.6 shows the sizes of the inverted indices for each of the path popularity investigation configurations using the same logarithmic scale as was used for the chart showing the same for the path length experiments. As can be seen, the average number of abstraction paths for each individual document profile in the inverted index is the smallest for the path popularity <9000, 10000> configuration at little

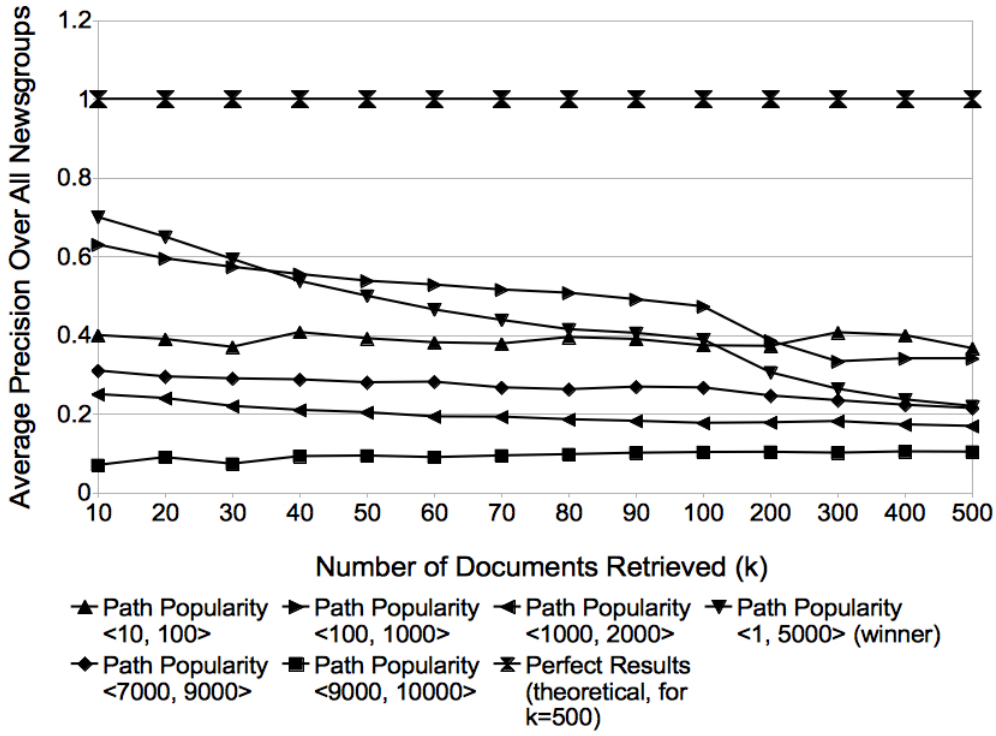


Figure 4.4: Path Popularity Investigation Results: Precision at k

more than 10 abstraction paths per document profile. Also, in this configuration there were only about 100,000 total records in the entire inverted index.

There was a dissimilarity in the results of the investigations with respect to the target newsgroup that deserves mention. Some groups, such as sci.med fared consistently well, sometimes achieving a precision of 1.0 for 10 documents retrieved. For other groups, such as misc.forsale, the precision was never greater than 0.4 for any retrieval level. Figure 4.7 shows the best and the worst of these differences. As can be clearly seen, sci.med produced results whose worst results were better than the best results achieved for the misc.forsale group. One may surmise that this is because of the greater variability exhibited by the lexicon characterizing the misc.forsale group that that exhibited by the sci.med group.

