Programa Oficial de Posgrado
(Master y Doctorado)

Trabajo Fin de Master

An introduction to content-based Information Retrieval by Normalized Compression Distance

Alumno: Rafael Martínez Martínez

Tutor: David Camacho Fernández

(Noviembre, 2009)

Table of contents

Abstract

This document presents an approach to content-based Information Retrieval. Most researches on this field apply the Normalized Compression Distance for clustering purposes. We present a research on Information Retrieval using NCD and statistical estimations based on outliers. This work also address the required architecture of documents and indexes (hypothesis matrix) used to evaluate the search engine. Finally we try to extract best parameter values for a given query such as the amount of results we are going to provide and its length.

Acronyms

AOM	Active Object Model
BIM	Binary Indepence Model
CSCP	Comprehensive Structured Context Profile
FN	False Negative
FP	False Positive
GUI	Graphical User Interface
HC	Huffman Code
ID	Information Distance
IR	Information Retrieval
LZ	Lempel-Ziv
LZW	Lempel-Ziv-Welch
MAP	Mean Average Precision
MTF	Move To Front
NID	Normalized Information Distance
NCD	Normalized Compression Distance
NL	Natural Language
NLP	Natural Processing Language
OOM	Object Oriented Models
ORM	Object-Role Modelling
OWL	Ontology Web Language
PHC	Plain Huffman Code
PPM	Prediction by Partial Matching
PRP	Probability Ranking Principle
QA	Question Answering
RDF	Resource Description Framework
RLE	Run Length Encoding
ROC	Receiver Operating Characteristics
SGML	Standard Generic Markup language
THC	Tagged Huffman Code
TN	True Negative
TP	True Positive
UML	Unified Modelling Language
VSM	Vector Space Model
XML	eXtensible Markup Language

List of figures

1. Introduction

Due to the increasing amount of digital information stored over the net, searching for electronic information has become an essential task. Every day, millions of documents are created and addressed into the internet and must be findable. The goal of any information that is stored in the net is to become read by interested people. The more people accessing the information, the better results for the creator. Generating documents for the web is the goal that it has been created for. At the same time, there are people needing information for every thing. Since more than a decade, web has become the first feed of information in quantity and quality, replacing traditional bibliography like books or articles (Pedroni, 1996). Web is now an essential part of our society and its education.

Information Retrieval (IR) can be defined as the activity of providing the user the information he/she is looking for. Sometimes is not possible to retrieve to the user the exact information he has in mind. In this way, systems are supposed to bring him the best suitable according to his needs. A lot of suppositions have been formalized in order to make things easier for this engineering (Jones, 2004). But the essence of these systems is far away from a content-based retrieval.

Crestani and Ruthven (Journal On Information Retrieval, 2007) define Contextual Information Retrieval as follows:

"*Context affects all aspects of Information Retrieval. A searcher's context affects how they interact with a retrieval system, what type of response they expect from a system and how they make decisions about the information objects they retrieve. Information contexts are formed through the creation, use and linkage of these objects within an information resource and the context of use impacts on how we evaluate retrieval systems. Our understanding of user and information context also influences how we design and construct retrieval systems themselves*".

This term is not the most suitable for this work although it considers a lot of things that are useful for us. In this way it is more appropriate to use the "content-based" term to define the goal of this document. We are not analyzing user past's reviews or related information that are around his search, neither studying his behaviour with documents retrieved. The aim of this document is to address the problem of large queries definition and how documents have parts, paragraphs or extracts that contain such information structured in different ways from the initial query.

Final results must be a set of documents that suite best user needs. Traditional Information Retrieval methods and techniques are required in order to organize results and measure their quality (Baeza-Yates, Ribeiro-Neto, 1999).

Content-based retrieval can be addressed in different ways like Natural Language Processing, ontologies, compression, etc. In this document, compression will be the selected method to implement our content-based search engine.

The main goals of this work are:

- Analyze the most suitable compression algorithm for this kind of searching.
- Find an optimal method to structure our information.
- Evaluate NCD behaviour with different types of queries.
- Evaluate useful domains where NCD is useful for.
- Find statistical methods that help to solve "identical" results.

Section 2 gives an overview of the state of the art. It covers most used techniques of traditional IR systems and their evaluation, including most popular Google's ideas. It also brings how compression is used nowadays in text searching, popular algorithms and compression-decompression systems.

Section 3 explains every single theory of the State of the Art used for this research. It covers complexity, Kolmogorov complexity, *Normalized Information Distance* and NID applied to compression (NCD).

Section 4 explains how we use the mentioned theory to provide best results for a given query. It contains all our major concepts to improve results, including data structure, information overlap and statistical parameters to detect similarity (*outliers*). We also give a brief overview of the developed software and architecture.

Section 5 presents a set of experiments practiced to evaluate this approach. It comments particularities of every single query and dataset used to evaluate the results. Every experiment has an evaluation in terms of traditional IR effectiveness.

Finally, in Section 6 and 7 we discuss obtained results and comment further work in much other areas.

2. State of the art

The IR problem consists mainly of building and storing information in the best way, using as less space as possible. Then we must provide a method to measure how good is a query from a potential relevant document and it must be effective and efficient. Finally, search engine must provide an ordered set of documents based on their similarity. In this way it is needed a 'ranking' algorithm (Baeza-Yates, Ribeiro-Neto, 1999).

From the human point of view, IR consists of analyzing users' behaviour with our data store, web or information space and extracting which information fits the best according to his needs.

There have been a wide range of methods to satisfy these requirements. First methods were based on similarity and distance from a query to a document, both of them considered as a set of 'keywords'. Data structure and compression were also discovered as good points of improvement for IR. Then clustering of documents was important in order to evaluate the goodness of each method. Finally other approaches tried to evaluate documents and queries further than a set of words that are not matched between them, those are the case of ontologies, entropy and other important research areas.

This part of the document tries to provide a brief explanation of most important approaches in the area of IR, their importance, utility and some examples that worked well in the past, present and why not in the future.

