THE ROLE OF SEMANTIC LOCALITY IN HIERARCHICAL DISTRIBUTED DYNAMIC INDEXING AND INFORMATION RETRIEVAL

BY

FABIEN DAVID BOUSKILA

École Nationale Supérieure des Télécommunications de Paris, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1999

Urbana, Illinois

ABSTRACT

The global growth in popularity of the World Wide Web has been enabled in part by the availability of browser-based search tools, which in turn have led to an increased demand for indexing techniques and technologies. This explosive growth is evidenced by the rapid expansion in the number and size of digital collections of documents. Simultaneously, fully automatic content-based techniques of indexing have been under development at a variety of institutions. The time is thus ripe for the development of scalable knowledge management systems capable of handling extremely large textual collections distributed across multiple repositories.

Hierarchical Distributed Dynamic Indexing (HDDI) dynamically creates a hierarchical index from distributed document collections. At each node of the hierarchy, a knowledge base is created and subtopic regions of *semantic locality* are identified. This thesis presents an overview of HDDI with a focus on the algorithm that identifies regions of semantic locality within knowledge bases at each level of the hierarchy.

To my parents, my brother Gautier, my sister Élise, Bertrand and Nathalie.

ACKNOWLEDGMENTS

I would like to thank Professor William Morton Pottenger for his proactive managing of the HDDI project, and for his open-door policy towards students. I gratefully acknowledge the assistance and contributions of the staff in the Automated Learning Group directed by Dr. Michael Welge at the National Center for Supercomputing Applications (NCSA) as well as the funding and technical oversight provided by Dr. Tilt Thompkins in the Emerging Technologies Group at NCSA. I also want to thank my thesis co-advisor Professor Wen-Mei Hwu and my academic advisor Professor William H. Sanders for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The explosive growth of digital repositories of information has been enabled by recent developments in communication and information technologies. The global Internet/World Wide Web exemplifies the rapid deployment of such technologies. Despite significant accomplishments in internetworking, however, scalable indexing techniques for distributed information lag behind the rapid growth of digital collections.

In the 21^{st} century, a significant amount of information useful in the practice of science will be available via such computer communications networks. The appearance of focussed digital libraries on the World Wide Web demonstrates the willingness of scientists and engineers to distribute detailed information beyond that traditionally available in the published literature (e.g., [1]). It is critical that new information infrastructure be developed that enables effective search in the huge volume of distributed information emerging in digital form.

Traditional methods of indexing combine multiple subject areas into a single, monolithic index. There are already enough documents on the Web that such indexing technology is often failing to perform effective search. The difficulty lies in the fact that since so many documents and subjects are being combined together, retrieving all the documents that match a particular word phrase often returns too many documents for effective search. This problem has been known for some time [2].

In order to properly address this problem, a paradigm shift is needed in the approach to indexing. First and foremost, it is clear that digital collections are now and will continue to be distributed. Our first premise is thus that *indexes* must also be distributed.

Secondly, it must be realized that the information contained in these distributed digital repositories is hierarchical in nature.[1] Traditionally, knowledge hierarchies have been created with human expertise.[2] Such an approach does not scale to the tremendous amount of emerging digital information for two reasons: as knowledge increases, new topics are emerging at a greater rate, and both this and the sheer volume of information preclude manual approaches to indexing. Our second premise is thus that distributed indexes must properly reflect the hierarchical nature of knowledge.

Thirdly, due to the vast increase in communications bandwidth and computing and online storage capabilities mentioned above, digital collections are frequently updated. This process reflects a key characteristic of 21^{st} century collections: namely, they are dynamic in nature. Our third premise is thus that any new information infrastructure must include *dynamic* indexing capabilities.

In the final analysis, these three technologies must be integrated into a cohesive whole. The goal of our research is thus to architect a knowledge management prototype based on HDDI: *Hierarchical Distributed Dynamic Indexing.*

This thesis will give a brief overview of the steps involved in HDDI, and then focus on my work in developing technology for identifying regions of semantic locality in knowledge bases. I will also use the results of our experiments to justify the need for a hierarchical classification of knowledge.

Chapter 2 provides the reader with an overview of HDDI. It describes the different steps and concepts leading to a scalable dynamic index of repositories.

Chapter 3 describes sLoc, the algorithm we developed at NCSA to identify regions of semantic locality in knowledge bases. After a review of the previous work, I explain the theory behind sLoc. Because sLoc is intended to be used for a whole set of different applications, we designed a set of measures that the user may select from to get the

[1] This point was made recently by the National Center for Supercomputing Applications (NCSA) director Larry Smarr and echoed by NSF CISE assistant director Ruzena Bajcsy at recent seminars at the University of Illinois.

[2] One popular form is the thesaurus (e.g., the National Library of Medicine's MeSH thesaurus).

clustering needed for his particular application. I then report on our experiments with these different measures and give more information on their use.

In Chapter 4, I describe the experiments we conducted in order to validate our approach with sLoc. For the particular case of a query search application, we compared sLoc to gold standard collections from the SMART project:[3] ADI, CISI and MED. This led us to some better insight into the use of the hierarchy in HDDI.

Chapter 5 states our conclusions and open issues for future work.

[3]Cornell University.

CHAPTER 2

OVERVIEW OF INFORMATION RETRIEVAL AND HDDI

This chapter will discuss the goals, concepts and strategies used in building a Hierarchical Distributed Dynamic Index. I will first give a brief overview of the information retrieval field,[1] and explain what specific applications of IR HDDI addresses. I will then describe the HDDI process step by step. Finally, we will take a look at how HDDI can be applied to knowledge management in various applications.

2.1 Overview of Information Retrieval

First of all, information retrieval (IR) is a broad term with multiple meanings. In the scope of this thesis, we shall be concerned only with *automatic information* retrieval systems. We are actually considering *automatic* as opposed to *manual* systems; we also want to make a clear distinction between *information* and *data*. In "Information Storage and Retrieval" [4] Korfhage describes the difference between information and data as follows:[2]

> *Data* are received, stored, and retrieved by an information endosystem. The data are impersonal; they are equally available to any users of the system. *Information*, in contrast, is a set of data that have been matched to a particular information need. That is, the concept of information has both personal

[1] This section is adapted from van Rijsbergen's 1979 book [3] for the most part. I added references to IR work in the eighties and nineties, and put this introduction into HDDI's perspective.

[2] The endosystem [5] consists of those factors that the designer can specify and control such as the equipment, algorithms and procedures used.

> and time-dependent components that are not present in the concept of data. For example, even if a system can provide to the user a list of ingredients in a certain breakfast cereal or the names of the signers of the Declaration of Independence, these data are not information if they are already known to the user or have no relevance to the user's need. In effect these data become noise in the system, disrupting the user's awareness and concentration. In addition, they potentially dilute the system response to any information need.

Even if information retrieval is not only about *document* retrieval, it has been accepted as a description for the work published by Cleverdon [6], Salton [7, 8], Sparck Jones [9] and others. Instead of documents, one can consider other pieces of information to be retrieved: similar terms and sentences, for example. For the sake of simplicity we will describe IR as retrieving documents or items, but one has to keep in mind that a document is not the only chunk of information that can be returned.

Another perspective is to consider information retrieval (IR) in relation to data retrieval (DR). The main aspect that differentiates IR from DR is that DR normally looks for an exact match – it returns a boolean answer to the request saying if the there is an item that exactly matches the request. IR will rather find all the items that partially match the query and then rank them with respect to a given set of criteria, only to return the best matches.

Data retrieval is generally described as deterministic, whereas information retrieval is essentially probabilistic. While DR uses simple yes/no relationships between elements (a is related to b, or is not), IR allows a degree of uncertainty in these relationships, and assigns a measure on it.

As van Rijsbergen puts it [3], data retrieval typically uses a *monoethic* classification: in order to belong to a class, an object must have a set of attributes both necessary and sufficient to belong to a class. In IR such a classification would not be very useful; a *polyethic* classification is rather used, one where an object will only share some attributes with the other class members.

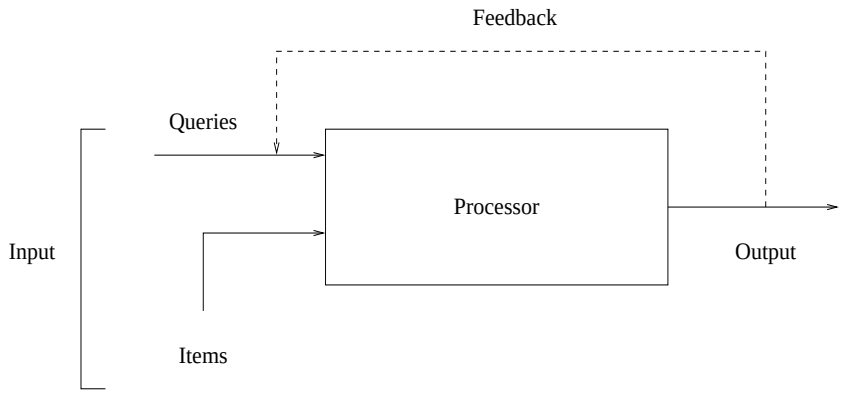

Figure 2.1 A typical IR system

Yet another difference between IR and DR is the way the queries are entered. DR requires the use of an artificial language with a restricted syntax whereas IR allows queries entered in natural language. This is due in part to the fact that IR will be much more tolerant to errors than DR. In IR, we look for a set of relevant documents, while in DR we want exactly matching items.

2.1.1 An information retrieval system

In this section, I will describe a general IR system using Figure 2.1 adapted from the typical IR system Van Rijsbergen describes in [3]. This is a framework simple enough to address many practical implementations in a general way. The whole system is based on three main components: input, processor and output.

Let us tackle the input side first. Here we want to get a representation of the documents and queries that a computer can use. Most systems will use a *document representative* rather than the plain input document. It can be a list of terms or, as we will see with HDDI, a list of noun phrases. Another approach to this problem is to directly submit the inputs (queries and/or documents) in an artificial language.

Provided that the retrieval system is on-line, the user can get feedback from the system and change its request accordingly. This is a major means for improving the effectiveness of information retrieval systems.

The processor is the second part of the retrieval system. This is the module actually concerned with structuring the information in order to actually retrieve the relevant items. This process usually involves some sort of classification and a strategy for performing the actual search over the resulting structure.

Eventually the processor does or does not find relevant documents and outputs the results. These typically consist of a set of items (documents or other chunks of information). At this point the processing part of IR is done, but the results need to be analyzed in order to know how well the system performs.

2.1.2 The three dimensions of information retrieval

Following van Rijsbergen [3], we identify three main areas of research in IR: content analysis, information structures, and evaluation. The first is concerned with describing the contents of documents in a suitable form for computer processing. The second, information structures, focuses on exploiting relationships between the pieces of information in the input, in order to improve efficiency and effectiveness of retrieval strategies. The third, evaluation, addresses the issues of measuring the effectiveness of retrieval.

2.1.2.1 Content analysis

Van Rijsbergen reports that since Luhn (see [10]) in the late 1950s, frequency counts of words in a document text have often been used to determine which words were significant enough to represent the document in the computer. These words have thus been called *keywords*. In addition, the frequency of occurrence of these words in the body of the text were also used to indicate how significant they are one compared to the other. This provided a simple weighting scheme for the keywords, and documents were represented using a *weighted keyword description* of the document.

At this point I want to make clear what is meant by *keyword* with respect to *term*. Traditionally the two have been used interchangeably, but in the scope of this thesis, we

will use term only to refer to a single word keyword (e.g., "car"). However, notice that a keyword can also be a set of words such as a *noun phrase* (e.g., "brand new blue car").

Van Rijsbergen reports that the use of statistical information about distributions of words in documents was exploited by Maron and Kuhns [11], and Stiles [12] who obtained statistical associations between keywords. In order to achieve better retrieval results these associations were used to build a thesaurus. The 1964 Washington Symposium on Statistical Association Methods for Mechanized Documentation [13] is a good reference for this early work. Karen Sparck Jones used measures of association between keywords based on their *frequency of co-occurrence*, that is, the frequency with which two keywords occur together within the same document. She has shown that such related words can be used to increase the proportion of the relevant documents which are retrieved [9].

2.1.2.2 Information structure

The *information structure* deals with the way the information is organized for the purpose of information retrieval. This is a crucial part of the process; if not executed properly, the information structure can lead to very poor efficiency. A popular organization is the inverted file where for each keyword, the documents in which it occurs are listed. Many experiments have attempted to demonstrate the superiority of another organization, in clustered-files, potentially better for on-line retrieval. The organization of these clustered files is produced by an automatic classification method.

Over the last 40 years, these automatic classification techniques have been the subject of an extensive research practice. Van Rijsbergen reports that Good [14] and Fairhorne [15] first suggested that the use of automatic classification in document retrieval. Doyle [16] and Rocchio [17] carried out the first serious experiments in document clustering back in 1965. Over the 1970s Salton's SMART system [7] provided the necessary tools for the research community to carry out experiments in this field. Over the 1980s document clustering has still been used extensively, and new methods have also been investigated from the late 1980s until now, such as Latent Semantic Indexing (LSI) by Susan Dumais [18]. LSI is based on an SVD (singular vector decomposition) of the term to document

matrix; it then reduces the dimension of the space to find the latent relationships between terms. It is different from document clustering because clusters are not actually formed; during the retrieval phase the query vector is mapped into the smaller dimension space and a distance metric is used to return the closest documents.

2.1.2.3 Evaluation

The first comparative test of information retrieval dates back to the first Cranfield test, between 1958 and 1962 [19]. Later on, in the 1970s, the SMART retrieval system [7], [20], and Sparck Jones [9] experiments are characterized by the use of test collections of documents, requests and relevance judgments that will be needed, and by the view of information retrieval as a batch process. Interaction with the user is not really compatible with this approach [21]. In 1981, Karen Sparck Jones edited *Information Retrieval Experiment*, a good summary of what had been done at the time. In late 1991, TREC (Text REtrieval Conference) started [22]. TREC provides researchers with a set of collections and queries. Results are submitted to the organizers for relevance evaluation.

Today effectiveness of retrieval is mostly measured in terms of precision (the proportion of retrieved documents which are relevant) and recall (the proportion of relevant documents retrieved) or by measures based on them.

2.1.2.4 Effectiveness and efficiency

The main purpose of the research in information retrieval is to improve *effectiveness* and *efficiency* of retrieval. Efficiency is usually measured in terms of the computer resources used such as memory used, storage necessary, and CPU time. Efficiency measurement is generally machine-dependent. In the previous section I mentioned that effectiveness is usually measured in terms of precision and recall. Because these measures are central to any study in information retrieval, let me state the definitions for precision and recall here:

$$\text{Precision} = \frac{\text{Number of relevant documents retrieved}}{\text{total number of documents retrived}} \tag{2.1}$$

$$\text{Recall} = \frac{\text{Number of relevant documents retrieved}}{\text{total number of relevant documents}} \tag{2.2}$$

The reason for emphasizing these two measures is that I make frequent reference to retrieval effectiveness but its actual use is delayed until Chapter 4. It will suffice until we reach that chapter to think of retrieval effectiveness in terms of precision and recall.

2.2 Overview of HDDI

Hierarchical Distributed Dynamic Indexing (HDDI) is a technology for organizing very large distributed and dynamic data sets into a cohesive hierarchical framework for optimal probabilistic search.

Our primary objective in the Hierarchical Distributed Dynamic Indexing project is to create a software system whereby large, distributed collections are automatically indexed for use in concept-based knowledge searches. The difficulty in creating indexing systems that effectively perform concept-based search is a well-known problem [23].

The following steps compose the HDDI system:

- Concept extraction
- Matrices build
- Hierarchy build
- Knowledge base creation
- Regions of semantic locality detection
- Hierarchy map formation

2.2.1 Concept extraction

During the concept extraction process, parts of speech within documents are tagged and all concepts (e.g., nouns and noun phrases in textual collections) occurring within the document collection are identified. Heuristics are applied to prune the result set. Documents are currently stored in Extensible Markup Language (XML) format. The resulting set of concepts is then used in the matrices build process.

2.2.2 Matrices build

Concepts extracted during the previous process are used to compute concept frequency and co-occurrence matrices. Concepts which occur within the same document are defined as co-occurring. The frequencies of co-occurrences of concept pairs among all documents in the set are also computed.

2.2.3 Hierarchy build

The hierarchy build is a metalevel organizational process that combines the matrices built in the previous step. These co-occurrence matrices provide the basis for organizing concepts into an ontology of knowledge based on the content of the collections. Systematic filtering, pruning, and meshing lower level (child) matrices form the hierarchical structure, producing higher level (parent) combined matrices as show in Figure 2.2. The process is iterative in that matrices are pruned and merged at successively higher levels. The resulting hierarchical index consists of high-resolution leaf-level index nodes that become increasingly less precise (i.e., lower resolution) as the hierarchy is built.

2.2.4 Knowledge base creation

Knowledge base formation is the second metalevel organizational process. For each concept in each matrix in the hierarchy, co-occurring concepts are ranked, resulting in a one-to-many mapping where each concept is associated with a list of related concepts

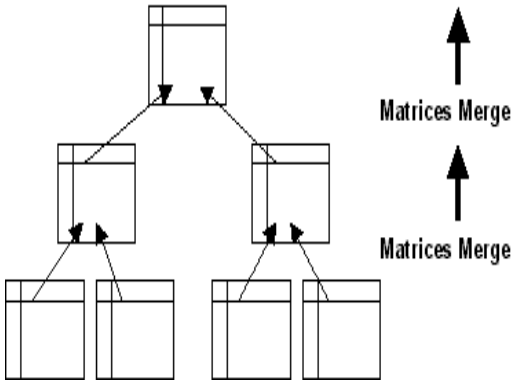

Figure 2.2 Hierarchy build

ranked by similarity. More general concepts are given a smaller weight, resulting in a lower similarity. Co-occurring concepts are ranked in decreasing order of similarity, with the result that more general concepts occur lower in the list of co-occurring concepts. Each concept pair (concept to ranked concept) is weighted, creating asymmetric measures of pairwise similarity between concepts [24, 25]. The knowledge base can be visualized as a graph, illustrated in Fig. 2.3, where vertices represent concepts and edges represent the pairwise similarity between two concepts.

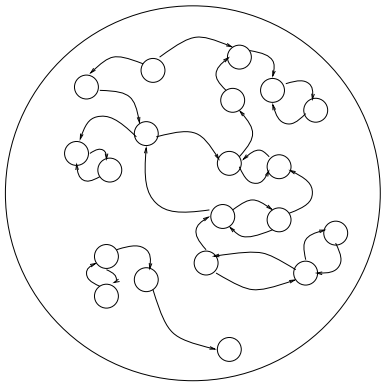

Figure 2.3 Knowledge base

2.2.5 Semantic locality detection

The resulting weight assignments from knowledge base creation are context-sensitive, and are used to determine regions of semantic locality (i.e., conceptual density) within each node of the hierarchy. During this phase focussed clusters of concepts within each knowledge base are detected. The result is a hierarchy of knowledge bases composed of regions of high-density clusters of concepts – subtopic regions, if you will. In simple terms, these regions consist of clusters of concepts commonly used together that collectively create a *knowledge neighborhood*. Regions of semantic locality are illustrated in Figure 2.4; each region is bounded by a dotted line.