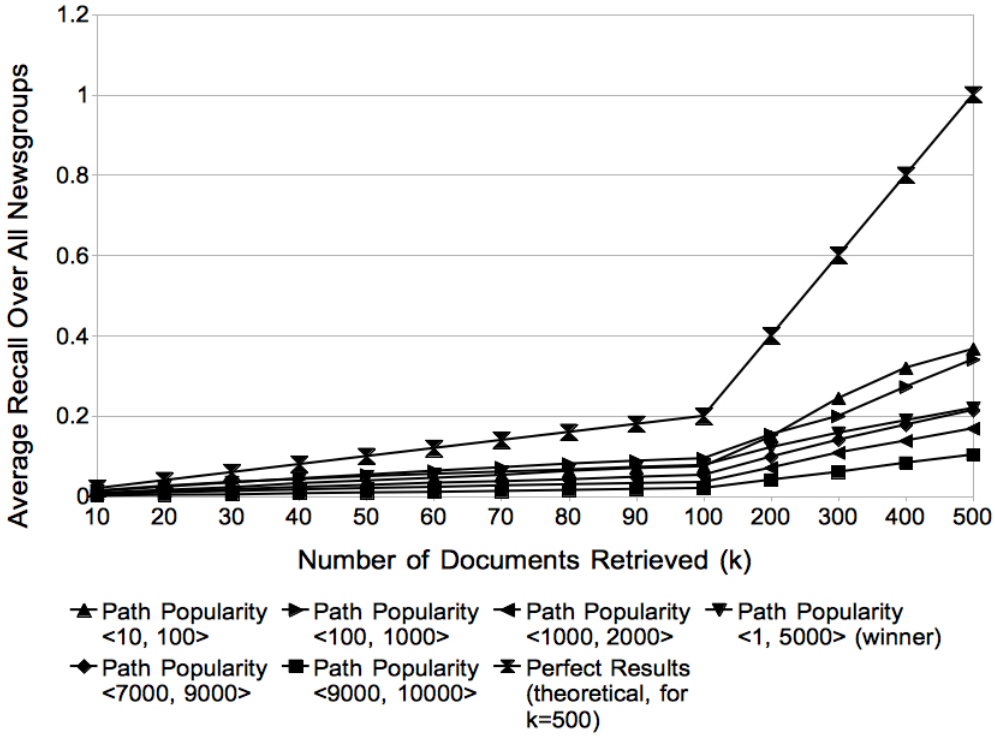


Figure 4.5: Path Popularity Investigation Results: Recall at k

Figures 4.8 and 4.9 depict precision vsrecall for the path length and path popularity investigations depicted above (FAP), along with an approximation of Cohn and Gruber's TFIDF results [23]. In both cases, though our results were not as good as the results achieved using the TFIDF inverted index, we did achieve results that show that the use of abstraction path based inverted indices for information retrieval is a viable alternative to keyword indexing, producing results that approach precision and recall figures for TFIDF indexing. Furthermore, we have identified many parameters that influence the results of our experiments. The viability of our methodology will depend on the identification of optimal values for these parameters.

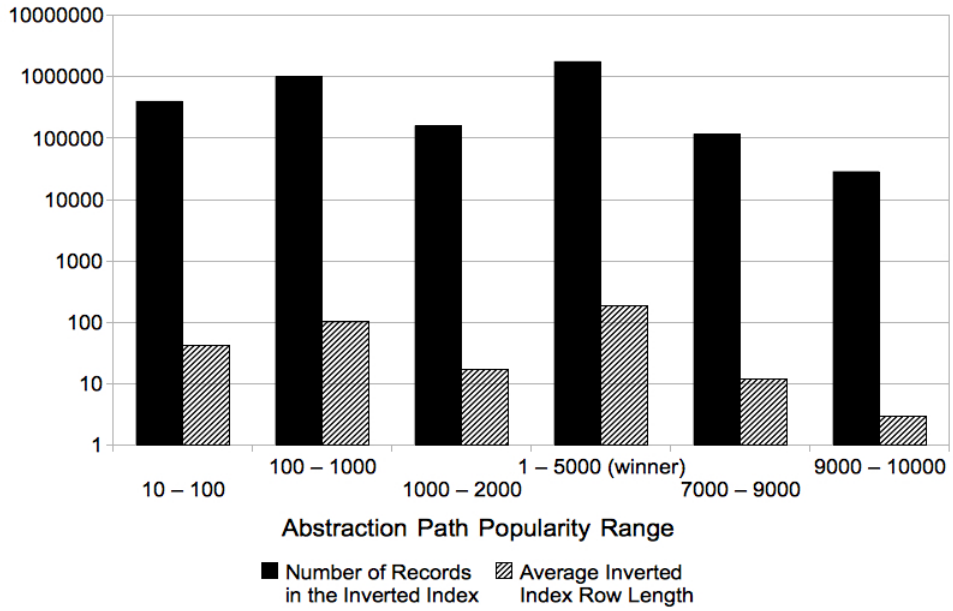


Figure 4.6: Number of Abstraction Paths in the Inverted Index and the Average Inverted Index Row Length for Path Popularity Investigations