2.1. Classic and Modern Information Retrieval

From the older sets of papers stored in the past, to the new digital world, IR has advance din an exponential way. Indexes have demonstrated that they are a good approach in order to find things faster. Other approaches like vectors of terms and matrixes are a type of indexes which contains structured terms in any way.

Indexes: In the IR field, research has considered 'inverted indexes' or 'inverted files' and in fact they have been considered as a standard[1]. The basic idea of indexed is to keep a dictionary (*vocabulary* or *lexicon* of a document) matched with every document Id where they appear in. The list of document Id is often called *posting.*

Indexes can be compressed and enhanced to better describe each document. Such is the case of organizing postings in order, or inserting term frequencies for each document.

[1] Some information retrieval researchers prefer the term inverted file, but expressions like index construction and index compression are much more common than inverted file construction and inverted file compression.

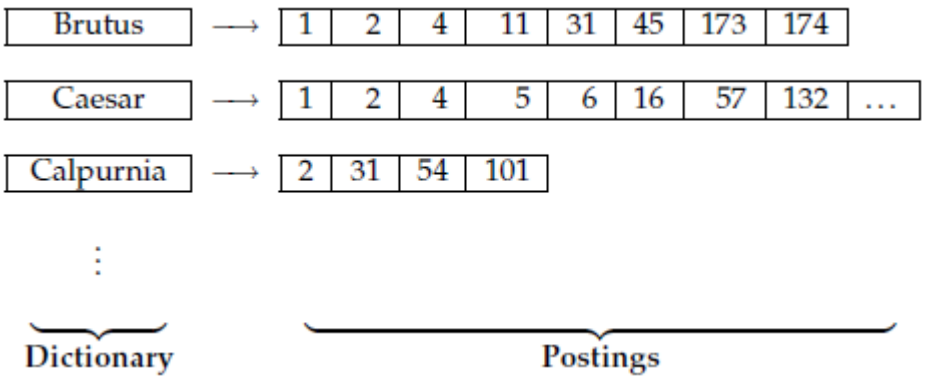


Figure 2.1- A simple view of a classic index

Boolean method: The simplest IR technique for retrieving documents from a given query. It is based on extraction of keywords from query, retrieve the documents Id for each keyword and intersect them to obtain final documents ordered by frequency of each term. It is a simple "*term1* AND *term2* AND … AND *termN" intersection*.

```
INTERSECT(ht1, . . . , tni)

      terms ← SORTBYINCREASINGFREQUENCY(ht1, . . . , tni)
      result ← postings( f irst(terms)) terms ← rest(terms)
      while terms 6= NIL and result 6= NIL
      do result ← INTERSECT(result, postings( f irst(terms)))
      terms ← rest(terms)

return result
```

Figure 2.2- The simple boolean algorithm

Vector Space Model: A good approach to automatic indexing is the Vector Space Model (Salton, Wong, Yang, 1975). An approach based on space density computation is used to choose an optimum indexing vocabulary for a given set of documents. This model considers a set of terms $Q=t_1,...,t_n$ from a given query. Every document is described as

$D_i=(d_{i1},\ldots\ldots,d_{ij})$

where d_{ij} is the frequency of the j-th term in the i-th document. Frequency can be estimated as binary (present / not present) or weighted with several methods (Spärck Jones, 1972; Salton, McGill, 1983; Salton, Fox, Wu, 1983; Salton, Buckley, 1988).

That lets us to have vectors that estimate similarity between pairs of documents, a very useful method for clustering for example[2]. For a given pair of documents, we can use conjunction or disjunction way,

[2] A wide range of well known methods for clustering collections based on term vectors exist in the bibliography (Salton, 1968).

$D_1=(1,0,0)$ AND $D_2=(1,1,0)$ | $D_1=(1,0,0)$ OR $D_2=(1,1,0)$

We can extract the distance from one vector to the other based on its co-sinus.

$$sim(\vec{D}_1, \vec{D}_2) = \cos\theta = \frac{\vec{D}_1 \cdot \vec{D}_2}{\left|\vec{D}_1\right| \times \left|\vec{D}2\right|} = \frac{\sum_{i=1}^{t} d_{i,1} \cdot d_{i,2}}{\sqrt{\sum_{i=1}^{t} d^2{}_{i,1} \cdot \sqrt{\sum_{i=1}^{t} d^2 2}}} \quad (2.1)$$

Probability Ranking Principle*:* The PRP assumes the relevance probability for a given query *q* and each document of a set. If a term A of a query *q* is present in the *d* document, we can assume that there exists a measurable probability that this *d* document is relevant for that query. This probability is often represented as:

$P(R=1|d, q_A)$

This expression can be extended for each term of the query *q*, assuming independence between them. Every term of the query has a weighted value. Finally, probability is the product of every single probability of each term.

The basis of the PRP was established by van Rijsbergen (1979):

"If a reference retrieval system's response to each request is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data."

Binary Independence Model*:* In order to estimate PRP value for $P(R=1|d, q)$ there exists a method introduced by Rijsbergen and Jones (1976). Documents and queries are both represented as binary term incidence vectors. That is, a document d is represented by the vector $x = (x_1, \ldots, x_M)$ where $x_t = 1$ if term t is present in document d and $x_t = 0$ if t is not present in *d*. The model recognizes no association between terms. Under the BIM, we model the probability $P(R|d, q)$ that a document is relevant via the probability in terms of term incidence vectors $P(R \mid \vec{x}, \vec{q})$. Then, using Bayes rule, we have:

$$P(R = 1|\vec{x},\vec{q}) = \frac{P(\vec{x}|R = 1,\vec{q}) \cdot P(R = 1|\vec{q})}{P(\vec{x}|\vec{q})} \quad (2.2)$$

and $P(R = 0|\vec{x},\vec{q}) = 1 - P(R = 1 \mid \vec{x}, \vec{q})$ (2.3)