Chapter 3 describes *sLoc*, the algorithm we designed and implemented to find regions of semantic locality. The resulting set of regions is called a *knowledge neighborhood*.

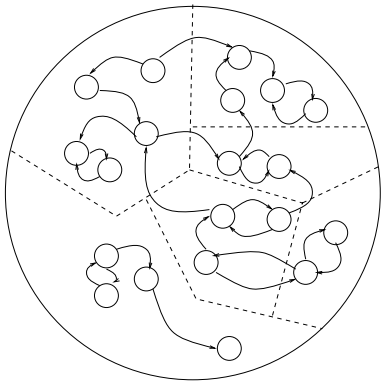

Figure 2.4 Regions of semantic locality in a knowledge base

2.2.6 Hierarchy mapping

The final process is to create a mapping of the information space by developing linkages between the clusters identified during semantic locality detection. These mappings provide the necessary links, or paths, between information areas at the higher level of the hierarchy and the lower levels of the hierarchy. Map linkages are based on computed density and probability of query match at each node level. This process defines relation-

ships between nodes, so that nodes are "aware" of the lower child nodes, which they were merged from, and of the higher parent node they were used to create. Upon completion of this phase, each node contains information needed to determine its relative position as a knowledge area in the knowledge hierarchy. Figure 2.5 depicts conceptually the operation of such a model.

2.3 HDDI Applications in Information Retrieval

HDDI, its concepts and inner algorithms can be used for a variety of applications including, but not limited to, query search, hot topic detection, and targeted marketing analysis.

2.3.1 Query search and retrieval

Using the hierarchy mapping described in Section 2.2.6, and mapping a query onto a node of the HDDI, we can implement a query search engine. The very issues of query search using HDDI represent a whole piece of research yet to be investigated. The basic idea is to map a query onto one or more HDDI nodes and to use the information in the HDDI to find the relevant items. However, other kinds of search can be implemented using HDDI. The information contained in each knowledge neighborhood can be used for *term suggestion*, for example.

In terms of precision and recall, the main advantage of HDDI over traditional search engines comes from its hierarchical nature. Indeed, by leveraging the fact that the resolution of the indexes become lower as the position of the node is higher in the tree, we get a versatile search engine able to adapt to queries of any resolution. Depending on the level of interaction desired between the application and the user, the HDDI search engine can either automatically adapt to the various levels of resolution in the query or give some feedback to the user (e.g., letting the user know if the query is too general or too precise to return meaningful results).

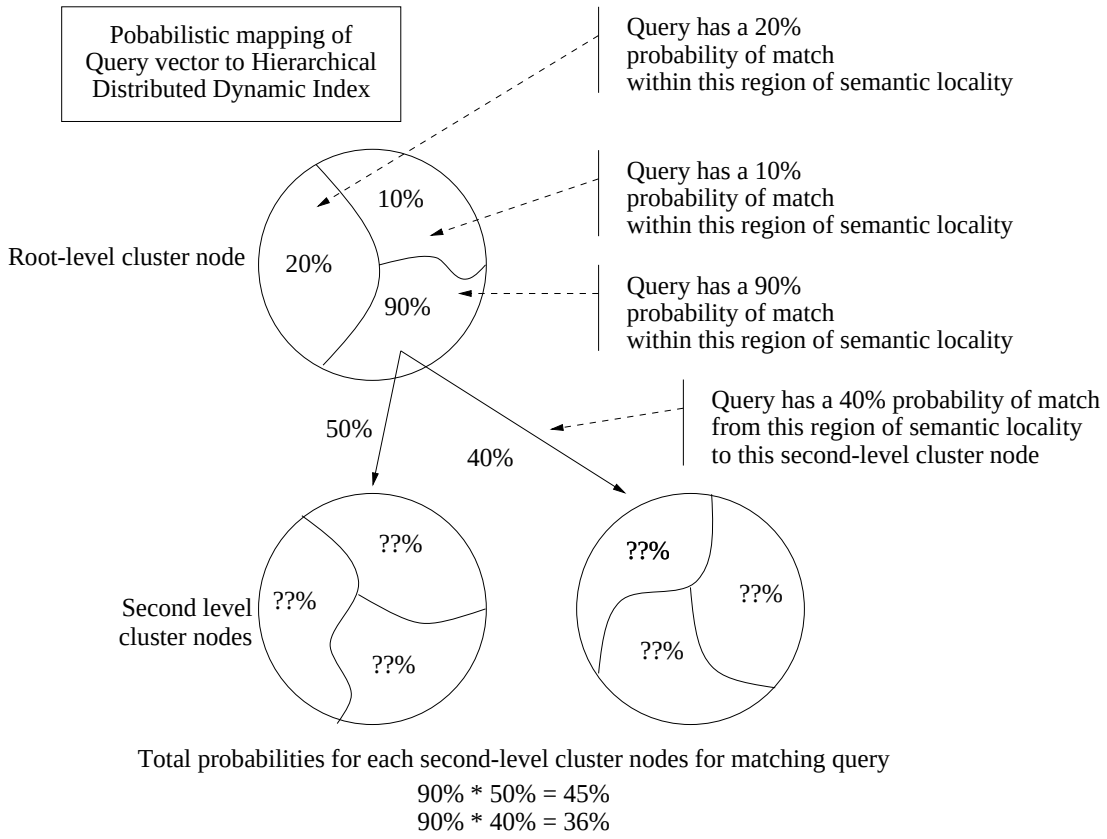

Figure 2.5 Query mapping process

2.3.2 Detection of emerging conceptual contents

Detecting emerging conceptual contents in textual collections is a growing concern of major companies who want to keep up with technology.

The idea here is to build HDDI nodes from time-stamped collections. For example, one HDDI node built from 1998 articles, and the other from 1999 articles. The basic idea then is to compare these two HDDI nodes in order to identify the concepts that emerged between 1998 and 1999.

We assume that there are two important features that an emerging concept should have. The first one is that, in order for a noun phrase to be an emerging concept, it should be semantically richer at one time than it was before. The second assumption is that an emerging concept should occur more frequently, since more and more documents

will reference it. The semantic richness of a noun phrase can be approximated by the number of other noun phrases that are in the same region of semantic locality.

We further assume that, to be an emerging concept, the number of occurrences of a particular noun phrase should exhibit an accelerating occurrence in a large corpus. Also, the number of occurrences cannot be too high; otherwise, it might be a redundant noun phrase.

Combining all these assumptions, we conclude that if we have a statistically significant sample of documents, by using these two indexes (size of cluster, occurrences) we will be able to identify emerging concepts.

2.3.3 Targeted marketing analysis

A targeted marketing analysis study is being carried out at NCSA, using the HDDI technology to predict which web pages or content will be of interest to a particular user. This can be used in many applications: for human-computer interaction to display a personalized set of links that fit the user's profile, for advertisement, for optimization of the back-end resources (servers and network loads can be balanced when we know the groups of pages that are systematically visited together).

The implementation uses server logs over a certain period termed the *critical period*. These logs are used as input collections for HDDI, as shown Figure 2.6; an HDDI node is built and a knowledge neighborhood is determined for the server. At the same time, one HDDI node is built for each user, using the information known about the particular user. Each of these user HDDI nodes has a knowledge neighborhood that can be mapped to the knowledge neighborhood of the server.

One particular application of this is market basket analysis. An HDDI node can be built to represent all the products a given user has purchased. The knowledge neighborhood may then identify dairy products as a region of semantic locality. On the server side, dairy products may be included with bread and pastries in one single region of the knowledge neighborhood. Mapping the two regions one to the other can lead to the

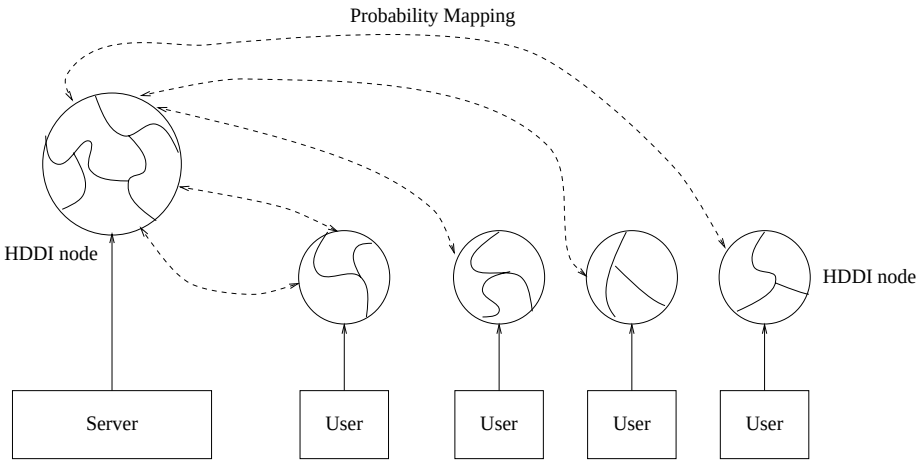

Figure 2.6 Sample mapping from user space to Web pages space

conclusion that, this user being a dairy products consumer, the store may look at him as a potential client for pastries and bread.

CHAPTER 3

SEMANTIC LOCALITY INDENTIFICATION

In order to build and search an HDDI, we want to identify the regions of semantic locality in the knowledge base at each node.

This chapter describes *sLoc*, the algorithm we designed and implemented to find regions of semantic locality. In other words, the purpose of sLoc is to derive the *knowledge neighborhood* (set of regions of semantic locality) from a knowledge base.

Remember that a knowledge base is created at each level of the hierarchy. A knowledge base is a graph representing the concepts (noun phrases) in the HDDI node linked with arcs of which the weights represent the similarities between concepts. The resulting weight assignments from knowledge base creation are context-sensitive, and are used to determine regions of semantic locality (i.e., conceptual density) within each node of the hierarchy. During this phase focussed clusters of concepts within each knowledge base are detected. The result is a hierarchy of knowledge bases composed of regions of high-density clusters of concepts – subtopic regions, if you will. In simple terms, these regions consist of clusters of concepts commonly used together that collectively create a *knowledge neighborhood*.

The motivation for the use of *semantic locality* comes from the commonly used premise that grouping similar concepts leads to increased effectiveness and efficiency in particular in query search and retrieval (see Sparck Jones [9]). Also, in the perspective of the HDDI hierarchy, semantic locality enables *tracking* of similar topics across the hierarchy.

I will first give a brief overview of how people have been addressing information clustering traditionally; then I will explain the algorithm and define a set of measures to assess its performance. Finally, I will illustrate its use with sample cases.

3.1 Previous Work

In this section I will review the traditional clustering methods used in information retrieval, their purpose, advantages and drawbacks. I will then present other algorithms used in graph partitioning and see why these were not applicable for the particular case sLoc addresses. We might want to note here that classification and clustering do not usually refer to the same thing. Classification tends to consider classes that are either given or computed at the beginning, and tries to classify new objects into these existing classes. Clustering is generally thought of as a dynamic process that creates and modifies the classes with respect to the objects it wants to classify. Clustering and classification will however be used synonymously in this thesis.

3.1.1 The use of clustering in information retrieval

In information retrieval the two most used classification schemes are: *term* classification and *document* classification. Notice that Sparck Jones's *keyword* classification[9], or HDDI's *concept* classification refer to the same idea: clustering terms rather than documents. Following Salton's description in [7]:

Term classification is designed to group terms into (synonym) classes in the hope of producing greater matching capacities between queries and document terms.

Document classification groups documents together to improve recall, but also to increase the response time of the retrieval system.

The two classifications are not independent because the terms assigned to the documents must necessarily form the basis for the classes produced by a document grouping procedure.

Term classification groups related low-frequency terms into common thesaurus classes (clusters, or regions of semantic locality). The terms included in a common class can then be substituted for each other in retrieval, and the recall output produced may be improved that way.

Document classifications make it possible to restrict the searches to the most interesting document classes only, thereby providing high-precision output [7].

3.1.1.1 The cluster hypothesis

Any sort of clustering in information retrieval (document, or keyword clustering) is based on the assumption that *closely associated documents (respectively keywords, or concepts in the HDDI terminology) tend to be relevant to the same request.* Van Rijsbergen [3] refers to this assumption as the *cluster hypothesis.*

3.1.1.2 Criteria of a good clustering

The first property we expect from a clustering method is that it provides a set of clusters that actually fit the constraints and requirements of our application semantics. However, the method needs additional properties in order to scale to any kind of input set. Jardine and Sibson [26] stated the major requirements of a good clustering method:

(1) The classification should be defined such that a given result is obtained for any given body of data.

(2) The classification should be independent of scale because a multiplication by a constant of the property values identifying the objects should not affect the classification.

(3) Small errors in the description of the objects must not lead to big changes in the clustering.

(4) The method must be independent of the initial ordering of the objects.

(5) Objects exhibiting strong similarities should not be split by being assigned to separate classes.

We argue that although true in the 1960s, these requirements may have to be reviewed in the context of the 1990s where the amount of data increases on a daily basis. The fact that the amount of data increases every day invalidates the need for the classification to be independent of scale; we rather want the classification to adapt to the scale, that is to create new classes or delete some to work best with the actual size of the current collection.

At the same time, another essential criterion in the assessment of a clustering method is the *efficiency* achieved by the algorithm. As stated in Section 2.1.2.4, efficiency is usually measured in terms of the computer resources used such as memory and storage requirements and CPU time.

Over the past 40 years, various methods have been used to achieve good efficiency and effectiveness. In 1979, Van Rijsbergen identified two major approaches [3] to information clustering, they are still good starting points for research in IR today:

(1) The clustering is based on a measure of similarity between the objects to be clustered.

(2) The cluster method proceeds directly from the object descriptions.

3.1.1.3 First approach: Methods based on similarities

Different considerations come into play when designing and assessing a clustering system for information retrieval. For example, one may want to strive for efficiency of the clustering process by defining the best split as the one produced at least cost in terms of memory size used, running time or number of operations used.

One may also wish to optimize some *classification criterion*, such as the cohesiveness between clusters, or the strength of the association between items included in a single class, or the corresponding weakness between items in different classes [7], [9]. Quoting

[7], let us consider class A of items, and let B represent the items not in A. A class may then be defined as the set A which minimizes the cohesion function

$$C = \frac{S_{AB}^2}{S_{AA} \times S_{AB}} \tag{3.1}$$

where S_{ij} denotes the similarity between items i and j.

S_{AB} can be defined as the average of all the pairwise similarities for the N_A terms in A and the N_B terms in B; that is

$$S_{AB} = \frac{1}{N_A N_B} \sum_{i \in A} \sum_{j \in B} S_{ij} \tag{3.2}$$

Other parameters can also characterize a classification process. Besides cost and cohesion, the resulting clusters can be forced to have a given structure. One common approach is to use the similarity measures between objects to cluster the graph. Let us consider a set of objects we want to cluster. We first compute numerical values of each pair of objects indicating their similarity. Notice that this similarity measure is usually symmetric (similarity from a to b is the same as similarity from b to a). However HDDI is based on asymmetric similarities [24].

We then chose a threshold value; two objects are considered linked if their similarity value is greater than this threshold. Clusters are defined as graphical patterns (e.g., stars, strings). For instance, we can consider a set of nodes to be a cluster if they form a *string*. Alternative patterns can be *stars* (each object is connected to one central object), or maximal complete subgraph, also known as *clique* (each node is connected to every other node). Yet another type is *clumps* where the similarity of each object with the collection of other objects in the cluster exceeds a given threshold. These class types are represented graphically in Figure 3.1 taken from Salton [7].

Every definition leads to different semantics given to the resulting set of clusters. An important parameter here is the *threshold* used to decide whether two objects will or will not be connected. Varying the value of this threshold leads to clusterings of different

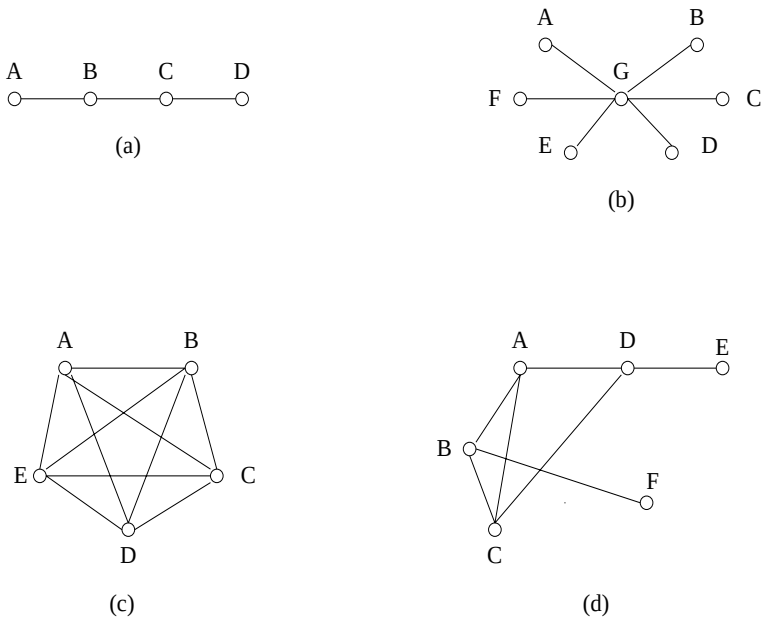

Figure 3.1 Typical class types: (a) string; (b) star; (c) clique; (d) clump

resolutions, i.e., clusterings where the size of a cluster including a given object will vary with the threshold value. A *hierarchy* can thus be created. The most important of these hierarchic methods is *single-link* [3].

Tables 3.1-3.4 show how, from the similarity matrix (3.1), connections can be established for different threshold values (0.7, 0.8 and 0.9)[1]. Each threshold corresponds to a different graph, and thus to a different set of clusters; the three resulting sets in the example are shown Figure 3.2. This figure exemplifies the issue of the value to be given to the threshold. One can set the threshold to the value that will fit application constraints the best, but the choice can also be made to let it take a set of values and build a hierarchy of clusters of *the same* underlying set of objects. This hierarchy is termed a *dendrogram*, and the method is called *single-link*. See Salton [7] for details about the single-link method. The single-link approach is based on clustering the same

[1]This example is adapted from van Rijsbergen's [3], but here we use similarities instead of dissimilarities to more closely model the HDDI approach.

unique model with various levels of thresholding. In contrast, in HDDI multiple models are clustered and used to build the hierarchy of HDDI nodes.