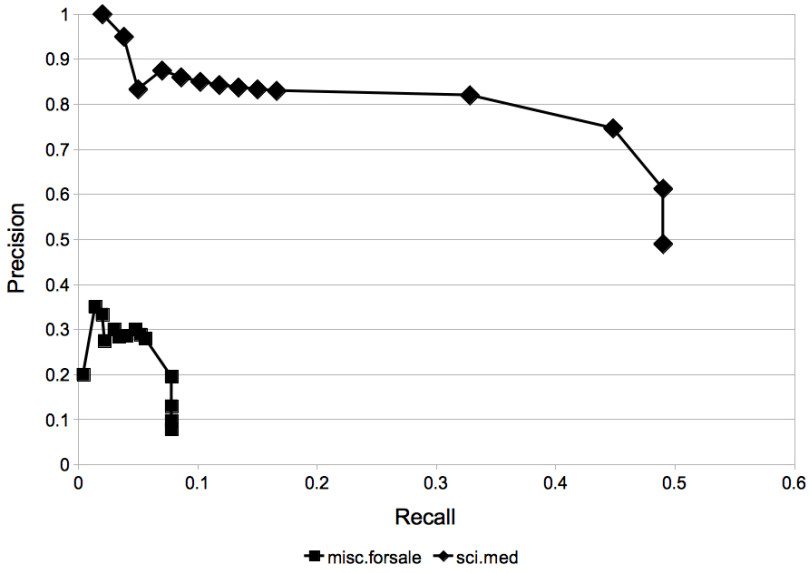


Figure 4.7: Precision vs. Recall: misc.forsale (the worst results) and sci.med (the best results)