In order to estimate how a present term in a document affects its probability of relevance, we can make the *Naive Bayes conditional independence assumption* that the presence or absence of one word in a document is independent of the presence or absence:

$$\frac{P(|\vec{x} \mid R=1,\vec{q})}{P(|\vec{x} \mid R=0,\vec{q})} = \prod_{t=1}^{M} \frac{P(x_t|R = 1,\vec{q})}{P(x_t|R = 0,\vec{q})} \tag{2.4}$$

Considering $x_t=1$ or $x_t=0$, we can assume the relevance of a document given a query as:

$$O(R \mid \vec{x},\vec{q}) = O(R \mid \vec{q}) \cdot \prod_{t:x_t=1} \frac{P(x_t = 1 \mid R=1,\vec{q})}{P(x_t = 1 \mid R=0,\vec{q})} \cdot \prod_{t:x_t=0} \frac{P(x_t = 0 \mid R=1,\vec{q})}{P(x_t = 0 \mid R=0,\vec{q})} \tag{2.5}$$

where $O(R \mid \vec{q})$ is a constant.

Natural Language Processing: A newer approach to accurate IR results consists in analyze language properties as they are, not simply by means of weights and probabilities. Having a look to some query, we can suppose a type of words that give no additional information. That is the case of pronouns, prepositions, adverbs and other words that are not nouns, adjectives or verbs (the most sense type of words). Refining that, we can consider that almost any of the mentioned words are composed by prefix+*stem*+suffix. Brants (2004) explains the most important researches in this area, where we can extract the next ones:

- *Stopwords* (no content words) can be eliminated.
- *Stemming* content words into shorter words.
- *N-grams* are a set of N words that has their own meaning i.e. 'in the other hand'
- Part-of-Speech tagging consists of assigning a syntactic category to each word in a text, thereby resolving some ambiguities. Some techniques have been raised like *statistical* (Ratnaparkhi 1996, Brants 2000), *memory-based* (Daelemans et al. 1996), *rule-based* (Brill 1992) and many more.
- *Compound and statistical phrases* used 'often', are a type of n-grams.
- *Compound splitting* for some language like Anglicans-derived.
- *Word disambiguation* depending on the context, looking to adjacent text.
- *Query length* affects to NLP effectiveness (Strzalkowski et al. 1999).

For this document we are very interested in retrieval results considering query length. In this sense, *Question Answering* and *Information Retrieval* are shown to be most effective using NLP techniques (Brants, 2000).

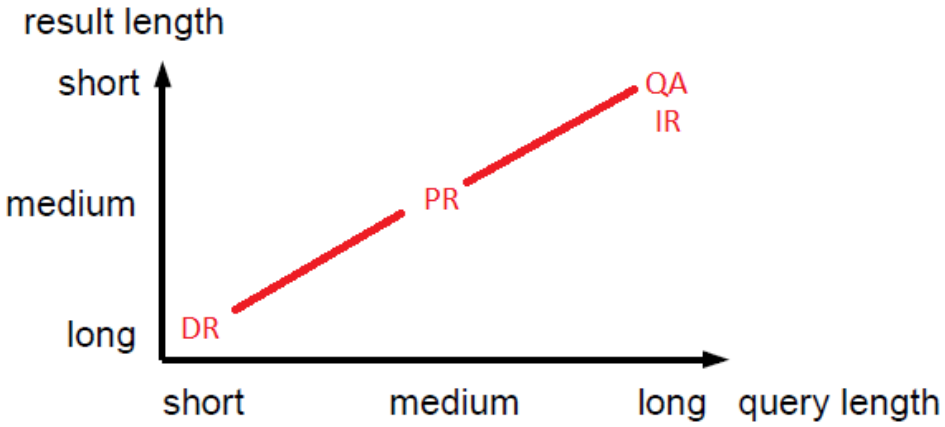


Figure 2.3- Classification of Document Retrieval, Passage Retrieval, Question Answering.

A practical successful example: the Google PageRank
(Brin and Page, 1998)

Very few algorithms and methods have had the importance on IR that Google's PageRank had. Efficiency and effectiveness combining with weighting have been merged into a simple technique that lets the user to participate actively in the importance of his information. We must note that this algorithm make only sense in the *hyperspace*. Only linked information can be processed an evaluated with this method. Isolated documents might receive some evaluation, not considered as the Google PageRank by their own.

Introducing the expression, PageRank seems in its simplest version as follows:

$$PR(A) = (1-d) + d * \sum_{i=1}^{n} \frac{PR(i)}{C(i)} \qquad (2.6)$$

Where:

- ***d*** is a constant between 0 and 1.
- ***PR(i)*** are the PageRank values of each page pointing from i to A
- ***C(i)*** is the number of outer links of page i.

Progressive modifications have been introduced since the first implementation of the algorithm, but the basis remains being the first one.

2.2. Compression in Information Retrieval

Text compression (Bell et al., 1990) consists on represent original words or sets of characters with fewer elements than their original representation. Text is intended as a sequence of symbols. Traditionally (Brisaboa et al., 2006) best text compression techniques are based in *codeword*. This is, a final symbol that represents every source symbols (words, characters, etc.) and is the same along the whole text. The map between source symbol and final symbol is called *code*.

In this scenario, we consider best compression codes as the ones that compress best any text and are faster to search in compressed mode than in uncompressed one. Many codes have been developed until the date, but more successful ones have been *Huffman* based. The *Huffman* code (1952) is the optimal code for any frequency distribution. It has been traditionally applied to text compression by considering *characters* as source symbols, and bits as representing symbols.

The technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols, n. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the symbol itself, the weight (frequency of appearance) of the symbol and optionally, a link to a parent node which makes it easy to read the code (in reverse) starting from a leaf node. Internal nodes contain symbol weight, links to two child nodes and the optional link to a parent node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has n leaf nodes and n − 1 internal nodes.