Table 3.1 Similarity matrix

1					
2	.6				
3	.6	.8			
4	.7	.7	.7		
5	.9	.4	.4	.9	
	1	2	3	4	5

Table 3.2 Binary matrix: threshold .9

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

Table 3.3 Binary matrix: threshold .8

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

Table 3.4 Binary matrix: threshold .7

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

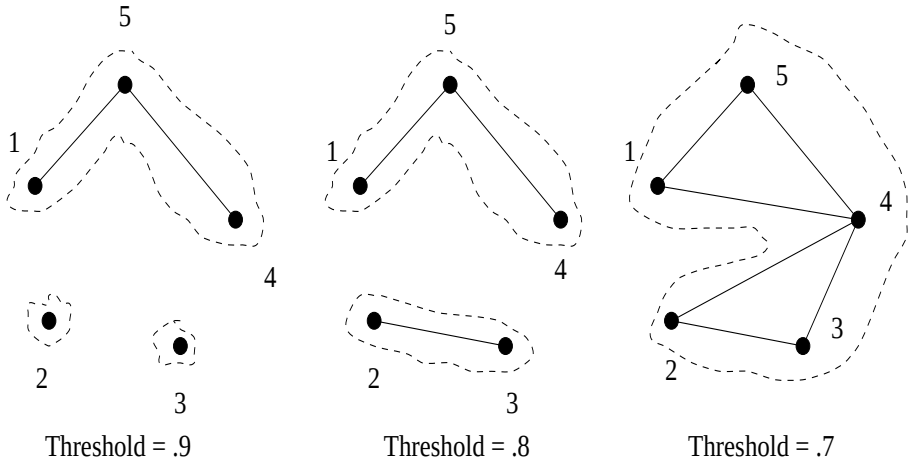

Figure 3.2 Resulting clusters for each value of the threshold

3.1.1.4 Second approach: Using object descriptions directly

Using the combination of class types, cohesion functions and dendrograms can then lead to good theoretical clustering. However, due to limited computing power, efficiency has been a major goal in the definition of the algorithmically defined cluster methods used in IR, in particular in document clustering. For this reason, many methods have preferred to proceed *directly from object description* to final classification *without* the expensive intermediate calculation of a similarity measure. The basic idea here is to perform a cheap *density test* to identify potential cluster centers and then cluster around these selected items [17]. Another characteristic of these methods is that they do not seek an underlying structure in the data but attempt to impose a suitable structure on it.[2]

In this section, following van Rijsbergen, I will give an overview of these methods. I will not describe in detail the heuristic algorithms here since they mainly apply to document clustering, but I want to outline the major ideas because they can still be applied to concepts clustering.

The most important concept is that of *cluster representative*. It is simply an object that summarizes and represents the contents of a cluster. The ideal cluster representative should be near every object in the cluster in some average sense; hence, the alternative name *centroid* for cluster representative. The similarity of the objects to the representative is measured by a *matching function*. The algorithms also use a number of *empirically* determined parameters, such as [3]

(1) the number of clusters desired;

(2) a minimum and maximum size for each cluster;

(3) a threshold value on the matching function, below which an object will not be included in a cluster;

(4) the control of overlap between clusters;

[2] We will see in Section 3.3 how sLoc, through macrometrics can take a similar standpoint.

(5) an arbitrarily chosen objective function which is optimized.

Following are the main three stages from [3]

(1) Select a number of objects as cluster centers. The remaining objects are then assigned to the centers or to a special cluster for outliers. From the initial assignment the cluster representatives are computed and all objects are assigned to the clusters once again. The final clusters might overlap.

(2) Iteratively change the clusters to allow the various input parameters to be adjusted so that the resulting classification meets the prior specification of the parameters more closely.

(3) Unassigned objects are forcibly assigned, and overlap between clusters is reduced. A good example of this kind of algorithm is Rocchio's [17] clustering algorithm.

3.1.2 Existing algorithms in graph partitioning

In our effort to find an algorithm for clustering knowledge bases, we also wanted to consider alternative approaches to classical IR clustering algorithms. In one sense, clustering a knowledge base can be cast as a graph partitioning problem; we therefore looked at the different problems that current graph partitioning techniques can help to solve, and evaluated the use of these techniques in HDDI.

Graph partitioning is an NP-hard problem[3] with numerous applications. It appears in various forms in parallel computing, sparse matrix reordering, circuit placement and other important disciplines. Several advanced software solutions are available from the internet. The most well-known are Chaco [27] and Metis [28].

Traditional graph partitioning algorithms compute a k-way partitioning of a graph such that the number of edges that are cut by the partitioning is minimized and each partition has an equal number of vertices. The task of minimizing the edge-cut can be

[3]NP stands for nondeterministic polynomial time.

considered as the objective and the requirement that the partitions be of the same size can be considered as the constraint.

Much of the current research in graph partitioning finds its application in parallel computing and load balancing. Therefore, the focus is mostly on getting a fixed number of clusters (that may correspond to the number of processors for example) and an equal number of vertices, in a nonoriented graph.

Another area of research where clustering is extensively used is image processing. The image processing research community has developed several linear or near linear algorithms for texture segmentation. While these algorithms have some disadvantages relative to clustering, they may allow significant improvements in visualizing some very large spatial data sets, such as detailed atmospheric data or SAR images [29].

Natural language processing, information systems modeling, program specification, information retrieval, machine learning and case-based reasoning sometimes use yet another kind of clustering method based on *conceptual structures*. Conceptual structures are a kind of knowledge representation developed by John Sowa in [30]. They are a graphical notation for typed first-order logic. Using these conceptual structures Mineau and Godin designed a conceptual clustering algorithm for knowledge bases [31].

In graph drawing disciplines the issue of visualizing large and complex graphs has also been addressed through clustering strategies. In [32] Sablowski and Frick demonstrate a new approach to clustering large graphs. They identify successive structural elements *(patterns)* in the graph. About the simplest pattern occurring in a graph are *leaves*, i.e., nodes with in-degree 1 and out-degree 0. Other patterns they use include *paths* (nodes with in-degree = out-degree = 1), triangles and similar simple structures.

Hinrich Schütze uses clustering for context-group dscrimination and disambiguation in computational linguistics [33]. In this study, similarity in word space is based on *second order co-occurrence*. Quoting Schütze:

> Words, contexts and clusters are represented in a high-dimensional real-valued vector space. Context vectors capture the information present in *second-order co-occurrence*. Instead of forming a context representation from the

> words that the ambiguous word directly occurs with in a particular context (first-order co-occurrence), we form the context representation from the words that these words in turn co-occur with in the training corpus. Second-order co-occurrence information is less sparse and more robust than first-order information.

This tells us that, depending on the context, allowing some transitivity in the similarity relation can improve the results of clustering based on similarity measures.

In data mining applications, Han, Karypis and Kumar [34] propose a clustering method based on hypergraphs that is expected to handle large input data sets better than the traditional k-means [35] or Autoclass [36]. The idea of the hypergraph model is to map the relationship of the original data in high dimensional space into a hypergraph. The hypergraph model is therefore based on the concept of a hyperedge. Quoting Kumar et al. [34]:

> A hyperedge represents a relationship (affinity) among *subsets* of data and the weight of the hyperedge reflects the strength of this affinity. A hypergraph partitioning algorithm is used to find a partitioning of the vertices such that the corresponding data items in each partition are highly related and the weight of the hyperedges cut by the partitioning is minimized.

This study used a number of partitions that was set *a priori*, and a different measure of similarity, which do not fit our requirements for sLoc. However [34] does introduce *fitness* and *connectivity* measures, which are akin to metrics used in sLoc (see Section 3.2.4.8 for a detailed comparison of these approaches).

In the final analysis we determined that none of the available algorithms were suitable for our application. Thus, the decision was made to base development on an existing linear graph partitioning algorithm that could be readily adapted to our needs [37].

3.1.3 Test collections

In the following chapters we will use various test collections to illustrate or validate our approach. Let me introduce them briefly here.

- Patterns 300 is a collection of 300 postings to a discussion list about computer software systems.
- MED, CISI and ADI are three collections of respectively 1033, 112 and 82 abstracts. MED is a sample from MEDLINE, the National Library of Medicine's database of references to articles published in biomedical journals. The CISI and ADI collections both contain information science documents.
- The Grainger collection contains 60 000 abstracts from the Grainger Engineering Library at University of Illinois.

3.2 The Semantic Locality Finder: sLoc

In this section we discuss the details of the algorithm, system design and implementation of the Semantic Locality Finder (sLoc).

The function of sLoc is to identify regions of semantic locality. For now, we can qualitatively characterize a region of semantic locality as follows:

- Concepts inside a region of semantic locality are similar to each other.
- Concepts belonging to different regions of semantic locality are dissimilar.

Similarity between concepts is defined quantitatively in [24, 25]. It is a mapping from one concept to another that quantitatively determines how similar they are semantically. We term the resultant mapping a *knowledge base*.[4] A knowledge base is represented as an asymmetric directed graph in which nodes are concepts and arc weights are similarity measures. Determining regions of semantic locality in such a graph involves clustering

[4]Note that follow-on work by Chen (of [24]) terms this a *concept space*.

or splitting the input knowledge base into a set of subtopic regions - smaller knowledge bases, if you will. These subtopic regions are regions of semantic locality or conceptual density. The number of concepts in each region of semantic locality may vary and the total number of regions is determined dynamically by sLoc. sLoc processes a knowledge base and outputs a *knowledge neighborhood* that consists of multiple regions of semantic locality.

3.2.1 Contextual transitivity in sLoc

The similarity relation is by definition not transitive. The theoretical basis for sLoc is the concept of what I call *contextual transitivity* in the similarity relation. In essence, this means that depending on the context (structure and distribution of the similarities in the knowledge base), a threshold is decided upon and transitivity is constrained accordingly. Contextual transitivity extends Schütze's conceptualization of *second order co-occurrence* [33] by using n-order co-occurrence, where n varies with the underlying semantic structure of the model.

In order to group concepts together in a region of semantic locality, sLoc uses contextual transitivity. Here is a simple example. Let us consider the three concepts *Java*, *applet* and *e-commerce*. If the concepts occur in the same document, they are called co-occurring concepts. Let us assume that *Java* and *applet* co-occur and that *Java* and *e-commerce* also co-occur. This means that there is at least one document that contains both *Java* and *applet*, and that there is at least one document that contains both *Java* and *e-commerce*, but *applet* and *e-commerce* do not co-occur necessarily. However it may be natural to gather the three concepts in one unique class and extrapolate that *e-commerce* and *applet* are actually related. That is what contextual transitivity means.

Note that similarity is not a Boolean variable, it can take a whole range of values. Two concepts can thus be more or less similar. In sLoc considerations on the knowledge base structure lead to a heuristic minimum similarity value for two concepts to be in the same cluster, and therefore to *constrain* transitivity. It is called *contextual* because this

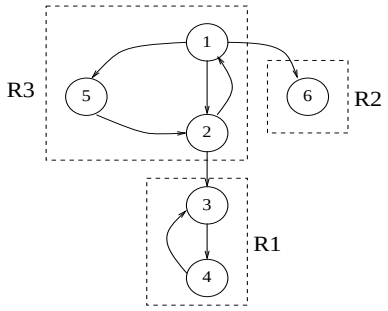

Figure 3.3 Strongly connected regions of a graph

minimum similarity value depends on the structure and distribution of the similarities in the knowledge base.

3.2.2 Design

The core of sLoc is based on an algorithm due to Tarjan [37, 38]. Tarjan's algorithm uses a variant of depth-first search to determine the strongly connected components of a directed graph. This was the first algorithm to solve the problem in a linear time. This is an important feature due to the fact that graph clustering is a NP-hard problem and the only heuristics we are aware of are not linear. The theory can be found in [38]. Figure 3.3 shows how Tarjan's algorithm identifies strongly connected regions (R_1, R_2, R_3), in a simple graph.

3.2.2.1 Notation

Before tackling the algorithm itself we must first introduce the following notation:

- Let $\mathcal{N}$ be the set of nodes i in the input graph, and let N be the total number of nodes.
- Let $\mathcal{A}$ be the set of arcs in the input graph, A the total number of arcs. An arc $a_{i,j} \in \mathcal{A}$ goes from node i to node j.

- Let $\mathcal{W}$ be the set of arcs weights in the graph, $w_{i,j}$ is the weight of the arc going from node i to node j.

Therefore $\mathcal{W} = \{w_{i,j}\}_{(i,j)\in\mathcal{N}^2}$. A knowledge base is an asymmetric graph therefore $w_{i,j}$ may differ from $w_{j,i}$. Moreover, if $a_{i,j} \notin \mathcal{A}$ then $w_{i,j} = 0$; in particular, for all i, $w_{i,i} = 0$. Now, let M be the mean of the arc weights:

$$M = \frac{1}{A} \sum_{(i,j)\in N^2} w_{i,j}$$

We call SD the standard deviation of the distribution of arc weights:

$$SD = \sqrt{\frac{1}{A-1} \sum_{(i,j)\in N^2} (w_{i,j} - M)^2}$$

3.2.2.2 The algorithm, step by step

Figure 3.4 depicts the three steps of the sLoc process. These steps are outlined in the following sections on the algorithm and its applications.

The first step in sLoc is to statistically "prune" the input graph before applying Tarjan's algorithm. Arcs of weight smaller than a certain *threshold* value τ are virtually pruned. This pruning and the process described in Figure 3.2, page 25, are very much alike. The difference here is that similarities are asymmetric; that is, there can be potentially two arcs between nodes in a given pair. Therefore, the arc from concept a to concept b can be pruned while the arc back from b to a remains.

The second step involves the actual identification of the clusters within the graph. Tarjan's algorithm is applied to find strongly connected regions. At this stage each strongly connected region is a cluster. The size of a given cluster is the number of nodes (concepts) it contains.

During the third step, clusters of size smaller than parameter s are discarded (they are assumed to be outliers). We interpret the remaining clusters as regions of semantic locality in the knowledge base.

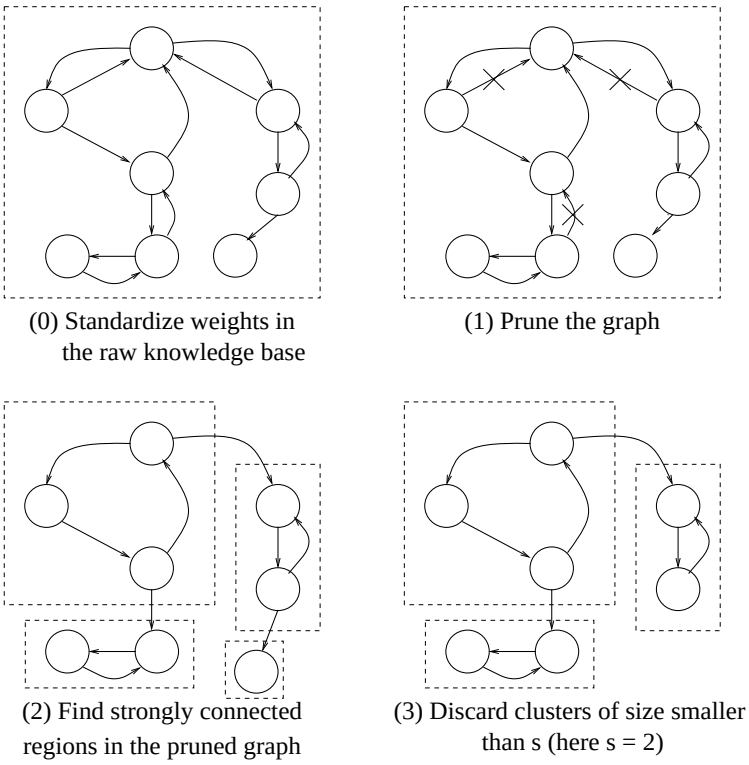

Figure 3.4 The sLoc process

The greater τ, the more arcs are cut off, and therefore the smaller in size the strongly connected regions. Thus the greater τ the smaller in size and the more focused will be the regions of semantic locality.

Two questions have arisen in the course of our work: (1) Given an input knowledge base, what threshold τ should be applied? (2) Is the usability of a given clustering dependent on the application?

Let us tackle the first question on how to determine the value of τ yielding an optimum clustering. Our premise is that the optimum τ can be determined statistically as a function of the mean M, the standard deviation SD and other knowledge base dependent variables (e.g., size, maximum weight, etc.).

For leaf-level collections in the HDDI, i.e., for those collections for which the knowledge base is directly computed from the co-occurrence frequencies in the collections, we use

$$\tau(\alpha) = \text{Max}(\mathcal{W}) - \alpha \times SD \tag{3.3}$$

In the equation above, τ is the cut-off weight used to prune the graph and α is a number of standard deviations. Thus, $\tau(\frac{1}{2})$ is the threshold corresponding to the maximum weight in the graph minus half of the standard deviation of the distribution of arc weights. Section 3.3 summarizes our results for a range of values of the parameter α. Figure 3.5 shows the distribution of arc weights at a leaf-level HDDI node, for the MED collection.

However, as we consider higher level nodes in the HDDI, the underlying distribution of weights changes to approximate more closely a normal distribution (see Figure 3.6). For the interior nodes, we use

$$\tau(\alpha) = M + 2SD - \alpha \times SD \tag{3.4}$$

I did not factor SD in the equation above in order to emphasize that the maximum value τ can take is $M+2SD$, the value below which 97.5% of a normal distribution occurs. It makes more sense to use this value here, rather than $\text{Max}(\mathcal{W})$, because $\text{Max}(\mathcal{W})$ does not make much statistical sense when the distribution of $\mathcal{W}$ is normal.

At this point in time we are still investigating answers to the second question posed above. Apparently, there is no ready-made definition. Our intuition tells us that the quality of a given clustering is related to the application: in other words, the *application* will likely play a key role in determining whether a given clustering is appropriate or not. If so, a given input knowledge base will have many optima, each one suited to a particular application (e.g., search engine, detection of emerging conceptual content, targeted marketing, etc.).