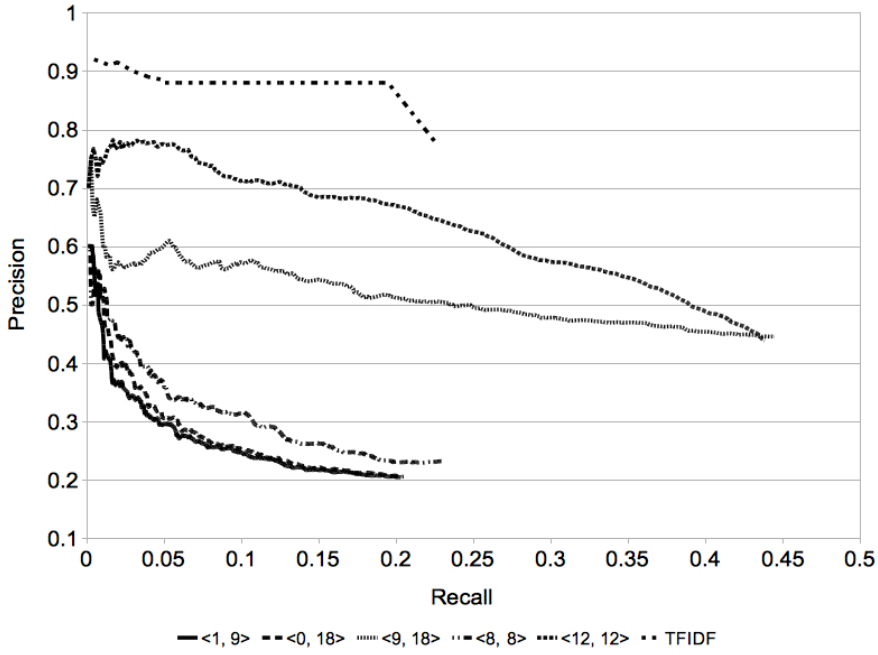


Figure 4.8: Precision vs. Recall: Path Length Investigations

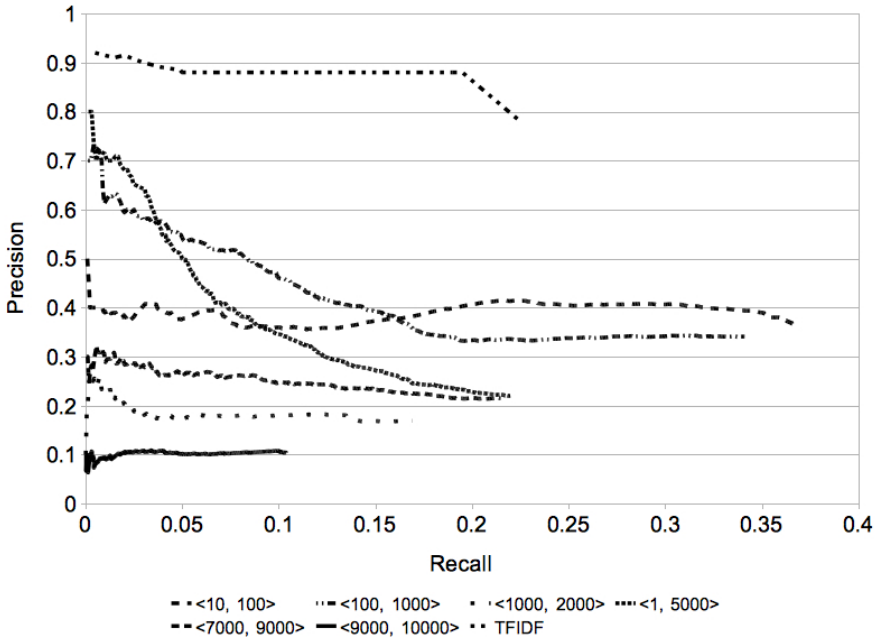


Figure 4.9: Precision vs. Recall: Path Popularity Investigations

CONCLUSIONS AND FUTURE WORK

In this paper we present an implementation of an information retrieval system that discovers/creates and uses FP-Tree motivated, ontologically-enhanced data representations (i.e. frequent abstraction paths) instead of the commonly used "bag of words" model. The major contribution of this thesis is the development of a system that quickly discovers frequent abstraction paths and uses them to perform information retrieval on using a finite corpus. As a demonstration of the effectiveness of our system we performed information retrieval experiments on 10 newsgroup subset of the 20 Newsgroups dataset. We are pleased with the operation of this system and intend to build upon it to enhance its effectiveness. The characterization of documents via frequent abstraction paths and the creation of a retrieval mechanism that uses them contribute a lot to our knowledge in this domain. Although, similarly to many information retrieval systems, our results were not perfect, we did achieve promising results and successfully completed a solid computer system for further investigation based on this reasoning.

Future Work

In all of the approaches that use frequent items to characterize data the problem is the massive number of discovered frequent itemsets that leads to high dimensionality of the data space, which in our case also transfers to the length of the inverted index that is generated by using the frequent abstraction paths. In our preliminary investigation we tried to limit the impact of both issues by picking abstraction paths within certain ranges of popularity among the characterized documents or within certain length ranges. There are far more complex (and accurate) methodologies

to reduce data dimensionality (e.g. latent semantic analysis, principal component analysis, etc.) and we plan to investigate their applicability in the future.

In the future, we intend to investigate the ramifications of each of the parametric considerations in all of their permutations to see how each parameter affects the efficacy of the system. The parameters we have identified are those of abstraction path popularity thresholds and ranges, abstraction path length thresholds and ranges, minimum support threshold, and dimension collapse via a common hypernym substitution strategy.