If the symbols are sorted by probability, there is a linear-time (O(n)) method to create a Huffman tree using two queues, the first one containing the initial weights (along with pointers to the associated leaves), and combined weights (along with pointers to the trees) being put in the back of the second queue. This assures that the lowest weight is always kept at the front of one of the two queues:

```
1. Start with as many leaves as there are symbols.
2. Enqueue all leaf nodes into the first queue (by probability
   in increasing order so that the least likely item is in the
   head of the queue).
3. While there is more than one node in the queues:
      1. De-queue the two nodes with the lowest weight by
         examining the fronts of both queues.
      2. Create a new internal node, with the two just-removed
         nodes as children (either node can be either child) and
         the sum of their weights as the new weight.
      3. Enqueue the new node into the rear of the second queue.
4. The remaining node is the root node; the tree has now been
   generated.
```

Figure 2.4- A Huffman pseudo-code (Wikipedia, 2009)

A final tree representation is obtained with the above algorithm:

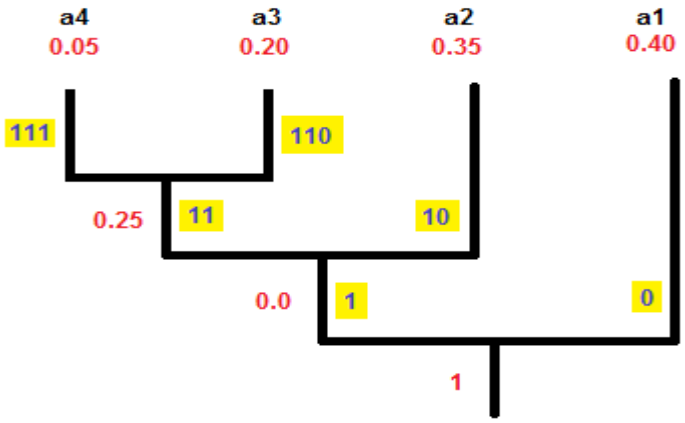


Figure 2.5- A Huffman tree example

On natural language text, this method gives poor compression ratios (around 65%). The key idea is to consider larger *input* symbols, such as words. The distribution of words in NL is much more skewed than letters one. As a result, ratios get down to 25%. The cost of having larger set of source symbols (the number of different words are much higher than the number of different characters) is not significant on large text collections, as the vocabulary grows slowly ($O(N^{\beta})$ symbols in a text of N words, for some $\beta \approx 0.5$, by Heaps Law (Heaps, 1978; Baeza-Yates and Ribeiro-Neto, 1999)).

Compression makes sense because it yields better *searching* (compression, search and decompression) times than looking in the entire text. Having this in mind, the only remaining thing is to analyze best *source* and *target* symbols. With such a large source of vocabulary, it is logic to have a larger target alphabet. Using bytes as target symbols was studied by Moura et al. (2000), who proposed a two byte-oriented and word-based Huffman coding as a way to speed up the process.

But there still remains a problem. The search process consists of compress a *pattern* and search the compressed text for it. But two or more target *codewords* can begin in the same way or simply a non-present pattern we are looking for is a *subset* of a real present one, so it can carry out some *false* matches. To avoid this, it makes necessary to process the text bits sequentially. A sequential processing ensures that the search is aware of the codeword beginnings and thus false matches are avoided.

One good approach is *Plain Huffman Code* (a code where source are text words and separators, and targets symbols are bytes). Compression ratios are 30% (5% overhead with respect the bit-oriented ones) but in exchange, searching is 30% faster as no bit manipulation is needed. Other good code is *Tagged Huffman Code*, similar to the above one, but using the highest bit of each byte to signal the first byte of each codeword. Hence, only 7 bits are used for the Huffman code. THC produces 11% larger compressed text, reaching 35% of compressed ratio.

Tagged Huffman Code have some advantages with respect Plain one. Tagged Huffman Code can access any position for decompression, even in the middle of a codeword, thanks to the flag bit. Plain Huffman Code can start decompression only from codeword

beginning. With a simple classic string matching algorithm, THC can be searched efficiently. This is not possible in PHC because of the *false matches* problem.

For our interest, there exist other methods with competitive compression ratios on NL, but unable of searching in compressed text *faster* than in uncompressed text. These include Ziv-Lempel compression (Ziv and Lempel, 1977-1978), Burrows-Wheeler compression (Burrows and Wheeler, 1994) implemented for *bzip2*, Move to Front (MTF), Run Length Encoding (RLE) and Prediction by Partial Matching (PPM).

Ziv-Lempel compression methods (LZ77 and enhancements, LZW and variants) are most used ones, leaving others as auxiliary preprocessing methods. The LZ-family methods consist of find the longest match to the current part of the input stream in the already passed part of the input stream. The simplest algorithm was implemented for LZ77:

- Define constant W to be "window size".
- Loop the input stream until the end of the stream is reached.
  - Sign p the current position in the input stream.
  - Scan the input stream from symbol max(0, p-W) to max(0, p-1).
    - Sign q the scan pointer.
    - Check the count of symbols that are equal in the input stream, starting from the points p and q. The stream comparison must be stopped on first unmatched symbol.
    - Check if the found length of equal peaces of the stream is maximal. The offset and length of this longest stream must be saved.
  - Output the found backward offset and length.
  - Continue the loop.

Figure 2.6- The basic LZ77 algorithm

The encoded stream contains this data:

- *Uncoded Symbol*
- *Offset to Previous*
- *Sequence Length minus 1*

In this version of LZ, *window size* refers to the maximum searchable length of any character sequence. LZ78 upgrades some features of previous LZ77, enhancing dictionary with appearance of each unencoded symbol. Each time, if symbol found that can not be encoded, the dictionary is enhanced with new entity that is the last found entity plus one symbol. LZ78 proposes the empty dictionary with the only value (the NULL value). If nothing appropriate is found in the dictionary, the NULL-value is output. After NULL value a single symbol is coming exactly as after any other dictionary value. LZW proposes the dictionary initialized with all possible symbols. This way there is no more any need in the NULL value.