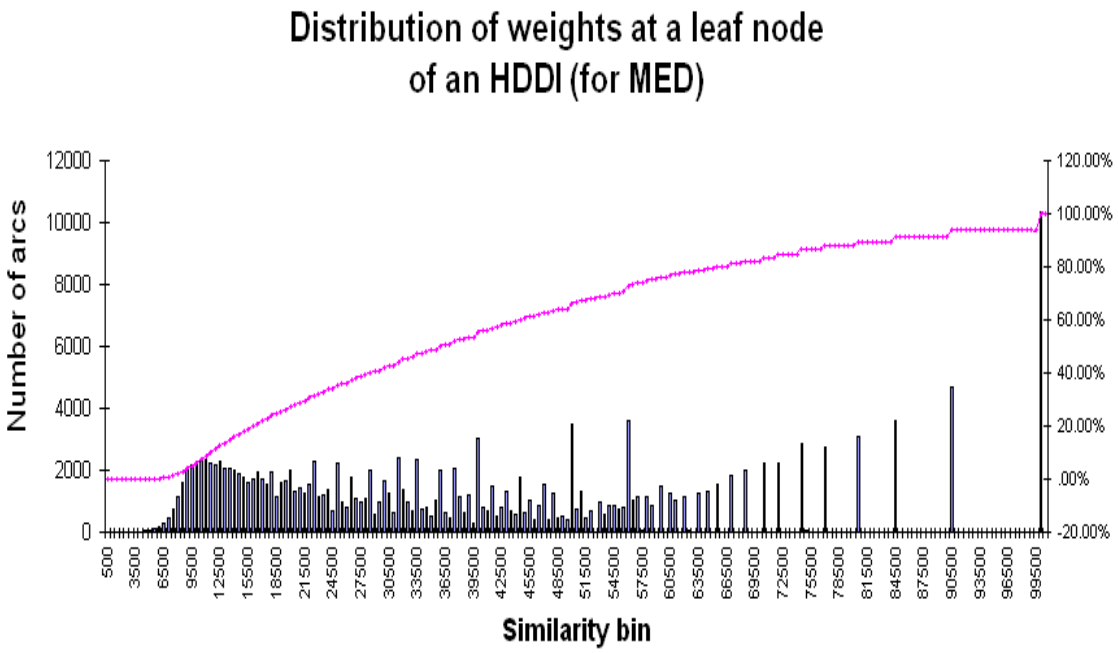

Figure 3.5 Distribution of weights at a leaf-level node

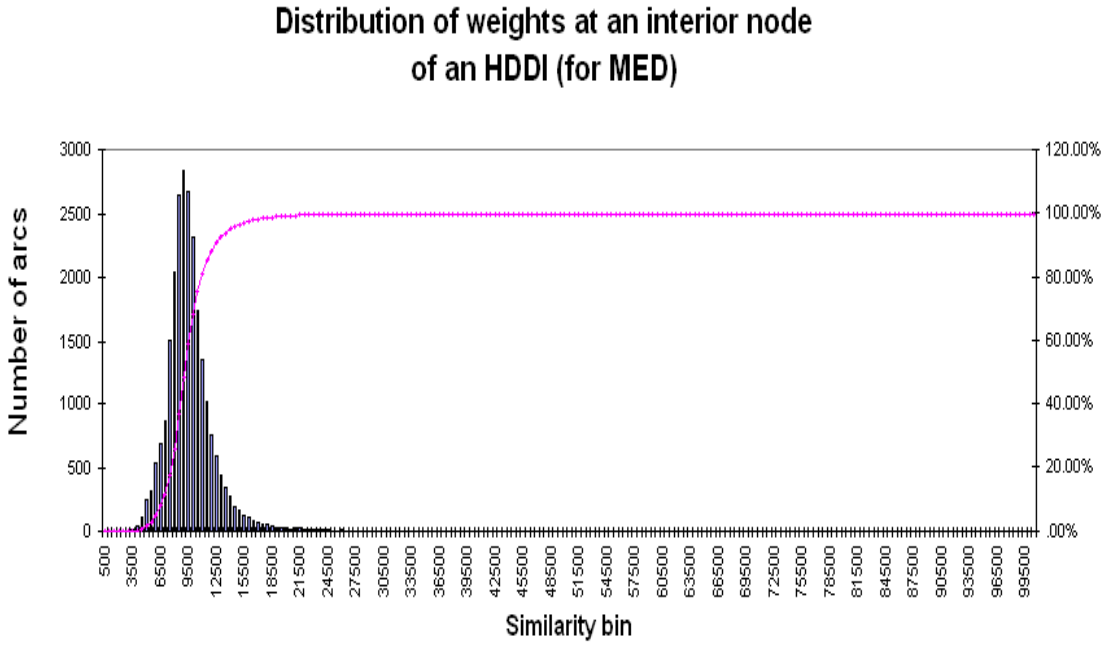

Figure 3.6 Distribution of weights in an interior node

3.2.3 Implementation

The core of the implementation of sLoc is Tarjan's algorithm. The following sections will give its complexity and pseudo-code.

3.2.3.1 Complexity

Tarjan's algorithm [37, 38] uses a variant of depth-first search to determine the strongly connected components of a directed graph. This was the first algorithm to solve the problem in a linear time. More specifically, Tarjan's algorithm has a complexity of $O(\text{Max}(N, E))$ where N is the number of nodes and E the number of edges.

The first step in sLoc involves computing M and SD, the mean and standard deviation of the distribution of the weights; this operation is still linear. Table 3.5 shows our results with 10 randomly picked sample collections out of the Grainger Abstracts Database (60 000 abstracts). The size of the collection was randomly determined and the actual documents included in the collection were randomly picked. Figure 3.7 plots the running time versus $\text{Max}(N, E)$, and it looks linear as expected. Table 3.6 shows the results of the regression analysis for this plot; all three methods (multiple R, R square and adjusted R square) lead to values that are very close to one. Therefore, both approaches corroborate the theoretical linear complexity of the algorithm.

3.2.3.2 Pseudo-code

The pseudo-code for Tarjan's algorithm, the "core" of sLoc, is given in Figure 3.8. It essentially calls the function `search()` recursively.

3.2.3.3 Platforms

I actually coded three different implementations of sLoc. The first one in Java is designed to read the input knowledge base from files and output the set of clusters and measures into files as well. The second is also coded in Java, but it uses data structures

Table 3.5 Running times for different input sets and $\alpha = 1.5$

Sample number	Number of documents	Number of nodes (concepts)	Number of arcs	Running Time (s)
0	49865	227151	5062672	7.55
1	36091	198813	4441346	7.52
2	3544	35947	779811	1.89
3	9522	82794	1815573	3.90
4	15498	118927	2623671	4.86
5	47671	223951	5004694	7.75
6	52154	231907	5183729	8.33
7	2750	29606	649800	1.72
8	36418	200305	4466191	7.62
9	40901	210952	4720938	7.64

Table 3.6 Regression statistics

Method	Value
Multiple R	0.993016017
R Square	0.98608081
Adjusted R Square	0.984340911
Standard Error	0.321181934
Observations	10

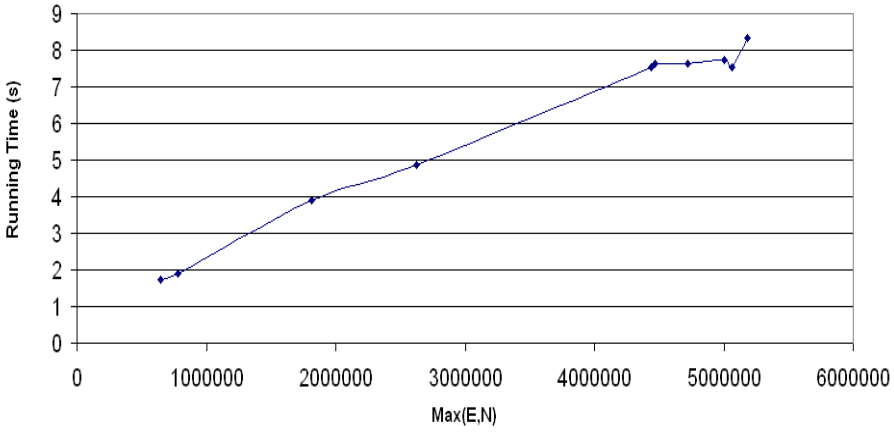

Figure 3.7 Running time versus $\text{Max}(N, E)$ for sLoc

```
void search(n) {
    visited(n) = true;
    dfn(n)     = df_count;
    link(n)    = df_count;
    df_count++;
    push(n);
    for every n' in succ(n) do {
        if not (visited(n'))
        then {
                search(n');
                link(n) = min(link(n),dfn(n'));
        }
        else { /* visited(n') */
                /* n' is in the same strongly connected
                  component as an ancestor of n */
                link(n) = min(link(n),dfn(n'));
        }
    }// end for

    if (link(n)=dfn(n)) then {
        /* n is the root of a strongly connected component
           whose nodes are the topmost elements of STACK, up
           to and including n. These elements are output and
           popped off the stack */
        r++;
        write("Strongly connected component number:",r);
        repeat {
          x = top_of_stack();
          pop();
          write(x);
        } until (x==n)
    }
} // end search

void main(){
    df_count = 1 ;
    r = 0 ;
    for every node (n) do {visited(n) =false}
    while there is a node (n) such that not(visited(n))
        do {search(n);}
}
```

Figure 3.8 Pseudo-code for Tarjan's algorithm

allowing it to be integrated in the Java implementation of HDDI. Both Java versions were coded using JDK 1.2 because of the library it provides to work with collections.

Java is a very powerful tool for rapid prototyping, although large memory requirements are not supported by the Java virtual machine because it can handle only 32-bit addressing on the machines employed in our experiments. In order to have a version of sLoc that we can use for experiments with large collection sizes, I also wrote a C^{++} on the SGI Origin2000, using the Polaris library [39].

3.2.4 Clustering measures

In order to find the optimum where the threshold τ leads to satisfactory regions of semantic locality, we introduce some measures on the resulting knowledge neighborhood.

We distinguish between what we call "micrometrics" assessing the interior structure of the clustered graph and "macrometrics" assessing the knowledge neighborhood quality based on measures that are independent of its internal structure.

Macrometrics will not take into account the underlying graph structure. We can draw a parallel between these measures and the input parameters used in the second approach to clustering described in Section 3.1.1. The main macrometrics are the number of clusters, the size of the clusters, and the coverage; they will be described later in this chapter.

Micrometrics give some feedback about the *internal structure* of the clustered graph; they measure the similarity of concepts in a cluster, and the dissimilarity of concepts in different clusters. However, these measures are very expensive computationally.

Moreover, the quality of these micrometrics is very dependent on the graph structure. In order to avoid a getting a huge set of measures, one for each cluster for example, we want to use a "reduced" measure for each micrometric. The reduced measure represents the distribution of a given metric across the whole knowledge neighborhood. However, the less homogeneous the underlying graph, the less these measures make sense, because one average cannot represent the complexity of a heterogeneous distribution.

In defining the micrometrics, we take an approach similar to that of Sparck Jones [9] and compute measures similar to those of Section 3.1.1.3. The *intercluster* and *intracluster* densities for a given knowledge base are computed over all clusters remaining after the completion of phase 3 (see Figure 3.4).

Let us call $C_1, C_2 \ldots C_n$ the n clusters of nodes found by sLoc. Notice that, mathematically, $(C_1, C_2 \ldots C_n)$ is *not* a partition of $\mathcal{N}$: we must add all the clusters of size smaller than s to get a real partition of $\mathcal{N}$. $(C_1, C_2 \ldots C_n)$ only covers part of the input knowledge base, the clusters of size smaller than s, rather than being forced into a larger cluster are simply discarded. Traditional clustering methods such as Rocchio's [17] try to force the outliers into existing clusters. In HDDI we want to leverage our knowledge of outliers and allow sLoc to play a role in "trimming" the knowledge base. Therefore, all the concepts in clusters of size smaller than s will be ignored for any further use at the leaf level of the hierarchy.

3.2.4.1 Coverage

At the end of the process depicted in Figure 3.4 clusters of size smaller than s are discarded. For large values of τ (respectively, small values of α) the number of nodes discarded can be fairly high. In other words, the ratio of the number of nodes in clusters of size greater than s to the total number of nodes present in the knowledge base originally is fairly low for large values of τ. We call the this later ratio the coverage c in the HDDI terminology. The coverage c is the percentage of the knowledge base actually represented in the regions of semantic locality sLoc outputs.

$$c = \frac{\sum_{k \in [1,n]} N_{C_k}}{N} \tag{3.5}$$

For example, a coverage of .9 means that 90% of the nodes in the input knowledge base will actually appear in a cluster of the knowledge neighborhood. The larger c the larger the number of concepts that will be retrievable through the knowledge neighborhood.

3.2.4.2 Number of clusters

The number of clusters n at the end of phase 3 (see Figure 3.4, page 34) is a good experimental indicator of distribution of clusters. We will see in Section 3.3 how the quality of the clustering relates to the number of clusters. Notice that n not only depends on the threshold τ, but it is also determined by the minimum cluster size s that we impose. As for the other measures, the way s is set must reflect the application needs. For some applications, or some input sets $s = 2$ might be too small.

3.2.4.3 Cluster size

The distribution of cluster sizes is characteristic of the clustering too. Section 3.3 gives an interpretation of how this distribution changes with the threshold τ.

3.2.4.4 Intracluster densities

The intracluster density $\rho(C_k)$ of a given cluster C_k can be easily computed from the mean M_{C_k} and standard deviation SD_{C_k} of the weights of arcs inside cluster C_k.

$$\rho(C_k) = \frac{M_{C_k}}{1 + SD_{C_k}} \tag{3.6}$$

Let us call N_{C_k} the number of nodes in cluster C_k. An alternative formula for intracluster density is

$$\rho(C_k) = \frac{1}{N_{C_k} \times (N_{C_k} - 1)} \sum_{(i,j) \in C_k} w_{i,j} \tag{3.7}$$

Formula (3.6) will assign a greater intracluster density to clusters of homogeneously high weight. Formula (3.7) was designed to take care of situations described in Figure 3.9. In the case presented, using formula (3.6) we would get $\rho(A) > \rho(B)$; to put it simply, formula (3.6) favors homogeneity over connectivity. Therefore we introduced formula (3.7) for cases when connectivity fits more closely to the application needs. It is very important to notice that in Equation (3.7) the denominator $N_{C_k} \times (N_{C_k} - 1)$ corresponds

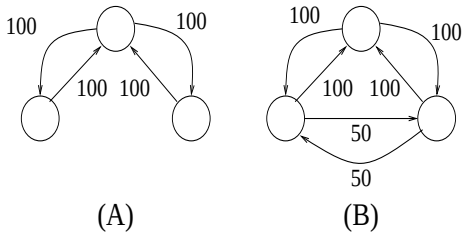

Figure 3.9 Homogeneous vs. tightly connected clusters

to the maximum number of arcs a cluster of N_{C_k} nodes can have (that is $2 \times \binom{N_{C_k}}{2}$). This makes the assumption that the *best* clusters have to be *fully connected* (cliques). We will see the limitations of this assumption in the section dedicated to metrics evaluation (see Section 3.3, page 48).

3.2.4.5 Intracluster density reduction measure

For a given value of the threshold τ, sLoc outputs a set of clusters $C_1, C_2 \ldots C_n$ and their respective intracluster densities $\rho(C_1), \rho(C_2) \ldots \rho(C_n)$. In order to compare the clusterings performed for each value of τ we need a reduction measure that will summarize the distribution of intracluster densities in one single figure. Therefore, we define ρ, the reduced intracluster density for a given value of τ, with the following formula:

$$\rho = \frac{1}{c \times N} \sum_{k \in [1,n]} N_{C_k} \times \rho(C_k) \tag{3.8}$$

In the formula above, from formula (3.5) we know that $c \times N$ is the total number of nodes that belong to a cluster of size larger that s. Thus, the reduction measure ρ is simply a weighted average of the intracluster densities where the weight assigned to a cluster is proportional to its size N_{C_k}. The use of this particular reduction measure can be justified by the fact that, assuming all the concepts have the same probability of being searched for, the probability of returning a given cluster as a response to a query is proportional to the number of nodes this cluster contains.

3.2.4.6 Intercluster density

In order to compute the intercluster density P, let $\mathcal{A}_{inter} \subset \mathcal{A}$ be the subset of $\mathcal{A}$ such that

$$\mathcal{A}_{inter} = \{a_{i,j} \in \mathcal{A} \mid i \in C_p, j \in C_q, p \neq q\}$$

The set $\mathcal{A}_{inter}$ contains all the arcs going from one cluster to another. What we call A_{inter} is the number of elements in $\mathcal{A}_{inter}$. Then M_{inter} and SD_{inter} are

$$M_{inter} = \frac{1}{A_{inter}} \sum_{a_{i,j} \in \mathcal{A}_{inter}} w_{i,j}$$

$$SD_{inter} = \sqrt{\frac{1}{A_{inter} - 1} \sum_{a_{i,j} \in \mathcal{A}_{inter}} (w_{i,j} - M_{inter})^2}$$

A first definition for the intercluster density P is

$$P = \frac{M_{inter}}{(1 + SD_{inter})} \tag{3.9}$$

An alternative definition can be

$$P = \frac{1}{n \times (n-1)} \sum_{a_{i,j} \in \mathcal{A}_{inter}} w_{i,j} \tag{3.10}$$

As with (3.6) and (3.7), (3.9) and (3.10) differ by the patterns they emphasize in the clustering. Formula (3.9) will assign a greater intercluster density when the arcs in $\mathcal{A}_{inter}$ are of similar weight, whereas (3.10) will emphasize the connectivity pattern of the clustering.

Here, notice the denominator in Equation 3.10, $n \times (n-1)$, where n is the number of clusters after the end of phase 3 in Figure 3.4, page 34; it can be viewed as an assumption

made on the clustered graph structure. This measure makes much more sense if the set of clusters are *fully connected*, that is, if there the weights $w_{i,j}$ are distributed fairly homogeneously between clusters pairs.

3.2.4.7 A possible definition of the optimum

The intracluster density measures the similarity of concepts within a cluster. As a result, it is our premise that reduced intracluster density ρ should be maximum at the optimum. At the same time, the intercluster density P measures how similar concepts are across different clusters. Therefore, we want the intercluster density to be minimized at this same optimum. At the same time, the closer the coverage c is to 100%, the larger the number of concepts represented in the knowledge neighborhood. These three metrics vary in completely different ranges; to make them comparable we have them all range between zero and one by simply dividing them by the largest value they actually take across the range of alpha. We then define standardized reduced intracluster density ρ_0 and intercluster density P_0:

$$\rho_0(\tau) = \frac{\rho(\tau)}{Max_{\tau \in \mathcal{T}}(\rho(\tau))} \tag{3.11}$$

$$P_0(\tau) = \frac{P(\tau)}{Max_{\tau \in \mathcal{T}}(P(\tau))} \tag{3.12}$$

A satisfactory clustering will have a ρ_0 close to 1, a P_0 close to 0, that is, $(1 - P_0)$ close to one, while the coverage has to be as close to 1 as possible too. Therefore, we arrive at the empirical formula for the optimization function F:

$$\mathrm{F} = (\rho_0 + (1 - P_0)) \times c \tag{3.13}$$

The optimum τ makes F reach a maximum. The above formula aims at determining the clustering that yields an optimum balance between coverage, intra- and intercluster densities. Further experimental evidence is needed, however, on the impact on performance in application environments to justify the use of such an optimum. Figure 2.5 depicts one such environment in which such experiments can be conducted.

This definition of a theoretical optimum does not intend to actually fit every real-world application sLoc would be used for. It is rather intended to provide a theoretical baseline, a reference to which other optima can be compared. For instance, the "optimum" clusters may be too large to be properly managed by a dynamic query engine that matches a query vector to one or more clusters. In addition, if concepts inside a given cluster are to be used as suggestions for an interactive search refinement, clusters of size greater than 10 will not be very user-friendly.

The logical next step is thus to test the different criteria for clustering efficiency in real-world applications. Figure 2.5 depicts the mapping of a query vector to a hierarchical distributed dynamic index. As discussed in Section 2.2.6, mappings between regions of semantic locality provide the necessary links, or paths, between subtopic regions at higher and lower levels of the hierarchy. Chapter 4 will broach several issues related to performing an actual query search on an HDDI, using gold standard collections from the SMART project.

3.2.4.8 Comparison with hypergraph-based partitioning

In [34], Han, Karypis and Kumar use HMETIS, a multilevel hypergraph partitioning algorithm to cluster large data sets. Their approach has caught our attention because their purpose is very similar to ours, that is, to achieve scalable clustering.