Some promising results were generated by picking specific abstraction levels. We want to take advantage of these results when building a "recommender" system in an information retrieval system. For example; we can envision an application that replaces the current list of likely searches that drops from the bottom of the text boxes on search engine pages. Our version would be some form of a tree based on lexical abstractions, rather than the current list. This tree would not only be based on spelling, but on subjects that are related through the ontology.

Another avenue of future work that we wish to pursue is the creation of a framework by which other ontologies may be adapted. Since there exist many domains that have produced these ontologies, e.g. the medical industry [25], we hope to use this system to resolve disparate lexica that differ with respect to time or region. An example of this may be if someone is performing a search for "potentiometer," which also may be referred to as a "voltage divider," or a "potential divider." We hope to discover whether or not such vernaculars may be resolved through the use of these ontologies.

Lastly, an important part of this investigation was in the choosing of a suitable dataset for analysis. The 20 Newsgroups dataset was chosen for its open availability, compactness and its convenient, if fundamentally flawed, relevance judgement mech-

anism. However, a 20,000 document dataset comes short of our expectations with regard to size. In the future, we hope to move to a larger (and more expensive) dataset, such as the TREC Ad-Hoc collection [26], for which relatively exhaustive relevance judgements have been made. We were prevented from using this dataset in this phase due to cost constraints.

REFERENCES CITED

[1] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation: a frequent-pattern tree approach," *SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, June 2000. [Online]. Available: http://dx.doi.org/10.1145/335191.335372

[2] E. M. Voorhees, "Question answering in trec," in *In TREC: Experiment and Evaluation in Information Retrieval*. MIT Press, 2005.

[3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, July 2008.

[4] T. Mitchell, *Machine Learning*. McGraw-Hill Education (ISE Editions), October 1997.

[5] (2009, March) University of glasgow stop word list. [Online]. Available: http://www.dcs.gla.ac.uk/idom/ir_resources/linguistic_utils/stop_words

[6] M. Konchady, *Text Mining Application Programming (Programming Series)*. Rockland, MA, USA: Charles River Media, Inc., 2006.

[7] M. Wurst. (2009, June) The word vector tool and the rapidminer text plugin. GNU Public License. [Online]. Available: http://wvtool.sf.net

[8] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, December 2006.

[9] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. ADDISON WESLEY PUBLI, June 2006.

[10] B. C. M. Fung, K. Wang, and M. Ester, "Hierarchical document clustering using frequent itemsets," in *Proc. of the 3rd SIAM International Conference on Data Mining (SDM)*. San Francisco, CA: SIAM, May 2003, pp. 59–70.

[11] T. Pedersen, S. Banerjee, and S. Padwardhan. (2009, February) Maximizing semantic relatedness to perform word sense disambiguation. [Online]. Available: citeseer.ist.psu.edu/pedersen03maximizing.html

[12] S. Wan and R. Angryk, "Measuring semantic similarity using wordnet-based context vectors," *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, 2007.

[13] P. Resnik, "Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language," *Journal of Artificial Intelligence Research*, vol. 11, pp. 95–130, 1999.

[14] M. Sussna, "Word sense disambiguation for free-text indexing using a massive semantic network," *Proceedings of the 2nd International Conference on Information and Knowledge Management*, pp. 67–74, 1993.

[15] J. J. Jiang and D. W. Conrath, "Semantic similarity based on corpus statistics and lexical taxonomy," *Proceedings of International Conference Research on Computational Linguistics*, 1997.

[16] D. Lin, "An information-theoretic definition of similarity," *Proceedings of the 15th International Conference on Machine Learning*, pp. 296–304, 1998.

[17] D. Widdows and B. Dorow, "A graph model for unsupervised lexical acquisition," *19th International conference on Computational Linguistics*, pp. 1093–1099, 2002.