A detailed list of most known compression methods, variants and algorithms can be found in Arkadi Kagan.

2.3. Clustering in Information Retrieval

Clustering in Information Retrieval makes sense when expert's review is not possible or the task is huge to human possibilities. There must be an automatic way to organize documents by their similarities and dissimilarities. Thus can revert into a better IR evaluation or whether retrieved information is the type of information the user is really looking for.

Manning C.D. et al. (2009) define *clustering* like

"Clustering algorithms group a set of documents into subsets or clusters. The algorithms' goal is to create clusters that are coherent internally, but clearly different from each other. In other words, documents within a cluster should be as similar as possible; and documents in one cluster should be as dissimilar as possible from documents in other clusters."

The main idea of clustering algorithms is based on *distance* measure (see Chapter 3). In document clustering Euclidean distance is the most common metric. There exist some clustering methods like *flat clustering* (creates a flat set of clusters without any structure that would relate clusters to each other) and *hierarchical clustering* creates a hierarchy of clusters. There are also clustering methods as *hard clustering* (each document belongs to exactly one cluster) and *soft clustering* (a document can cover more than one cluster). It becomes a very important task for our goal to effective clustering document in an almost optimal way. Such little errors can bring us to an evaluation error of our approach.

The *cluster hypothesis* states the fundamental assumption we make when using clustering in information retrieval.

> **Cluster hypothesis**. Documents in the same cluster behave similarly with respect to relevance to information needs.

The hypothesis states that if there is a document from a cluster that is relevant to a search request, then it is likely that other documents from the same cluster are also relevant. This is because clustering puts together documents that share many terms. There exist a wide range of techniques in order to cluster effective. Most used ones are *search result clustering*, where recovered documents from a query are clustered as similar. *Language modeling* (Liu & Croft, 2004) has high gains in *precision* and *recall*; and *cluster-based retrieval* (Salton, 1971) has higher efficiency for faster searches.

Other type of clustering is user-oriented and enhances the interface to avoid user to refine results retyping keywords. *Scatter-Gather* (Cutting et al., 1992) *clustering* refers to search result refinement by the user or internal dissimilarities, that is, subsets of retrieved documents from a query. *Collection clustering* (McKeown et al., 2002) gives effective information on presentation for exploratory browsing.

2.4. Contextual IR is not Content-based IR

Although most used algorithms for IR purposes are focused on keywords and their appearance on documents, we must know that there is other type of approaches. We refer to *content* and *contextual* information retrieval. It might seem the same thing but both methods are very different. *Content-based IR* involves the text as a single item, having its own meaning. *Contextual IR* combines both keyword and user environment in order to enhance results. Labelling documents or saving user behaviour are two types of well known proceedings.

As the number of documents and their length are always growing up, it becomes necessary to represent them in a shorter way. It will yield more efficiency as less memory and resources are needed. Also, the usefulness of computer technologies is bringing users to commodity when searching with tools or web engines. Users want to find information with a minimum cognitive load and maximum of enjoyment (Marchionini, 1992).

Contextual Information Retrieval

Anick and Vaithyanathan (1997) define *context* as "two components - a cluster of logically related articles (its extension) and a small set of salient concepts, represented by words and phrases and organized by the cluster's key terms (its intension)". Probably last definition don not give us enough meaning. Day (2001) describes better the concept of *context* in the Information Retrieval environment:

> *"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."*

Thus definition allows a better comprehension of what developers must do to represent it. In order to do that, it makes sense to define an *entity* for storing such information. That entity if often called a *model*. Traditionally, context only applied to location, identity and time (Strang, 2004). Throughout this section we will survey the most relevant context modeling approaches. These are classified by the scheme of data structures which are used to exchange contextual information in the respective system.

Key-Value models: It is the most simple data structure for modelling contextual information. It used key-value pairs to model the context by providing the value of context information (e.g. location information) to an application as an environment variable. Key-Value pairs are easy to manage, but lack capabilities for sophisticated structuring for enabling efficient context retrieval algorithms.

Mark-up Scheme Models: This is not a simple flat way to model context, yet it defines a structured document to organize it. With the definition of new mark-up standard languages, it became an interesting manner to store information avoiding ambiguity. The *Standard Generic Mark-up language* (SGML) and its subclass XML provides a powerful scheme to manage information as a so-called *container*. RDF[3] (Resource