Like most traditional graph partitioning algorithms, HMETIS requires the user to enter the number of clusters before starting. The bad clusters are then eliminated using a *fitness* function and a threshold. Let e be a set of vertices representing a hyperedge and C be a set of vertices representing a partition. The fitness function that measures the goodness of the partition C is defined as follows:

$$fitness(C) = \frac{\sum_{e \subseteq C} weight(e)}{\sum_{|e \cap C|>0} weight(e)} \tag{3.14}$$

The fitness function measures the ratio of weights of edges that are within the partition and weights of edges involving any vertex of this partition. The high fitness value suggests that the partition has more weights for edges connecting vertices within the partition. This measure actually incorporates both concepts of intra- and intercluster densities, or at least the weight-part of them. The numerator of the fitness function in Equation (3.14) is the sum of the weights (similarities) within a cluster; we find this same sum in sLoc's definition of the intracluster density function $\rho(C)$. The denominator is concerned with edge cuts, in this regard it is similar to sLoc's intercluster density.

Kumar et al. also use another criterion to filter out vertices that are not highly connected to the rest of the partition. The connectivity function of vertex v in C is defined as follows:

$$connectivity(C) = \frac{|\{e|e \subseteq C, v \in e\}|}{|\{e|e \subseteq C\}|} \tag{3.15}$$

The connectivity measures the percentage of edges that each vertex is associated with. A high connectivity value suggests that the vertex has many edges connecting good proportion of the vertices in the partition. Note that this measure assesses strictly *intracluster* connectivity.

It is very interesting to notice how Kumar at al. merge *inter- and intracluster densities* into one measure (fitness) whereas sLoc keeps them separate, and also how these sLoc metrics merge and balance *weights* and *connectivity* inside intra- and intercluster density, whereas Kumar et al. make a clear distinction between the two. Note that Kumar does not address the issue of intercluster connectivity.

Notice also that as intra- and intercluster densities, fitness and connectivity can be called micro-metrics. They give relevant information about the clustered graph structure, but they are very expensive to compute.

3.3 Experiments with Macro- and Micrometrics

A set of experiments on input knowledge bases of different sizes and content have been conducted. The parameter α was used as the X-axis variable[5] to plot several measures on the output knowledge neighborhood. We have noticed some general trends in the experiments we have carried out. In this section, rather than trying to be general, we will focus on particular examples for the reader to get a feel about what happens qualitatively and quantitatively in the knowledge neighborhood as we tune α. Because we believe they are more relevant to search and retrieval requirements, we picked the definitions of intra- and intercluster densities that favored connectivity over homogeneity, that is, we use in this section definitions (3.7), page 42, for intracluster density, and (3.10), page 44, for intercluster density. These formulae favor clusters with a high degree of connectivity (lots of arcs between concepts); our intuition is that connectivity is essential to semantic locality.

3.3.1 General observations

First, one of the major advantages of sLoc is that the algorithm execution time is linear in the maximum number of nodes/arcs in the input knowledge base $\text{Max}(N, A)$. The algorithm is *scalable* in this regard.

The performance of sLoc is closely dependent on how the weights and arcs are assigned when the input knowledge base is created. In kBase, the knowledge base creation algorithm, the distribution of weights is not uniform: about half of the weights are equal to

[5] See Equation (3.3), page 35, for a definition of α.

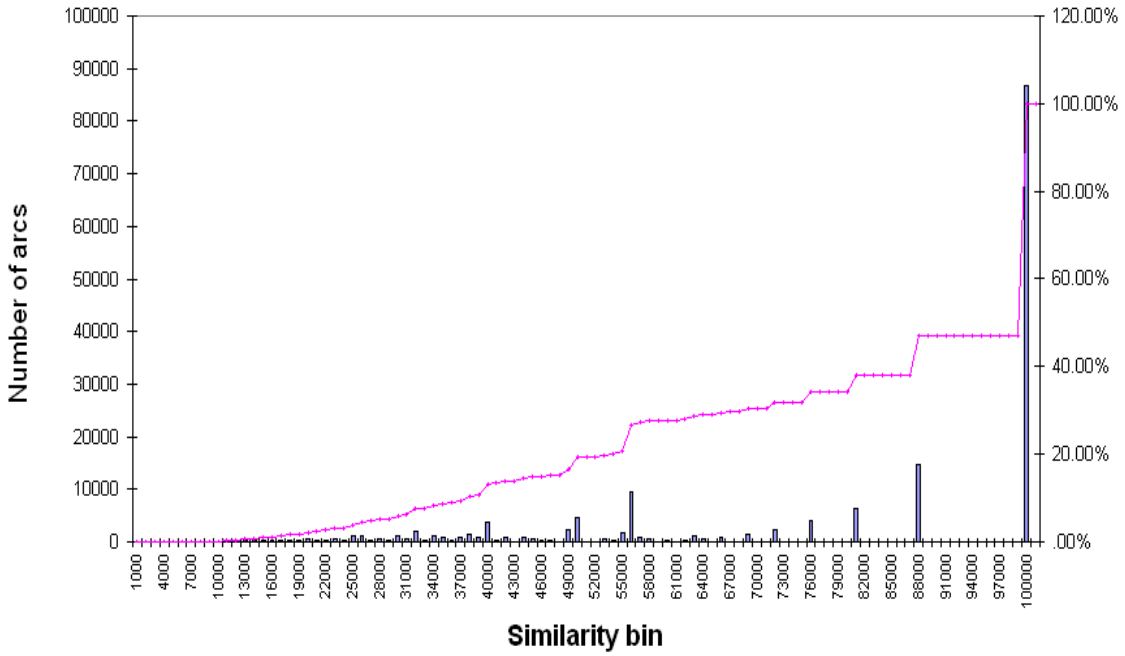

Figure 3.10 Distribution of arc weights at a leaf-level node (for Patterns 300)

the maximum weight for leaf-level HDDI nodes. The number of peaks in the distribution significantly impacts the number of resulting partitions.

The example in Figure 3.10 shows the distribution of arc weights for an input knowledge base created from 300 postings to a discussion list about computer software systems, the Patterns 300 collection. This exemplifies a *leaf-level* node weight distribution; in the next section I will take a closer look at the changes encountered when considering an interior node.

The high peak on the right-hand side of Figure 3.10 represents the number of arcs of weight close to Max($\mathcal{W}$). Actually, all these arcs have a weight strictly equal to Max($\mathcal{W}$). The rest of the graph is not uniform (e.g., some other peaks can be seen). We noticed that the number and location of these peaks in the graph determined different stages in the plot in Figure 3.11, which represents the total number of clusters versus the value of α at the end of the process in Figure 3.4, phase (2). Notice that at the end of phase (2) clusters of size smaller than s have not yet been discarded. When the threshold $\tau(\alpha)$

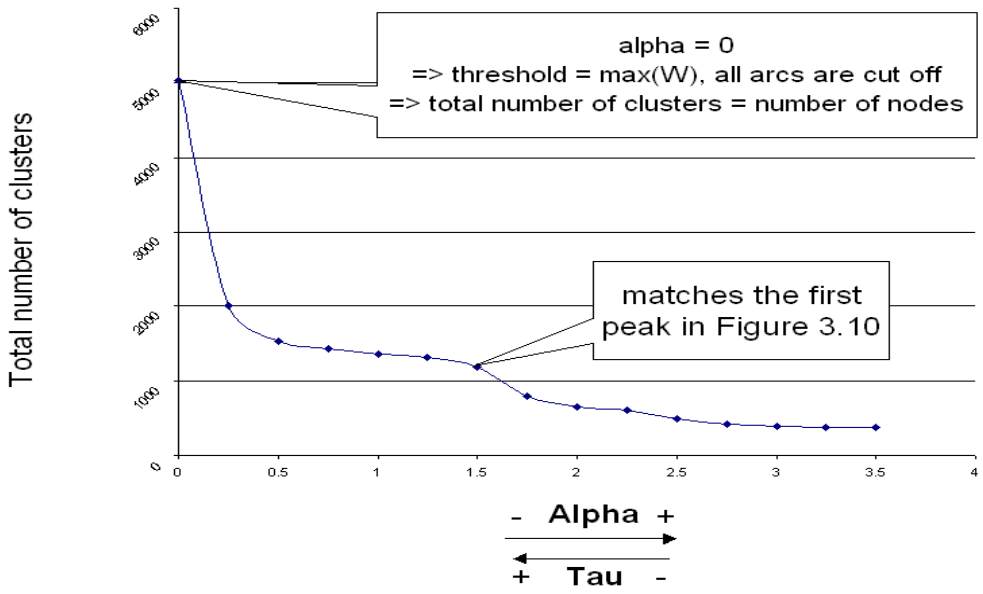

Figure 3.11 Clusters at the end of phase (2) vs. α

increases beyond one of the peaks of Figure 3.10, many arcs are cut and some clusters are broken into smaller ones. Note that the threshold $\tau(\alpha)$ increases as α *decreases*. In other words, in Figure 3.11 the pruning becomes less aggressive as α increases.

Figure 3.12 shows the actual number of clusters in the knowledge neighborhood at the end of the whole process, that is, after clusters of size smaller than s have been pruned out. To properly understand these results, an explanation of the process employed in computing the resulting knowledge neighborhood is required.

First, sLoc computes partitions of the graph for several values of α, and a minimum weight $s = 2$. A value of alpha $\alpha = .25$, for example, means all arcs of weight smaller than $\text{Max}(\mathcal{W}) - \frac{SD}{4}$ will be pruned from the graph in phase (1) of Figure 3.4. In particular, for $\alpha = 0$ all arcs in the graph are cut and every node is a cluster of size one. This is the most aggressive thresholding. The minimum weight s below which a cluster is considered an outlier is two. Thus no clusters are found for $\alpha = 0$. As α increases, multiple clusters of size one get connected and form clusters of size greater than one. This number of

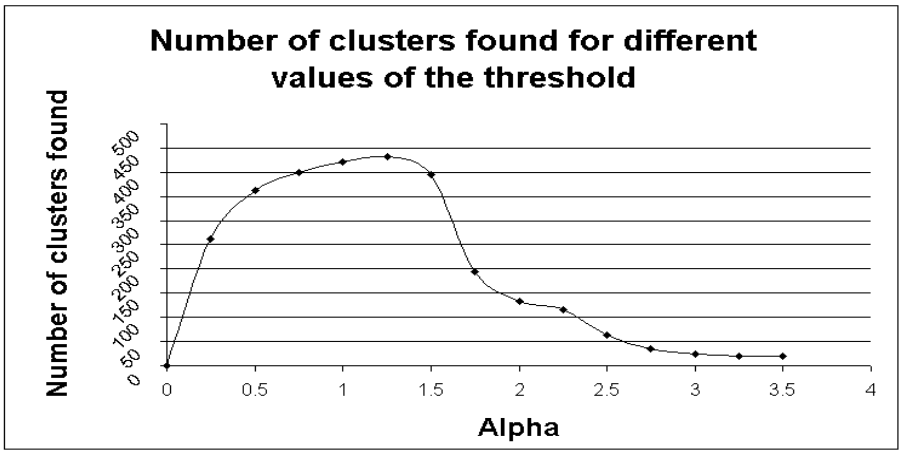

Figure 3.12 Number of clusters vs. α

clusters reaches a maximum for $\alpha = 1.25$ and then decreases. This maximum can be interpreted qualitatively as the point where

- isolated nodes can no longer join together to form larger clusters and
- clusters of size greater than one start merging into larger clusters.

As α continues to increase, the number of clusters decreases and a few clusters grow very large.

3.3.2 The use of micrometrics

Micrometrics give some feedback about the *internal structure* of the clustered graph, they measure how similar concepts are in a cluster, and how dissimilar concepts in different clusters are.

Reduced intracluster density for the Patterns 300 collection is represented in Figure 3.13. The drop in intracluster density around $\alpha = 1.7$ is due to the formation of large clusters, the aggregation of smaller clusters. These larger clusters are therefore made of a set of tightly connected clusters which are loosely connected together. This loose connection inside the large clusters causes their intracluster density to be very low.

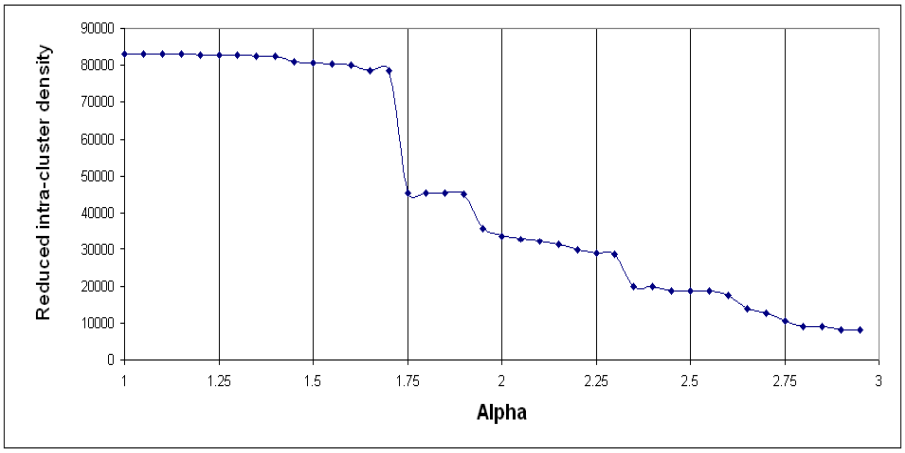

Figure 3.13 Intracluster densities for Patterns 300

Since the reduced intracluster density emphacizes more the intracluster density of large clusters, this reduced measure reflects faithfully the drop of the intracluster density of the newly formed large clusters.

Intercluster density, Figure 3.14, also reflects the formation of larger clusters. Even though the sum of the arc weights between clusters decreases when small clusters gather into one large entity, the intercluster density may increase. This is due to the drop in the total number of clusters (used in the denominator of the definition for intercluster density, Equation 3.10, page 3.10). When α increases from 1.6 to 1.8, the number of intercluster arcs decreases, due to the agglomeration of smaller clusters into larger ones. Therefore the sum of the intercluster similarities decreases as well. However, at the same time, the total number of clusters drops (see Figure 3.12, page 51). The denominator in Equation 3.10, page 44, is proportional to the total number of clusters squared; therefore, it drops very fast, actually faster than the sum of the intercluster similarities increases. Therefore intercluster density increases.

These two micrometrics give consistent results for interior nodes also. The way knowledge bases in interior nodes are computed is still the subject of extensive research, although as a first cut, we build higher level nodes by pruning out concepts based on their *item frequencies*, that is, the number of documents that contain the particular noun

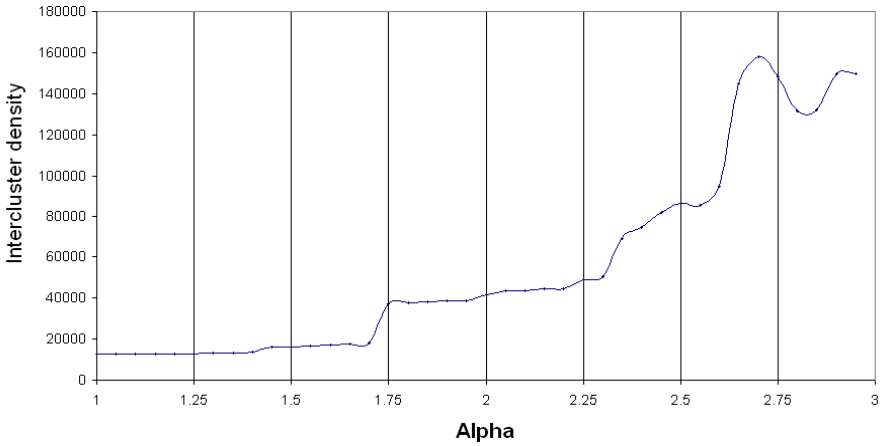

Figure 3.14 Intercluster densities for Patterns 300

phrase. We make the simple assumption here that relatively good discriminators are low item-frequency concepts and, equivalently, that relatively poor discriminators are high item-frequency concepts.

Figures 3.15 and 3.16 show reduced intra- and intercluster density for an interior node made from the Patterns 300 collection. In this example, I pruned out all the noun phrases that appeared in 10 or fewer documents. Notice that the definition of τ here is different than in the leaf-level case, we had to take into account the various kinds of distributions (normal distribution for interior nodes, see Figure 3.17). See Equation 3.4, page 35 for the definition of τ in the case of an interior node.

3.3.3 The use of macrometrics

The use of macrometrics is less expensive and may approximate the results given by micrometrics. Basically, the main cause of drop in intracluster densities is due to the formation of large clusters by aggregation of smaller ones. This same process can be observed using macrometrics such as the number of clusters or the distribution of cluster sizes.

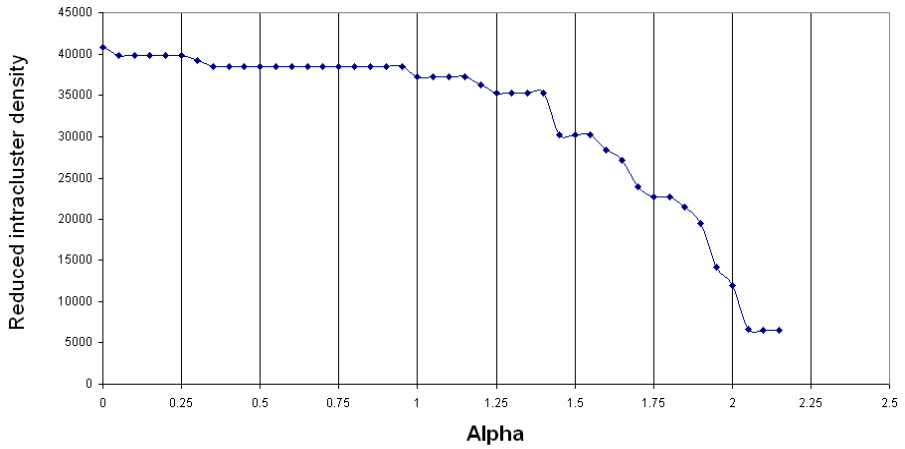

Figure 3.15 Reduced intracluster density for an interior HDDI node (from Patterns 300)

We note that the simple optimization function F', where intra- and intercluster densities are replaced by the number of clusters, may be an alternative to the optimizing function F (based on micrometrics), and F' is cheaper in terms of complexity:

$$\text{F'} = n \times c \tag{3.16}$$

3.3.4 Results for the optimization function F

Figure 3.18 shows how the optimum function varies for different values of α for a leaf-level HDDI node. In all the experiments we conducted, the curve has about the same shape, always exhibiting a maximum for a value of α ranging between 1.25 and 2.