[18] M. S. Hossain and R. A. Angryk, "Gdclust: A graph-based document clustering technique." in *ICDM Workshops*. IEEE Computer Society, 2007, pp. 417–422. [Online]. Available: http://dblp.uni-trier.de/db/conf/icdm/icdmw2007.html#HossainA07

[19] M. Akbar and R. A. Angryk, "Frequent pattern-growth approach for document organization." in *ONISW*, R. Elmasri, M. Doerr, M. Brochhausen, and H. Han, Eds. ACM, 2008, pp. 77–82. [Online]. Available: http://dblp.uni-trier.de/db/conf/cikm/onisw2008.html

[20] T. Qian, H. Xiong, Y. Wang, and E. Chen, "On the strength of hyperclique patterns for text categorization," *Inf. Sci.*, vol. 177, no. 19, pp. 4040–4058, 2007.

[21] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to wordnet: An on-line lexical database*," *Int J Lexicography*, vol. 3, no. 4, pp. 235–244, January 1990. [Online]. Available: http://dx.doi.org/10.1093/ijl/3.4.235

[22] (2009, May) Home page for 20 newsgroups data set. [Online]. Available: http://people.csail.mit.edu/jrennie/20Newsgroups/

[23] I. Cohn and A. Gruber. (2009, May) Information retrieval experiments. [Online]. Available: http://www.cs.huji.ac.il/ ido_cohn

[24] N. Slonim, N. Friedman, and N. Tishby, "Unsupervised document classification using sequential information maximization," 2002. [Online]. Available: citeseer.ist.psu.edu/article/slonim02unsupervised.html

[25] (2009, August) Unified medical language system (umls). [Online]. Available: http://www.nlm.nih.gov/research/umls/

[26] E. M. Voorhees and D. K. Harman, *TREC: Experiment and Evaluation in Information Retrieval*, ser. Digital Libraries and Electronic Publishing. MIT Press, September 2005. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0262220733

[27] B. C. M. Fung, K. Wang, and M. Ester, *The Encyclopedia of Data Warehousing and Mining*. Hershey, PA: Idea Group, August 2008, ch. Hierarchical Document Clustering, pp. 970–975.

[28] R. Agrawal and T. Imielinski, "Mining association rules between sets of items in large databases," 1993, pp. 207–216.

[29] R. A. McAllister and R. A. Angryk, "An efficient abstraction-based data model for information retrieval," in *The 22nd Australasian Joint Conference on Artificial Intelligence (to appear)*, 2009.

[30] W. Li, J. Han, and J. Pei, "Cmar: Accurate and efficient classification based on multiple class-association rules," *Data Mining, IEEE International Conference on*, vol. 0, p. 369, 2001.

[31] (2009, July) Jgrapht. [Online]. Available: http://jgrapht.sourceforge.net/

[32] (2009, July) Word vector tool. [Online]. Available: http://sourceforge.net/projects/wvtool/

[33] A. K. McCallum. (1996) Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. [Online]. Available: http://www.cs.cmu.edu/ mccallum/bow

[34] P. Bednar. (2007) Jbowl. [Online]. Available: http://sourceforge.net/projects/jbowl

[35] Wordnet: a lexical database for the english language. [Online]. Available: http://wordnet.princeton.edu/

[36] Postgresql: the world's most advanced open source database. [Online]. Available: http://www.postgresql.org/

APPENDIX A

SYMBOLS AND ABBREVIATIONS

W_{stop}	stopwords
w_{stop}	stopword
W	words
w	word
W_q	words in a query
D	a document corpus
d	a document
$\|D\|$	number of documents
T	word vector files
t	word vector file
t_w	word vector entry
tf	term frequency value
idf	inverse document frequency value
$tfidf$	term frequency $\times$ inverse document frequency value
DG	the document graphs
dg	a document graph
MDG	the master document graph
s	a synset
$V_{not\ propagated}$	the non-propagated master document graph vertices
DT	a document tree
MDT	a master document tree
H	a hash table
h	a hash table entry
m	the number of vertex labels
τ	a partial tfidf table

p	a precision variable
r	a recall variable
α	the recall weight
S	a set of synsets
s	a synset
v	a vertex
e	an edge
q	a query
D_q	the documents returned from a query
Ω	an inverted index
ω	an inverted index entry
π	the postings list
Ξ	an FP-Tree
Φ	a set of frequent patterns
ϕ	a frequent pattern