[3] http://www.w3.org/RDF/

Description Framework) is the most common format to define and manage models, and CSCP (Comprehensive Structured Context Profile) introduced by Held et al. (2002) provides all functionality and expressivity to model all profile information as required for contextual information.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cscp="context-aware.org/CSCP/CSCPProfileSyntax#"
xmlns:dev="context-aware.org/CSCP/DeviceProfileSyntax#"
xmlns:net="context-aware.org/CSCP/NetworkProfileSyntax#"
xmlns="context-aware.org/CSCP/SessionProfileSyntax#"
<SessionProfile rdf:ID="Session">
        <cscp:defaults rdf:resource="http://localContext/CSCPProfile/previous#Session"/>
        <device><dev:DeviceProfile>
                <dev:hardware><dev:Hardware>
                <dev:memory>9216</dev:memory>
                </dev:Hardware></dev:hardware></dev:DeviceProfile>
                </device>
</SessionProfile>
</rdf:RDF>
```

Figure 2.7- A CSCP profile example

Graphical Models: As it refers, graphical models use available graphical modeling languages in order to define profiles. A powerful one is the Unified Modeling Language (UML)[4]. Due to its generic structure, UML is also appropriate to model the context. Another example is the nicely designed graphics oriented context model introduced in Henricksen et al. (2003), which is a context extension to the Object-Role Modeling (ORM)[5] approach (Halpin, 2001) according some contextual classification and description properties (Henricksen et al., 2002). In ORM, the basic modeling concept is the fact, and the modeling of a domain using ORM involves identifying appropriate fact types and the roles that entity types play in these.

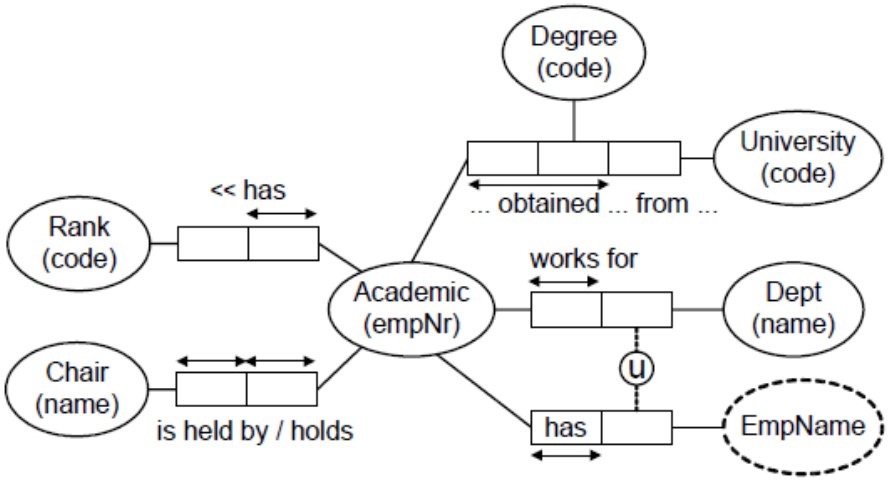


Figure 2.8- Contextual ORM example

Object Oriented Models: The OOM are based on logical units that store all available contexts for ubiquitous environments. Encapsulation, reusability and scalability are three main advantages of this approach. A valid and specific interface must be uses in order to work with objects. Well known approaches are *cues* (Schmidt et al., 1999) and *Active Object Modelling* (AOM) of the GUIDE project (Cheverst et al., 1999).

[4] http://www.omg.org/technology/documents/profile_catalog.htm
[5] http://www.orm.net/

Logic Based Models: In a logic based context model, the context is defined as facts, expressions and rules. Usually contextual information is added to, updated in and deleted from a logic based system in terms of facts or inferred from the rules in the system respectively. Such kind of logic can be applied to the system in order to reasoning from available facts or situation and derive new knowledge from a provided set of rules. An example of this deduction can be:

User.hasEntered("time")∧time.now("night")→User.sleep()
User.sleep()∧lowSurfing()→Spam("bed Ads")

Interesting approaches were introduced by *Multicontext Systems* (Ghidini et al., 2001) which is less on context modeling than on context reasoning and *Extended Situation Theory* (Akman & Surav, 1997), tried to cover model-theoretic semantics of natural language in a formal logic system which introduced some presuppositions.

Ontology Based Models: Ontologies are a promising instrument to specify concepts and interrelations (Uschold & Grüninger, 1997). Ontologies are one way of modelling context by specifying the domain knowledge linked to the aspect considered. They are particularly suitable to project parts of the information describing and being used in our daily life onto a data structure utilizable by computers. Ontologies can define different aspects of the context, such as the domain, the task or the user. A semantic indexing if often used to process searches. Thus consists in searching for the concept referenced in the granules (items that contain knowledge) and weighting those concepts according to the semantic representativity of the granules. Hernandez et al. (2007) propose a way to define domain and task through domain ontologies and provide a friendly user interface to browse on it. The Web Ontology Language (OWL)[6] is the family of knowledge representation languages for authoring ontologies, and is endorsed by the *World Wide Web Consortium*. OWL provides the capability of creating classes, properties, defining instances and its operations:

- ***Classes:*** User-defined classes which are subclasses of root class owl.
- ***Properties:*** A property is a binary relation that specifies class characteristics. They are attributes of instances and sometimes act as data values or link to other instances.
- ***Instances:*** Instances are individuals that belong to the classes defined. A class may have any number of instances. Instances are used to define the relationship among different classes.
- ***Operations:*** OWL supports various operations on classes such as union, intersection and complement. It also allows class enumeration, cardinality, and disjointness.

OWL ontologies are most commonly serialized using RDF/XML syntax.

[6] http://www.w3.org/TR/owl-ref/

2.5. Evaluating Information Retrieval

In order to evaluate the effectiveness of our information retrieval system with our ad-hoc results, we need three things:

- Document set
- Information we are searching for (most commonly called *queries*)
- Relevance judgment (which documents are *relevant* and which are *non-relevant*)

To provide accuracy to measures it is a good idea to have a more-less big set of documents. It depends of what we are measuring. If we like to provide similarity between documents, is not so crucial, but if we are testing an IR classic search engine it becomes interesting to test as much different information as possible. There are *standard* sets of information and their queries to compare and rank most effective systems. Most common ones are "The *Cranfield* collection"[7], TREC[8], NTCIR[9], CLEF[10] and "20 Newsgroups"[11].