For small values of α, $\rho_0 + (1 - P_0)$ is fairly large and is maximum when alpha is just above zero. Small clusters (size of 2 or 3) are more prone to be fully connected with arcs of maximum similarity than they would be for larger values of α (see Figure 3.19, page 57). Indeed, when α is just above zero, the only arcs that are not pruned out have a weight of $\text{Max}(\mathcal{W})$, this makes the intracluster densities very large. At the same time, intercluster densities are fairly low. This can be explained by the way weights are assigned; it is fairly uncommon to see $w_{i,j} = \text{Max}(\mathcal{W})$ and $w_{j,i} < \text{Max}(\mathcal{W})$ at the

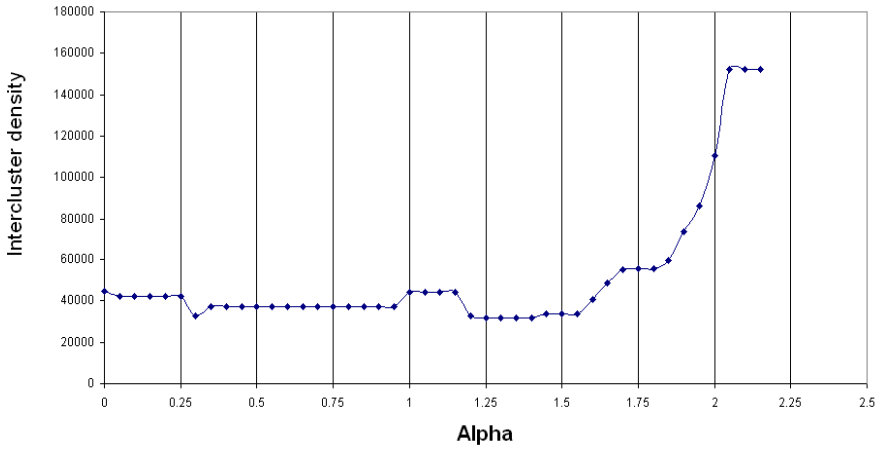

Figure 3.16 Intercluster density for an interior HDDI node (from Patterns 300)

same time. However, the coverage c is very low in this first part, that is, a relatively small proportion of the knowledge base is represented in the knowledge neighborhood. The coverage is therefore used as a *filter* when it multiplies $\rho_0 + (1 - P_0)$ to balance the impact of ρ_0 and P_0 on the optimization function F.

In Figure 3.20, one can notice that c has a logarithmic shape, reaching 85% for $\alpha = .6$, its filtering action on $\rho_0 + (1 - P_0)$ then lessens, so that intra- and intercluster densities really lead the way F varies.

For large values of α, that is, when the thresholding is not aggressive, some clusters start merging into a couple of very large ones. These very large clusters have very poor intracluster density because they are far from fully connected. At the same time the intercluster density is increasing. This is due in particular to two factors. First, the number of clusters is fairly low, and therefore in formula (3.10) the sum of the arc weights between clusters is not balanced by the denominator. Second, when $w_{i,j}$ is low, chances are higher that node j is not connected at all to node i, which increases the probability of this arc being an intercluster arc.

It is very interesting to see that the definition of F still works for interior HDDI nodes, despite the different structure and weight distribution of their knowledge bases. In Figure 3.21, page 58, F even exhibits a clearer peak than for leaf-level nodes.

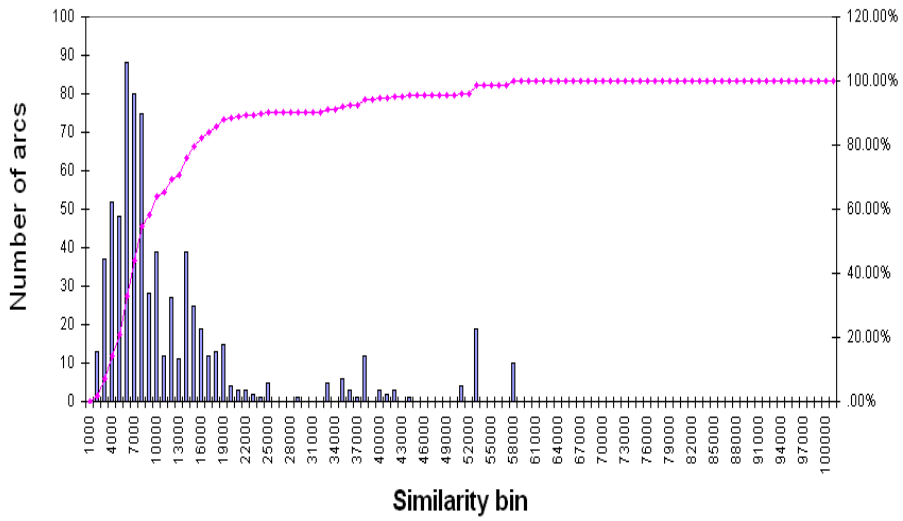

Figure 3.17 Weight distribution for an interior HDDI node (from Patterns 300)

3.4 Summary and Conclusion

In order to build an HDDI, we need to cluster the knowledge base graph of each HDDI node. After a thorough study of existing IR clustering techniques, we decided to design our own algorithm to cluster HDDI knowledge bases: sLoc. This algorithm meets the requirements of HDDI in terms of complexity; it is linear and therefore allows *dynamic* identification of regions of semantic locality.

In order to assess the quality of the clustering we designed micro- and macrometrics. We arrived at an optimization function F which uses a subset of these metrics to define a desirable clustering. Finally, we provided the reader with experiments and their interpretation to give a qualitative understanding of how these metrics and optimization function work.

Now, we need to assess these theoretical measures in a real-life application environment in order to validate our approach. Therefore, in Chapter 4 we test sLoc for a query search and retrieval application, using gold standard collections.

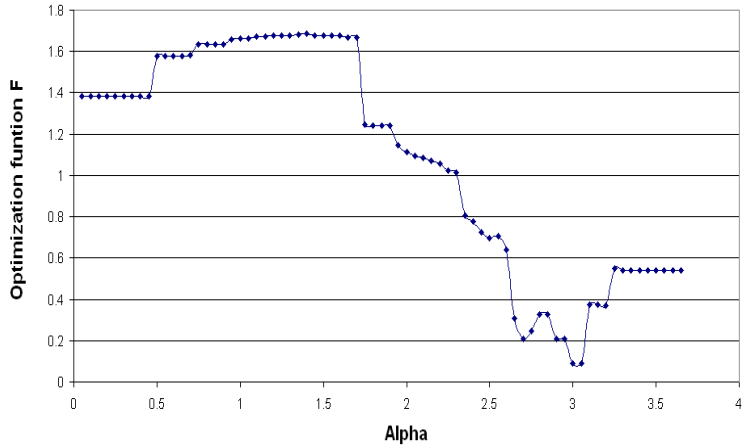

Figure 3.18 Optimization function F vs. α

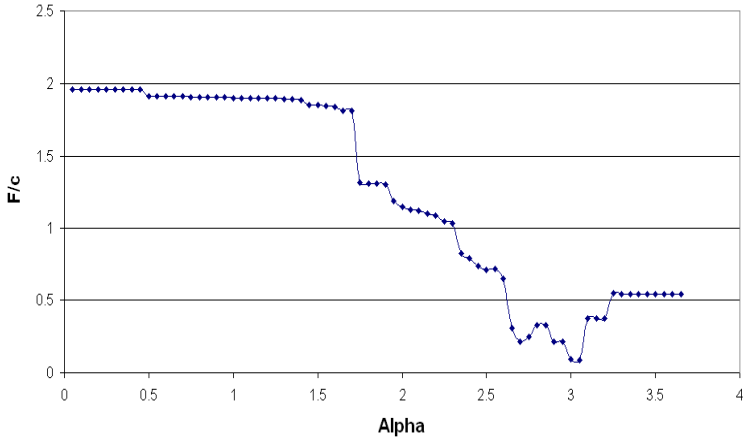

Figure 3.19 $\rho_0 + (1 - P_0)$ vs. α

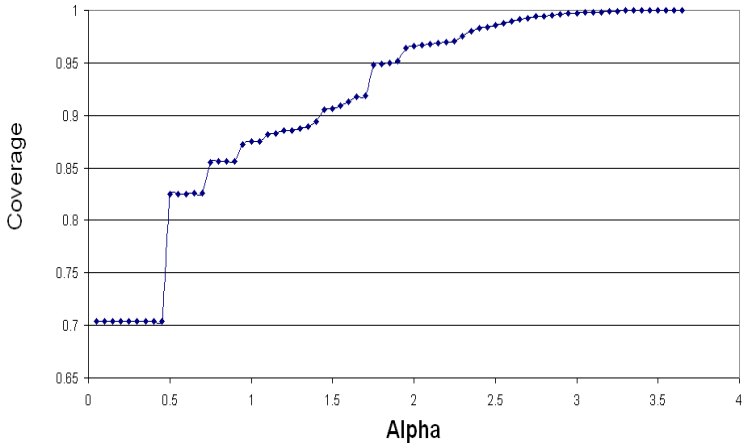

Figure 3.20 Coverage c vs. α

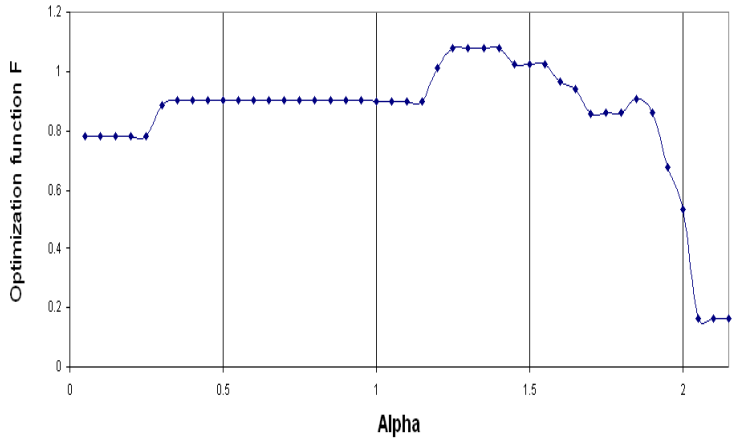

Figure 3.21 Optimization function F for an interior HDDI node (from Patterns 300)

CHAPTER 4

VALIDATION AND GOLD STANDARDS

In information retrieval, evaluation is often considered the most troublesome area of research. In order to validate our approach we looked at a proved method, using well-known gold standards.

4.1 The SMART Project and Gold Standards

SMART [20, 8] is originally an implementation of the vector-space model of information retrieval Salton suggested in the 1960s. The documentation coming with the software package outlines two main purposes for SMART:

- The primary goal of SMART is to provide a framework for information retrieval research. It provides standard versions of indexing, retrieval, and evaluation algorithms.

- A secondary goal is to have an actual small scale information retrieval system available to users. "Small scale" is loosely defined; people on the SMART project report having worked with was about 300 Mbytes of netnews.

The SMART system uses natural language queries, but also provides queries in Boolean syntax and just as a set of keywords.

We looked at the SMART project for two major reasons. First it is a project of reference in the field of IR. Second, it provides gold standard collections that come with a set of queries and the relevant set of documents for each query. A gold standard collection

is composed of a set of documents, a set of queries and the relevant documents associated with these queries. These collections are called *gold* standards because they include the collections, queries and proper responses chosen by domain experts. In addition, many such gold standard collections have been used by various research projects, thus making comparison between various techniques possible.

4.2 Building an HDDI from SMART Collections

In our effort to validate sLoc using SMART collections, we learned a lot, not only about HDDI, but also about the SMART gold standard queries and the assumptions they make implicitly.

The validation process has served as a ground-breaking step too, because, as expected, a query search and retrieval mechanism would not work without building at least a simple HDDI. We thus had to investigate new issues such as hierarchy building and search strategies.

Therefore this chapter is also intended to give the reader a feel about the issues in building an HDDI using hands-on experiments with ADI, MED and CISI, three SMART gold standard collections.

4.2.1 Study of the gold standard SMART queries

The three collections – ADI, CISI and MED – come with a set of queries and for each query the set of relevant items. Because we want to achieve high precision and recall, we investigated the structure of these queries.

At this point let us introduce the term "item frequency." The item frequency of a concept is the number of items (documents) it occurs in, regardless of how many times it actually occurs in each item. Let us characterize the queries for each collection in terms of the distribution of their item frequencies. More specifically, let us, for each collection, compare the item frequency distribution in the queries with the item frequency distribution across the documents.

In order to keep it simple and general I used only three different bins. The first one gathers all the terms of item frequency 1, the next gathers the terms of item frequency ranging between 2 and 20 and the last between 21 and the maximum item frequency. I chose 20 arbitrarily; it turns out to be a convenient median of the distribution in some cases, but I could as well have chosen some other figure here.

4.2.1.1 ADI: Collection and queries

ADI is the smallest collection of the three; it is only made of 82 documents. Figure 4.1 shows that 87.1% of the queries are terms that occur in 1 to 20 documents. Of these, only three query terms (9.68%) occur in only one document of the collection. Now Figure 4.2 represents the item frequency distribution of *all* the terms in the collection (not only those of the queries). Terms of item frequency 1 represent 69.25% of all the terms in the documents.

Two important conclusions have to be stated here. First, the queries given in SMART do not reflect the item frequency distribution in the collection; queries are mostly made of terms of item frequency greater than one. This makes the *assumption* that users search using relatively *poor discriminators* (terms occurring in many documents). Second, given the query item frequency distribution Figure 4.1, in order for HDDI to match these queries and get good precision and recall, it must be able to match both the queries' item frequency distribution at one or more levels, and it must match the document set item frequency distribution at the leaf level.

4.2.1.2 CISI and MED: Collection and queries

The same remarks apply here, too. CISI is a larger collection (913 documents), but the same patterns as in ADI emerge in query item frequency distribution and document set item frequency distribution. In Figures 4.3 and 4.4, note that while 46% of the terms in the document set appear in one document only – 146 of them, that is, only 4.3% of the 3367 terms – are represented in the queries.

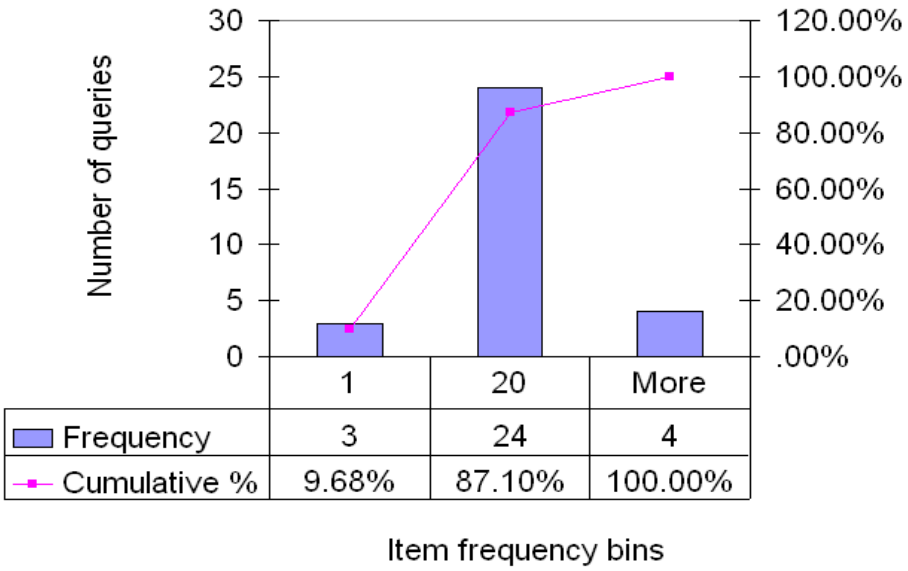

Figure 4.1 Item frequency distribution of ADI queries

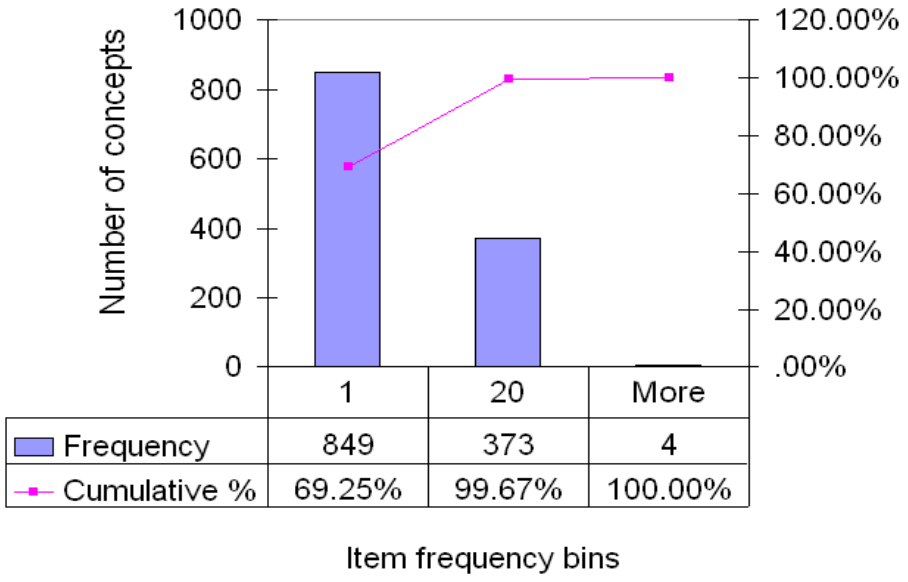

Figure 4.2 Item frequency distribution of ADI document set

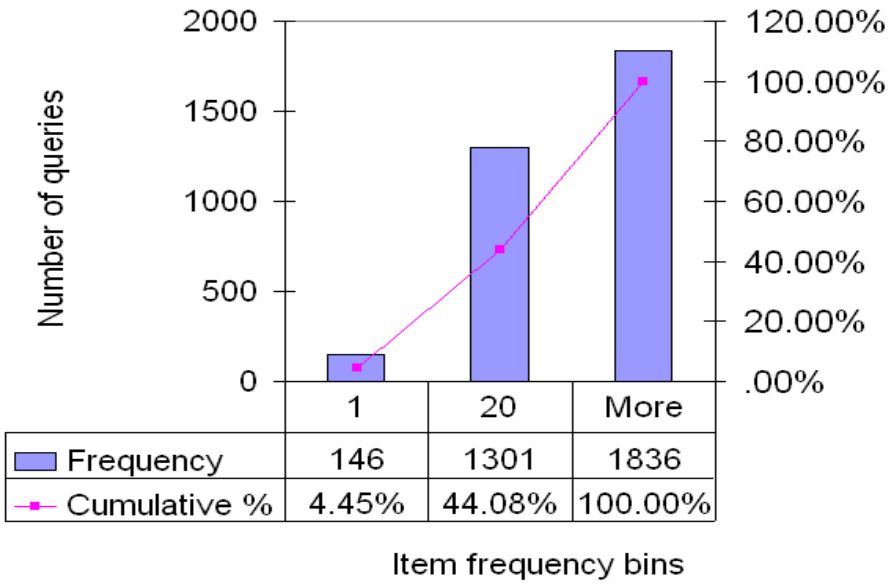

Figure 4.3 Item frequency distribution of CISI queries

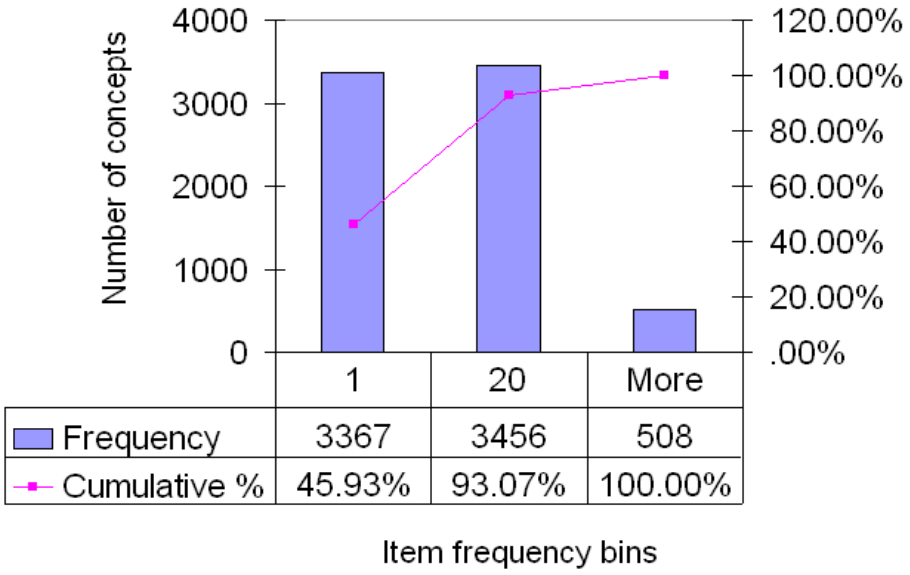

Figure 4.4 Item frequency distribution of CISI document set

Note also that while terms of item frequency greater than 20 represent only 16.3% of the CISI collection, they make 55.2% of the query terms. This additional imbalance corroborates our claim that queries for the three gold standard collections are mostly made out of relatively poor discriminators.

The distributions in MED are very close to those in CISI. Therefore the same conclusions apply (see Figures 4.5 and 4.6).

4.2.1.3 Conclusions

In the previous sections, the three SMART gold standard collections have been studied. Now, let me state clearly these major conclusions, which will be our guidelines in building the HDDIs:

(1) Query sets in our gold standard collections and document sets have different distributions. While the document set is half made of relatively good discriminators (e.g., of item frequency 1), SMART queries assume the user searches using relatively poor discriminators.

(2) To perform the query search and retrieval, queries and documents are mapped onto the knowledge neighborhoods. In order to ensure maximum precision and recall, these knowledge neighborhoods must match both query and document set distributions. This motivates the need for a hierarchy of knowledge neighborhoods of decreasing resolution.

4.2.2 Study of the leaf-level HDDI nodes

Because an HDDI is built from the bottom up, our first focus is the leaf level, that is, the HDDI node directly made from the actual text collections. In order to stick to the SMART model we used their concept extraction algorithm; the concepts in the HDDI we built are then (single-word) terms rather than noun phrases, because SMART, as most of the traditional IR systems, uses (single-word) terms as keywords.

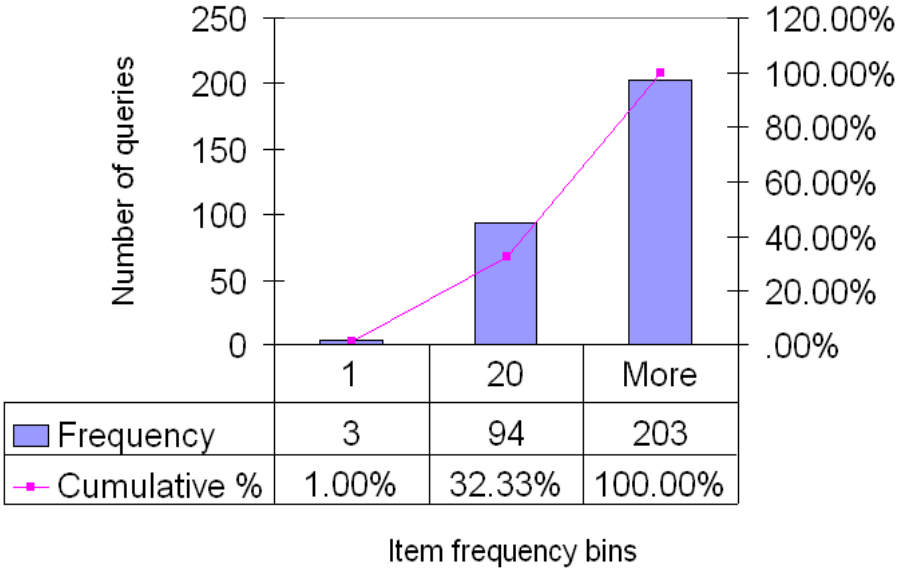

Figure 4.5 Item frequency distribution of MED queries

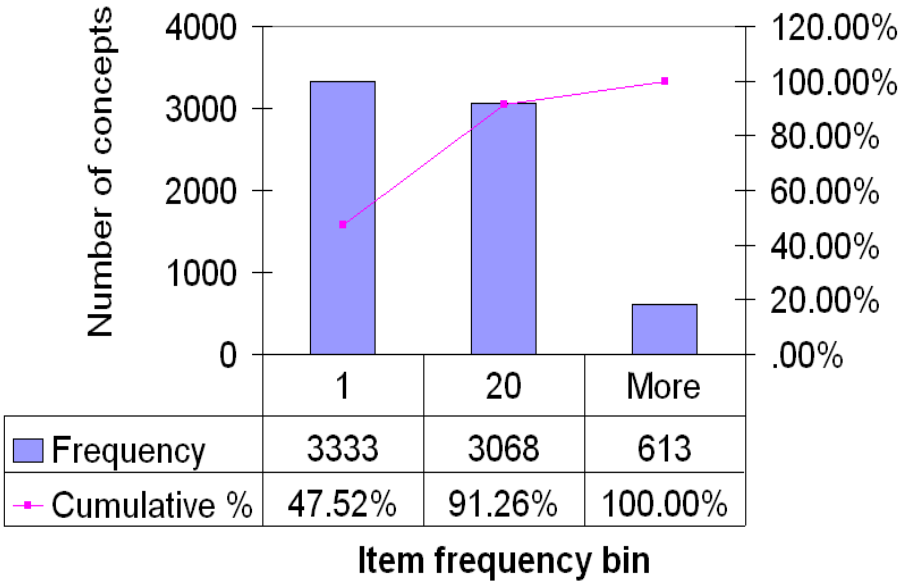

Figure 4.6 Item frequency distribution of MED document set

In order to understand what is going on when the clustering is performed, we need to analyze the item frequency distribution in the SMART collections. Figures 4.2, 4.4 and 4.6 show the proportion of terms of item frequency 1, compared to other item frequencies. This basically means that in all three collections more than 45% of the terms occur in only one document across the whole collection.

These terms of item frequency 1 will be identified as good discriminators by kBase, the algorithm that creates the knowledge base in an HDDI node – good because they discriminate *clearly* between documents in the input set.

A poor discriminator is a term that can be found in a large percentage of the entire collection. The algorithm kBase assigns high weights to arcs connecting pairs of a good discriminators.

When sLoc identifies the regions of semantic locality, it filters out relatively poor discriminators and forms clusters of good discriminators. Now, what happens when more than 45% of the terms have an item frequency of 1? The meaning of such an unbalanced distribution is that 45% of the terms are excellent discriminators. Consequently, sLoc builds a knowledge neighborhood formed of terms of item frequency 1 for the most part, filtering out almost all the terms of greater item frequency. Table 4.1 shows the percentage of terms of item frequency 1 in the knowledge neighborhoods built by sLoc for each of the three SMART collections. The rest of the terms in the knowledge neighborhood have an item frequency close to 1.

The bottom line of the study at the HDDI leaf level is that the knowledge neighborhoods contain only excellent discriminators. This of course is dependent on the algorithm used to compute the similarities in the knowledge base, kBase, but it is our observation that all the indexing algorithms use some sort of band-pass filter effect to focus only on a certain quality of discriminators. Therefore, changing kBase for another algorithm would not change the need for a hierarchy. As we saw earlier, the queries proposed by SMART use very few high quality discriminators; therefore, we see the need for a higher level in the hierarchy, an *interior node* with a lower resolution. In other words, we need

Table 4.1 Terms of item frequency 1 in the knowledge neighborhoods at a leaf node

Collection	Percentage of terms of Item Frequency 1
ADI	91.00
CISI	85.10
MED	86.20

to build a hierarchy in order to match the varying precision of the queries. The next section studies interior HDDI nodes built from ADI, CISI and MED.

4.2.3 Study of interior HDDI nodes

An interior HDDI node is loosely defined as a node made from combination and pruning of lower level HDDI nodes. It is of lower resolution than the child-nodes it is made from.

In order to achieve lower resolution, one simple method is to select the higher item frequency terms of the leaf-level nodes to make a higher level (interior) node. In the present experiment we actually use only one leaf-level node to make a higher level node, whereas in the final version of HDDI, multiple distributed leaf-nodes will be combined into higher level nodes. We prune out all the terms of item frequency smaller than or equal to a certain threshold in the collection. We then run kBase and sLoc on the pruned collection and get the corresponding interior node and its knowledge neighborhood.

We arbitrarily chose 20 as a threshold for MED and CISI. ADI, however, only contains 82 documents, so pruning terms of item frequency 20 was too aggressive, and we only pruned out terms of item frequency 1.

4.2.3.1 CISI: Collection and queries

Figure 4.7 shows the item frequency distribution in the knowledge neighborhood for an interior node made from the CISI collection. It differs greatly from the kind of

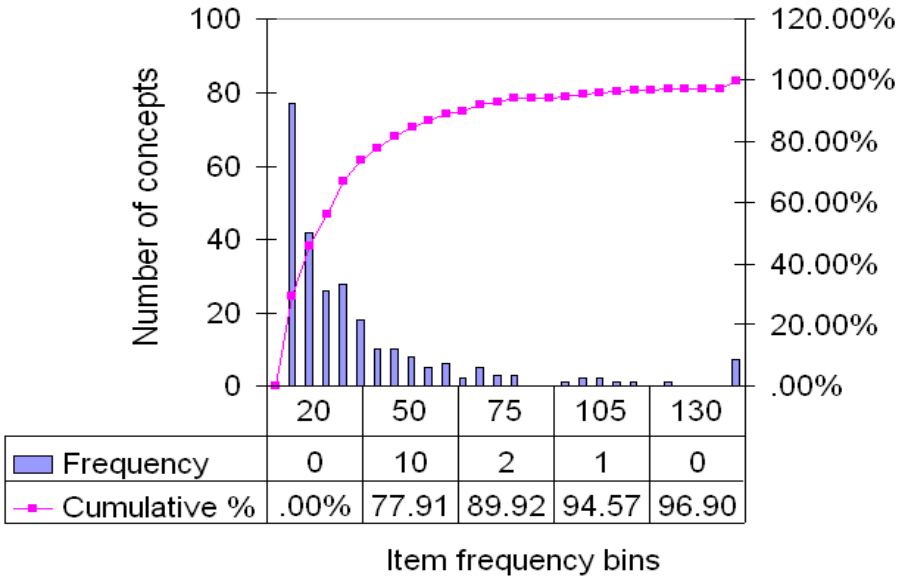

Figure 4.7 Interior node item frequency distribution for CISI

distribution we get for leaf-level HDDI nodes, where about 90% of the concepts in a node have item frequency 1. Here of course no term in the knowledge neighborhood can have an item frequency lower than the threshold; that is, all the terms in the knowledge neighborhood occur in 21 or more documents in the collection. Moreover, the item frequencies represented in the knowledge neighborhood cover a wide range of values. The best discriminators (lowest item frequency terms) are more numerous, but very poor discriminators can also be found.

4.2.3.2 MED: Collection and queries

Figure 4.8 shows the item frequency distribution in the knowledge neighborhood for an interior node made from the MED collection. The distribution is similar to that of CISI in many respects. This is another example for which this simple approach, in which hierarchy building is based on item frequencies, seems to work. Theoretically, matching queries against these kinds of knowledge neighborhoods can lead to good precision and recall. Multiple levels may be needed, however, because terms of item frequency smaller

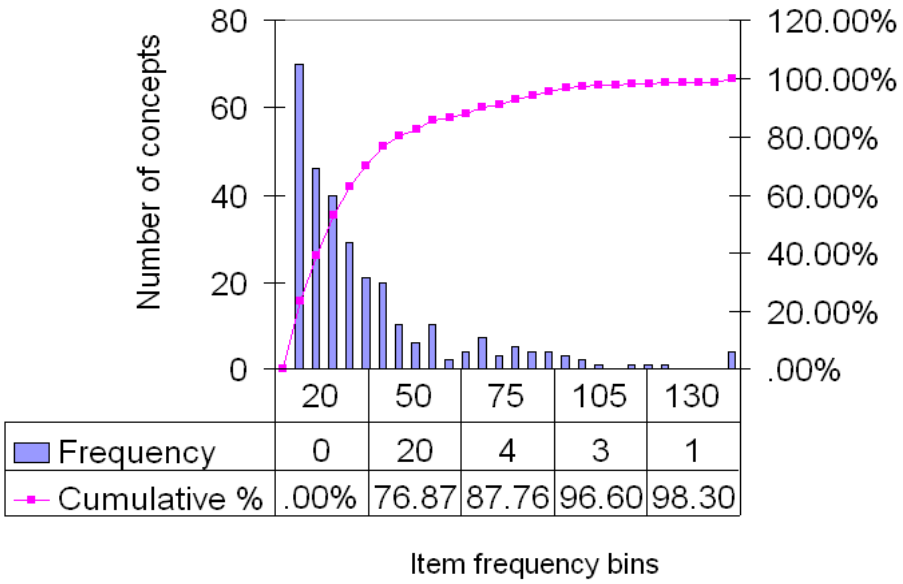

Figure 4.8 Interior node item frequency distribution for MED

than 20 are not represented here; if such terms are part of the query, only knowledge neighborhoods with less aggressive pruning will match these terms.

4.2.3.3 ADI: Collection and queries

ADI is a good example of when this simple approach does not lead to interesting results. Indeed, the terms of item frequency 1 as we saw in Figure 4.2 page 62, represent almost 70% of the collection. Figure 4.10 is a finer breakdown of the item frequency distribution in the document set. Terms of item frequency 2 represent another 15% of the collection. Therefore when we prune out terms of item frequency 1 (see Figure 4.9, page 70), terms of item frequency 2 represent roughly half of the new distribution and we get a knowledge neighborhood of almost only terms of item frequency 2.

4.2.4 Study of a sample hierarchy

Based on the analysis above, we know that different levels of the hierarchy actually represent different resolutions from the standpoint of query search and retrieval. The

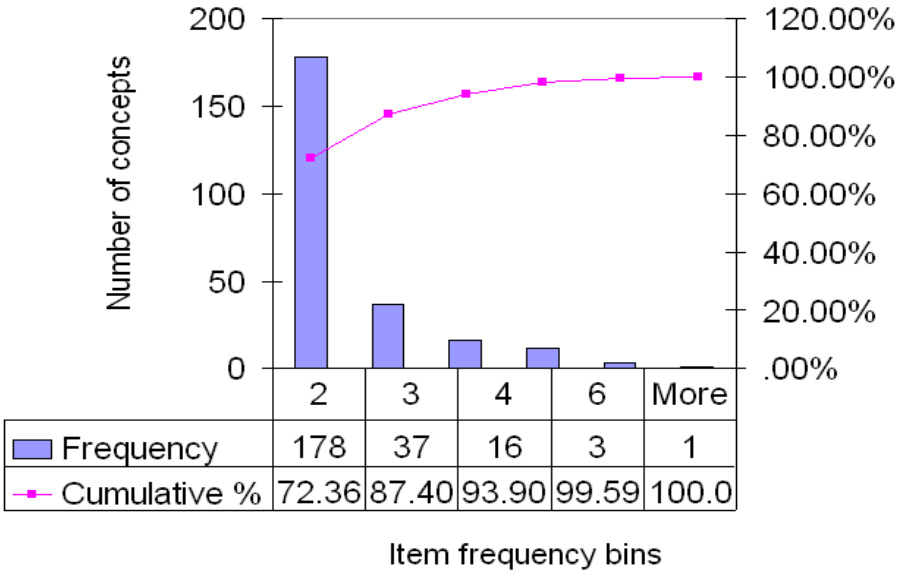

Figure 4.9 Interior node item frequency distribution for ADI

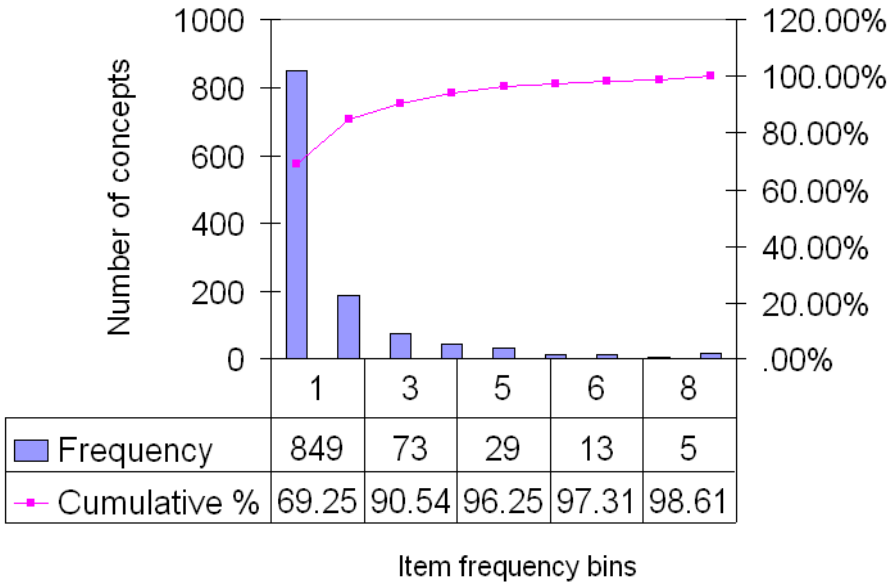

Figure 4.10 Item frequency distribution of ADI document set

Table 4.2 Average percentage of a query matching at least one level of the HDDI

SMART Collection	Number of queries	Average number of query terms per query	Average percentage of matching query terms per query
ADI	35	6.1	82.8%
CISI	112	29.3	91.2%
MED	30	10	90.6%

higher the level in the hierarchy the lower the resolution. We claim that these different levels can be used to match the various levels of resolution within a single query.

In order to validate this claim, we built the 20 first levels of the hierarchy, that is, for a threshold value ranging from 0 (leaf-level node) to 19, we pruned out terms of item frequency smaller than the threshold and computed the corresponding knowledge neighborhood. Then, for each query we looked up the number of query terms that actually matched some cluster at some level of the hierarchy. Query terms that are less precise matched the higher levels, while higher precision terms matched lower levels. For each of the three collections, Table 4.2 shows the average percentage of terms in a query that matched at some level of the hierarchy. The results of Table 4.2 lend credence to the hierarchical approach; the table shows that using a simple HDDI with 20 levels[1] suffices to match 90% of a given SMART query for the MED and CISI collections. The match with ADI is slightly less because ADI is smaller in size. It may be that the approach consisting of pruning based on item frequencies is too coarse here.

4.3 Query Search and Retrieval Strategies

So far we have seen how different levels in the hierarchy are built and what their characteristics are. Now the question is: What strategy for query search and retrieval will best leverage this hierarchy?