The two most frequent and basic measures for information retrieval effectiveness are precision and recall (van Rijsbergen, 1979). These are first defined for the simple case where an IR system returns a set of documents for a query.

Precision (P) is the fraction of retrieved documents that are relevant:

$$Precision = \frac{\#relevant\_items\_retrieved}{\#total\_retrieved\_items} \tag{2.7}$$

Recall (R) is the fraction of relevant documents that are retrieved:

$$Recall = \frac{\#relevant\_items\_retrieved}{\#total\_retlevant\_items} \tag{2.8}$$

These notions can be made clear by examining the following contingency table:

	Relevant	**Non-Relevant**
Retrieved	*True positives (TP)*	*False Positives (FP)*
Not Retrieved	*False Negatives (FN)*	*True Negatives (TN)*

Figure 2.9- Table of relevance and retrieving matching

And then:

$$P=TP/TP+FP \tag{2.9}$$
$$R=TP/TP+FN \tag{2.10}$$

[7] 1398 abstracts of aerodynamics journal articles, a set of 225 queries, and exhaustive relevance judgments of all (query, document) pairs.
[8] 1.89 million documents (mainly, but not exclusively, newswire articles) and relevance judgments for 450 information needs, which are called *topics* and specified in detailed text passages
[9] `http://research.nii.ac.jp/ntcir/data/data-en.html`
[10] `http://www.clef-campaign.org/`
[11] It consists of 1000 articles from each of 20 Usenet newsgroups (categories)

We might think that we can measure the effectiveness of the system by its *accuracy*:

$A=TP+TN/TP+TN+FP+FN$

This seems plausible since there are two actual classes, relevant and non-relevant, and an information retrieval system can be thought of as a two-class classifier which attempts to label them as such (it retrieves the subset of documents which it believes to be relevant). This is precisely the effectiveness measure often used for evaluating machine learning classification problems.

There is a good reason why accuracy is not an appropriate measure for information retrieval problems. In almost all circumstances, the data is extremely skewed: normally over 99.9% of the documents are in the non-relevant category. A system tuned to maximize accuracy can appear to perform well by simply deeming all documents non-relevant to all queries. Even if the system is quite good, trying to label some documents as relevant will almost always lead to a high rate of false positives. However, labelling all documents as non-relevant is completely unsatisfying to an information retrieval system user. Users are always going to want to see some documents, and can be assumed to have a certain tolerance for seeing some false positives providing that they get some useful information. The measures of precision and recall concentrate the evaluation on the return of true positives, asking what percentage of the relevant documents has been found and how many false positives have also been returned.

The advantage of having the two numbers for precision and recall is that one is more important than the other in many circumstances. Typical web surfers would like every result on the first page to be relevant (high precision) but have not the slightest interest in knowing let alone looking at every document that is relevant. In contrast, various professional searchers such as paralegals and intelligence analysts are very concerned with trying to get as high recall as possible, and will tolerate fairly low precision results in order to get it. Individuals searching their hard disks are also often interested in high recall searches. Nevertheless, the two quantities clearly trade off against one another: you can always get a recall of 1 (but very low precision) by retrieving all documents for all queries! Recall is a non-decreasing function of the number of documents retrieved. On the other hand, in a good system, precision usually decreases as the number of documents retrieved is increased. In general we want to get some amount of recall while tolerating only a certain percentage of false positives.

A single measure that trades off precision versus recall is the *F measure*, which is the weighted harmonic mean of precision and recall:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \tag{2.11}$$

Where $\beta = \frac{1-\alpha}{\alpha}$ and $\alpha \in [0,1]$

Values of $\beta < 1$ emphasize precision, while values of $\beta > 1$ emphasize recall.

The reason for choosing a harmonic measure better than a simple arithmetic one, resides in the definition of recall. We can always get a 100% of recall, just retrieving all documents. Using an arithmetic mean as F=P+R/2 will bring us to the erroneous conclusion that effectiveness is 50%. In this case, every search engine, whatever their algorithm is, would have a minimum of 50% of accuracy. Obviously, we can not accept a search engine to be good with a 0 precision.

Let us suppose we have an already ordered list of retrieved documents and we have already calculated its precision and recall. We can evaluate it then with a *precision-recall curve*. Precision-recall curves have a distinctive saw-tooth shape: if the $(k+1)^{th}$ document retrieved is non-relevant then recall is the same as for the top k documents, but precision has dropped. If it is relevant, then both precision and recall increase, and the curve jags up and to the right. It is often useful to remove these jiggles and the standard way to do this is with an interpolated precision: the *interpolated precision* p_{interp} *at* a certain recall level r is defined as the highest precision found for any recall level $r' \geq r$:

$$p_{\text{int}erp}(r) = \max_{r' \geq r} p(r') \tag{2.12}$$