[1]Note that the number of levels is bounded only by the maximum item frequency; we chose 20 arbitrarily.

The initial idea in HDDI is to find the "root node" where the query first matches a knowledge neighborhood and then, using a probabilistic mapping between HDDI nodes (i.e., between knowledge neighborhoods of different levels), going deeper into the HDDI tree down to the leaf-nodes, then retrieving the relevant items from the leaf nodes. This process is illustrated in Figure 2.5, page 15.

The first issue here is how to effectively map one level with another. Our original idea is to build a probability matrix M where $M(i, j)$ represents the similarity between the region of semantic locality i of the higher level node and the region of semantic locality j of the lower level node. This similarity measure between two regions of semantic locality is based on the cosine measure of similarity filtered by the topology of the knowledge neighborhood [40].

This intuitive definition of similarity between regions of semantic locality is not always successful in mapping one level of the hierarchy to another. In particular from the leaf node to the first interior node, we saw (see Table 4.1, page 67) that the knowledge neighborhoods at leaf-level nodes were almost entirely made of terms of item frequency 1. Those same terms cannot be present in interior nodes if the hierarchy build is based on item-frequency pruning.

In order to mitigate this effect, we conducted our experiments with a different definition of the leaf-node: only terms of item frequency 1 or above are considered when the knowledge base and neighborhood are computed. That is, we "skip" the first level of the hierarchy. This can be justified by the similar approach latent semantic indexing (LSI) takes in [18]: only terms occurring in two documents or more are represented in the term to document matrix.

4.3.1 Mixed transversal and vertical search strategy

Figure 4.11 represents the first search heuristic we tried. Basically, for a given query we look up the matching clusters at each level of the HDDI. In our experiments we considered that a cluster actually matches a query when their probability of overlap was greater than 10%.

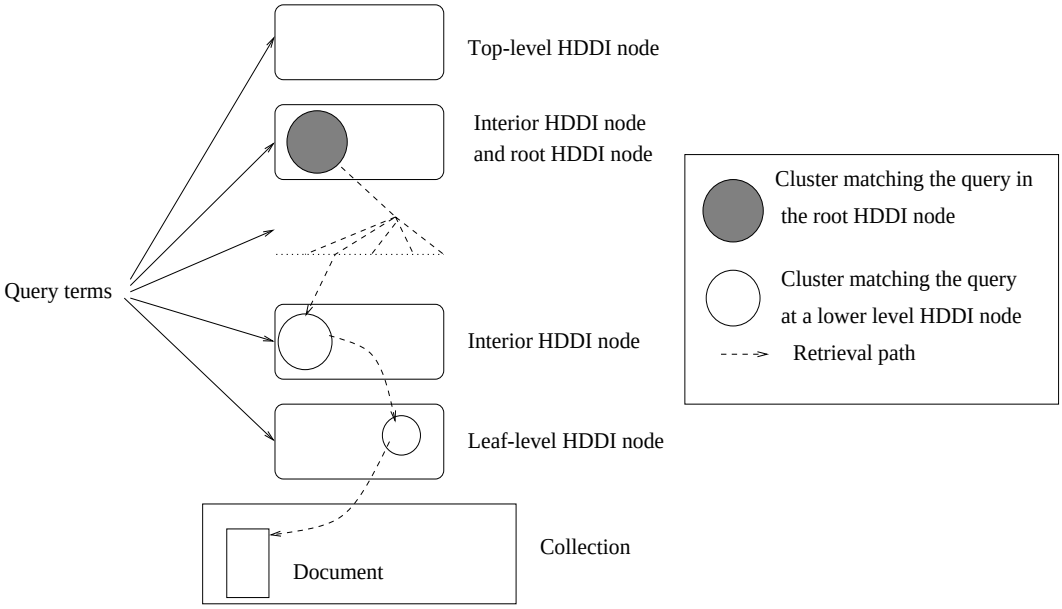

Figure 4.11 Mixed transversal and vertical search strategy

Then, we find the root node, the highest node in the hierarchy in which at least one region of semantic locality matches the query. We then use the recursive algorithm in Figure 4.12 to search the whole HDDI tree from top down. At each level of the HDDI we determine the clusters that will lead to the most relevant documents. Such clusters are clusters that match the query and that have a nonzero overlap probability with the cluster considered at the upper level. If such clusters cannot be found then we explore all the clusters that have a nonzero overlap probability with the cluster considered at the upper level. Eventually, zero, one or more documents are returned with an associated measure corresponding to the product of the probabilities of overlap between the clusters visited on the way from the root node down to the collection. This measure gives an idea of how relevant a document is with respect to the query used.

In this experiment, we did not tune all the parameters to get the best precision and recall. However, we now have a good understanding of the pros and cons of this method. One interesting aspect of this method is that it actually takes advantage of the extensive

```
foreach cluster in (clusters that match the query at the root node) {
    doMatch(cluster, root_node - 1, 1) ;
}
doMatch(upperLevelCluster, level, probability){
 if (level>0){
    foreach cluster (cluster that match the query at
                     this level of the HDDI){
        p=probability of overlap between upperLevelCluster and cluster;
        if (p>0) doMatch(cluster,level-1,probability*p);
    }

    if (no cluster matches the query at this level
        OR all the probabilities of overlap p are zero){
        foreach c  (nonzero probability of overlap p between
                    upperLevelCluster and cluster c at this level){
            doMatch(c,level-1,p);
    }
 }
 else { // level = 0 (collection)
    print each document which probability of overlap with
    upperLevelCluster is nonzero.
 }
}
```

Figure 4.12 Algorithm for the mixed transversal/vertical search strategy

overlap between the hierarchy of knowledge neighborhoods and the query set. We saw (Table 4.2, page 71) that a given query will match at many levels of the HDDI.

4.3.2 Conclusion

This chapter showed a real-life application of sLoc and HDDI. Through the use of SMART gold standard collections ADI, MED and CISI we had the opportunity to analyze the typical distributions of concepts in standard text collections and standard query sets. The interpretation of these distributions corroborates the need for a hierarchical classification of knowledge.

We built HDDIs for ADI, MED and CISI and verified that on average more than 80% of the query terms in a query mapped to one or more level of the HDDI. These results (see Table 4.2, page 71) lend credence to the hierarchical approach. Using a simple HDDI with 20 levels suffices to match 90% of a given SMART query for the MED and CISI gold standard collections and 80% for ADI.

Finally, we designed a simple search strategy to map SMART queries onto an HDDI. This first-pass strategy helped us identify some of the key factors of a good retrieval strategy using HDDI.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

The HDDI project started less than a year ago now. A lot has already been achieved, but the project is still in its infancy. The main purpose of this thesis was to give an overview of the tools, measures and issues related to sLoc and HDDI.

This thesis summarized my work on the HDDI project. My main focus has been sLoc, the algorithm we use in HDDI to identify the regions of semantic locality, but I also brought my contribution to the conception of the HDDI architecture. In this chapter, I will identify our key discoveries with sLoc and HDDI and give guidelines for future work.

5.1 Regions of Semantic Locality and sLoc

In order to build an HDDI, we need to cluster the knowledge base graph of each HDDI node. After a thorough study of existing IR clustering techniques, we decided to design our own algorithm to cluster HDDI knowledge bases based on *contextual transitivity*: sLoc. This algorithm meets the requirements of HDDI in terms of complexity; it is linear and therefore allows *dynamic* identification of regions of semantic locality.

Note that other IR techniques may also use an approach akin to contextual transitivity. In particular, latent semantic indexing (LSI) is based on a similar principle [18]. In LSI, the term-to-document matrix is modified using a truncated singular value decomposition (SVD). The truncated SVD creates a new space of lower dimension. The use of

this new space enables the discovery of hidden relationships between concepts. Quoting Susan Dumais [18]:

> The approach is to take advantage of implicit higher-order structure in the association of terms with documents ("semantic structure") in order to improve the detection of relevant documents on the basis of terms found in queries.

We have the intuition that what Dumais refers to as taking advantage of "implicit higher-order structure in the association of terms with documents" is actually akin to using some sort of contextual transitivity to discover "latent" relationships between terms. Further mathematical research is needed to determine how LSI and contextual transitivity are related.

In order to assess the quality of the clustering, we designed micro- and macrometrics. We arrived at an optimization function F which uses a subset of these metrics to define a desirable clustering. We provided the reader with experiments and their interpretation to give a qualitative understanding of how these metrics and optimization function work.

In Chapter 4, we saw how, using sLoc, we could get a knowledge neighborhood at each level of the HDDI. Using the first 20 levels of the HDDI, the corresponding knowledge neighborhoods identified by sLoc successfully matched about 90% of the terms of standard queries for MED, CISI and ADI.

Now, open issues are still to be addressed in sLoc. More specifically, some parameters in sLoc have been set empirically, and research can be done to determine more accurately how they can influence the results.

For example, s, the minimum number of concepts for a region of semantic locality to be valid, has been set to two. That is, we made the assumption that the final application of sLoc and HDDI values regions of semantic locality even if they contain only two concepts. Depending on the context and on the particular application, s may have to take a larger value.

Another issue is how to evaluate the performance of F, the optimization function used in sLoc to determine how aggressive the thresholding must be. This performance

definitely depends on the type of application. In the case of query search and retrieval, we got good results in terms of coverage (about 90% of the terms of standard queries for MED, CISI and ADI), but using our mixed transversal and vertical search strategy, one issue that arose was the presence of large regions of semantic locality. The bottom line is that tuning of the optimization function F has to be done as part of the whole query strategy. That is why we focused on defining metrics and explaining how to make use of them, rather than spending time tuning F.

Yet another issue with sLoc is how it will scale to cases where multiple repositories are combined into one single knowledge base. In other words, how will sLoc behave when we will start to build the hierarchy out of multiple repositories? Here as well the performance of sLoc is completely left up to the designer of the hierarchy build algorithm. Using the various metrics we introduced, F will be tuned. Experiments have to be carried out for each application in order to determine what feature of sLoc is most important; it may be coverage, sometimes intracluster density, and sometimes only intercluster density may matter. We see sLoc as a highly flexible tool, fast and versatile; now only the application designer will know how to get the best out of it.

5.2 Hierarchical Classification of Knowledge and HDDI

Our basic premise in HDDI is that, to effectively process information in a dynamic and scalable way, we need to organize it according to a hierarchical classification.

The validation experiments carried out in Chapter 4 confirmed the need for building a hierarchy of HDDI nodes. The first reason is to not only overcome, but also leverage the band-pass effect of most indexing algorithms that will narrow down the resolution of an index to a given quality of discriminators. Second, building a hierarchical classification of knowledge is a significant aid in conceptual search; well-known search engines such as *Yahoo!* do use a hierarchical classification to match the way people would intuitively go about searching information. Third, queries, by nature, are not organized according to a

hierarchical structure; they have their own internal structure and we need to match this structure in order to achieve maximum precision and recall.

We designed and implemented a first-pass search and retrieval strategy using HDDI. We learned a lot from the results. On the one hand, the strategy works well for the cases when regions of semantic locality at adjacent levels overlap well and relevant documents are identified. On the other hand, we noticed two main issues. First, the retrieval path from the root node to the collection is sometimes discontinuous. That is, a region of semantic locality at one level of the HDDI does not overlap with any region of semantic locality at the HDDI node right below. Second, large clusters in some HDDI nodes can lead to significant loss in precision.

Alternatives can be found to improve the performance of our search and retrieval system. We can take any of the following courses:

- Modify our strategy, that is, modify the way the retrieval paths through the HDDI are determined;

- Tune the parameters in sLoc to get different knowledge neighborhoods. We have been using the optimization function F (see Equation (3.13), page 45) to determine the threshold τ used in sLoc; other values of τ would lead to other types of clustering and eventually to other relevance judgments.

- Change the way the hierarchy is built, using another pruning approach based on properties other than item frequency.

- Use a different approach to computing similarities between regions of semantic locality using, for example, co-occurrence matrices.

- Change the way kBase computes the arc weights to avoid homogeneous knowledge neighborhoods with respect to item frequency.

With respect to implementation, HDDI will eventually distributed across different systems. New issues will arise when addressing this implementation, such as naming,

synchronization and all the traditional issues of distributed computing. We are aware of them, and although not tackled in this thesis, they are part of our constraints in designing the system framework.

As explained in Section 2.1.2, in designing an IR system there are some research issues that have to be considered on the way. Although the HDDI architecture answers some of these, other features of HDDI are left to be adapted to a particular application. Specifically, in its present version the HDDI model does not explicitly define the way queries would be entered. Another issue, as we saw, is the strategy used to map the query onto the HDDI. Finally, besides precision and recall analysis, other indicators of relevance might come into play.

5.3 Other Applications

In this thesis, I focused on the query search and retrieval application of HDDI, but extensive research is being carried out at NCSA in detection of emerging conceptual contents and targeted marketing analysis. Hopefully, the tools, measures and understanding we obtained from the standpoint of query search and retrieval will benefit other applications as well. Finally, as we improve our understanding of the potential of HDDI, we can expect new application ideas to emerge from our research efforts.

REFERENCES

[1] Los Alamos National Laboratory, "arXiv.org e-Print archive." http://xxx.lanl.gov, 1999.

[2] National Research Council CSTB, *Computing The Future.* Washington DC: National Academy Press, 1992.

[3] C. J. van Rijsbergen, *Information Retrieval.* London: Butterworths, 1979.

[4] R. B. Korfhage, *Information Storage and Retrieval.* New York: Wiley, 1997.

[5] R. B. Korfhage and T. G. DeLutis, "A basis for time and cost evaluation in information systems," *The Information Bazar. Proceedings of the Sixth Annual Colloquium on Information Retrieval*, pp. 293–326, 1969.

[6] C. W. Cleverdon, "On the inverse relationship of recall and precision," *Journal of Documentation*, vol. 28, pp. 195–201, 1972.

[7] G. Salton, *Dynamic Information and Library Processing.* Englewood Cliffs, New Jersey: Prentice Hall, 1975.

[8] G. Salton and M. McGill, *Introduction to Modern Information Retrieval.* New York: McGraw-Hill, 1983.

[9] K. S. Jones, *Automatic Keyword Classification for Information Retrieval.* London: Butterworths, 1971.

[10] H. P. Luhn, "A statistical approach to mechanised encoding and searching of library information," *IBM Journal of Reasearch and Development*, vol. 1, pp. 309–317, 1957.

[11] M. E. Maron and J. L. Kuhns, "On relevance, probabilistic indexing and information retrieval," *Journal of the ACM*, vol. 7, pp. 216–244, 1960.

[12] H. F. Stiles, "The association factor in information retrieval," *Journal of the ACM*, vol. 8, pp. 271–279, 1961.

[13] M. E. Stevens, V. E. Guiliano, and L. B. Heilprin, *Statistical Association Methods for Mechanized Documentation.* Washington DC: National Bureau of Standards, 1964.

[14] I. J. Good, "Speculations concerning information retrieval," Tech. Rep. PC-78, IBM Research Centre, New York, 1958.

[15] R. A. Fairhorne, *The Mathematics of Classification.* London: Butterworths, 1961.

[16] L. B. Doyle, "Is automatic classification a reasonable application of statistical analysis of text?" *Journal of the ACM*, vol. 12, pp. 473–489, 1965.

[17] J. J. Rocchio, "Document retrieval systems – optimization and evaluation," PhD thesis, Harvard University, 1966. Report ISR-10 to National Science Foundation, Harvard Computation Laboratory.

[18] S. T. Dumais, M. W. Berry, and G. W. O'Brien, "Using linear algebra for intelligent information retrieval," *SIAM Review*, vol. 37, no. 4, pp. 573–595, 1995.

[19] C. W. Cleverdon, "Report on the testing and analysis of an investigation into the comparative efficiency of indexing systems," tech. rep., College of Aeronautics, Cranfield, UK, 1962.

[20] G. Salton, "Automatic text analysis," *Science*, vol. 168, pp. 335–343, 1970.

[21] S. E. Robertson, "Retrieval and relevance: On the evaluation of IR systems." The ISI Lazerow Lecture, November 1993.

[22] D. K. Harman (ed.), "The first text retrieval conference (trec-1)," *NIST Special Publication 500-207*, 1993.

[23] C. Lynch and H. Garcia-Molina, "Interoperability, scaling, and the digital libraries research," in *1995 IITA Digital Libraries Workshop*, May 1995.

[24] H. Chen and K. J. Lynch, "Automatic construction of networks of concepts characterizing document databases," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, pp. 885–902, September 1992.

[25] W. M. Pottenger, "Theory, techniques, and experiments in solving recurrences in computer programs," PhD thesis, Univ. of Illinois at Urbana-Champaign, Center for Supercomputing Res. & Dev., May 1997.

[26] N. Jardine and R. Sibson, "A model for taxonomy," *Mathematical Biosciences*, vol. 2, pp. 465–482, May 1968.

[27] B. Hendrickson and R. Leland, "The Chaco user's guide: version 2.0," Tech. Rep. SAND94–2692, Sandia, 1994.

[28] G. Karypis and V. Kumar, "Multilevel k-way hypergraph partioning," Tech. Rep. 98-036, University of Minnesota CS&E, 1998.

[29] B.-H. Wang, "Texture segmentation of SAR images," in *Proceedings of Image Understanding Workshop*, May 1997.

[30] J. F. Sowa, *Conceptual Structures: Information Processing in Mind and Machine.* Reading, MA: Addison-Wesley, 1984.

[31] G. W. Mineau and R. Godin, "Automatic structuring of knowledge bases by conceptual clustering," *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, pp. 824–829, October 1995.

[32] R. Sablowski and A. Frick, "Automatic graph clustering (system demonstration)," in *Proceedings of the Symposium on Graph Drawing, GD '96*, Berkeley, CA, September 1996.

[33] H. Schütze, "Automatic word sense discrimination," *Computational Linguistics*, vol. 24, no. 1, pp. 97–124, 1998.

[34] E.-H. Han, G. Karypis, and V. Kumar, "Clustering in a high-dimensional space using hypergraph models," Tech. Rep. 97-063, University of Minesota, 1998.

[35] J. A. Hartigan and M. A. Wong, "Algorithm 136. a k-means clustering algorithm," *Applied Statistics*, vol. 28, p. 100, 1978.

[36] P. Cheeseman, M. Self, J. Kelly, W. Taylor, D. Freeman, and J. Stutz, "Bayesian classification," in *Seventh National Conference on Artificial Intelligence*, Saint Paul, MN, pp. 607–611, 1988.

[37] R. E. Tarjan, "Depth first search and linear graph algorithms," *SIAM J. Computing*, vol. 1, pp. 146–60, 1972.

[38] A. V. Aho, J. E. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[39] W. Blume, R. Doallo, R. Eigenmann, J. Grout, J. Hoeflinger, T. Lawrence, J. Lee, D. Padua, Y. Paek, W. Pottenger, L. Rauchwerger, and P. Tu, "Parallel programming with Polaris," *IEEE Computer*, vol. 29, pp. 78–82, December 1996.

[40] W. M. Pottenger and D. Zelenko, "Concept space comparison and validation," Tech. Rep. UIUCDCS-R-98-2071, University of Illinois at Urbana-Champaign CS, December 1998.