There is often a desire to boil this information down to a few numbers, or perhaps even a single number. The traditional way of doing this (used for instance in the first 8 TREC Ad Hoc evaluations) is the *11-point interpolated average precision*. For each information need, the interpolated precision is measured at the 11 recall levels of 0.0, 0.1, 0.2... 1.0. For each recall level, we then calculate the arithmetic mean of the interpolated precision at that recall level for each information need in the test collection. A composite precision-recall curve showing 11 points can then be graphed.

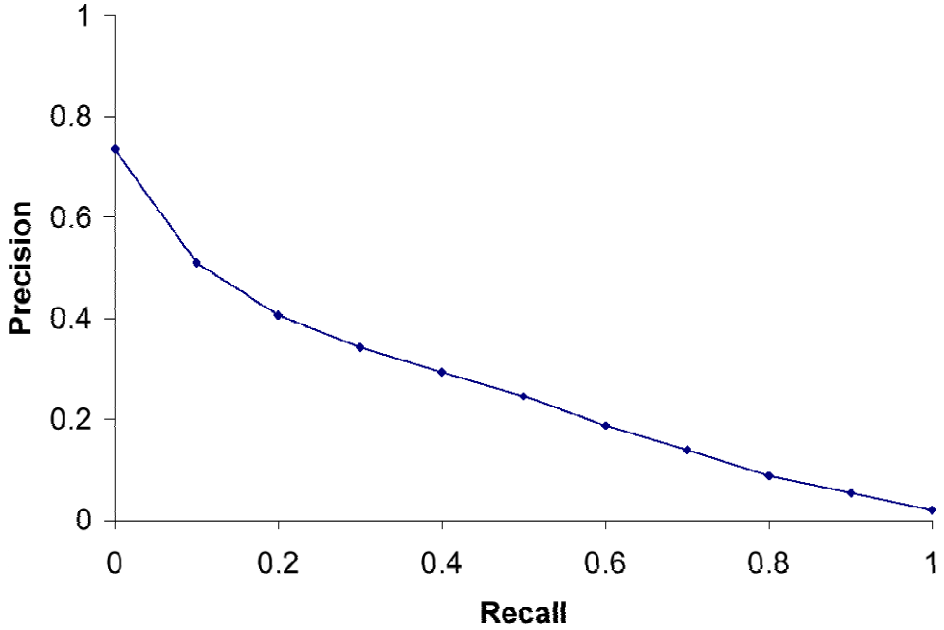


Figure 2.10- Averaged 11-point precision/recall graph across 50 queries for a representative TREC system. The Mean Average Precision for this system is 0.2553.

In recent years, other measures have become more common. Most standard among the TREC community is *Mean Average Precision* (MAP), which provides a single-figure

measure of quality across recall levels. Among evaluation measures, MAP has been shown to have especially good discrimination and stability. For a single information need, Average Precision is the average of the precision value obtained for the set of top *k* documents existing after each relevant document is retrieved, and this value is then averaged over information needs. That is, if the set of relevant documents for an information need $q_j \in Q$ is $\{d_1 ... d_m\}$ and R_{jk} is the set of ranked retrieval results from the top result until you get to document d_k, then:

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}) \tag{2.13}$$

When a relevant document is not retrieved at all, the precision value in the above equation is taken to be 0. For a single information need, the average precision approximates the area under the uninterpolated precision-recall curve, and so the MAP is roughly the average area under the precision-recall curve for a set of queries. Using MAP, fixed recall levels are not chosen, and there is no interpolation. The MAP value for a test collection is the arithmetic mean of average precision values for individual information needs. (This has the effect of weighting each information need equally in the final reported number, even if many documents are relevant to some queries whereas very few are relevant to other queries.) Calculated MAP scores normally vary widely across information needs when measured within a single system, for instance, between 0.1 and 0.7. Indeed, there is normally more agreement in MAP for an individual information need across systems than for MAP scores for different information needs for the same system. This means that a set of test information needs must be large and diverse enough to be representative of system effectiveness across different queries.

Another concept sometimes used in evaluation is an *ROC curve*. ("ROC" stands for "Receiver Operating Characteristics", but knowing that doesn't help most people.) A ROC curve plots the true positive rate or sensitivity against the false positive rate or (*1-specificity*). Here, *sensitivity* is just another term for recall. The false positive rate is given by *fp/fp+tn*. Figure 2.11 shows the ROC curve corresponding to the precision-recall curve in Figure 2.10. An ROC curve always goes from the bottom left to the top right of the graph. For a good system, the graph climbs steeply on the left side. For unranked result sets, *specificity*, given by *tn/fp+tn*, was not seen as a very useful notion. Because the set of true negatives is always so large, its value would be almost 1 for all information needs (and, correspondingly, the value of the false positive rate would be almost 0). That is, the "interesting" part of Figure 2.10 is *0<recall<0.4*, a part which is compressed to a small corner of Figure 2.11. But an ROC curve could make sense when looking over the full retrieval spectrum and it provides another way of looking at the data. In many fields, a common aggregate measure is to report the area under the ROC curve, which is the ROC analogue of MAP. Precision-recall curves are sometimes loosely referred to as ROC curves (Manning et al., 2009).

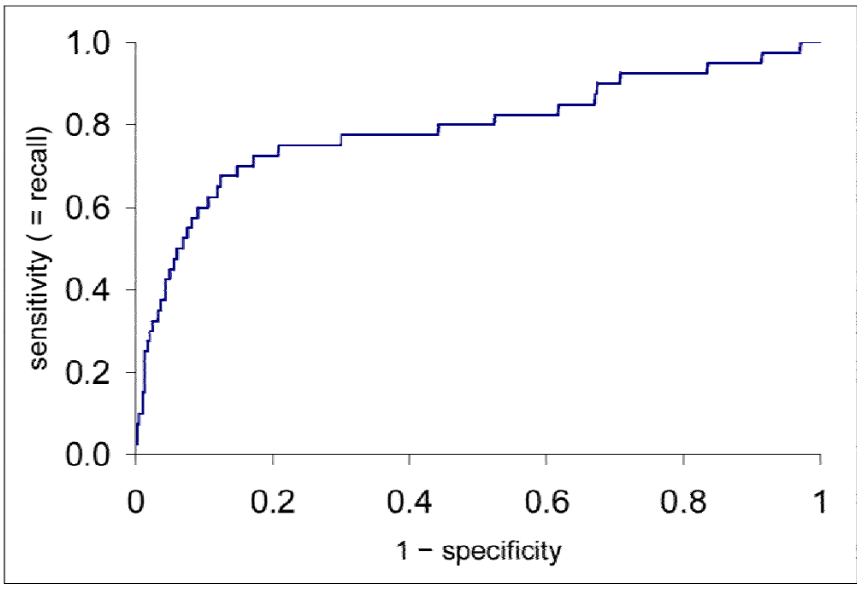


Figure 2.11- The corresponding ROC curve to the curve in Figure 2.10

Along this document we are not going to consider non-boolean relevance, but it also exists. There are some *weighted* methods of evaluating relevance based on judges made by independent observers. At last, relevance is a human-based measure. Manning et al. (2009) describe a wide range of methods for weighted relevance in Chapter 8 of their book "*Introduction to information retrieval*".

3. Measuring Information Distance

As explained in Section 2, contextual IR does not have many similarities with content-based information retrieval. Our approach tries to assess the problem of measuring distance. NLP address the problem in a less mathematical way, yet it analyzes information by its meaning and uses techniques like *query reformulation* (Efthimiadis, 1996); *question answering* (Huettner, 2000) or statistical analysis over the words.

But what happens if we try to extract a value that determines what type of information a passage has? Fortunately, such researches have already been taken into account. Can we define the same information in a different manner? The answer is yes. A good example can be compressed information. We can obtain source information from the compressed one using the appropriate transformation. We will see later that this transformation is commonly called *complexity*. Once we have information coded, next step is measuring *how much information* we have. How can we obtain the *amount* of information we have? *Entropy* (Shannon, 1948) is the answer. Entropy can be estimated via compression (Blelloch, 2000) and used in order to compare similarity, distance or dissimilarity between two or more sources of information. As every measure, *distance* is defined by a quantitative value. This approach tries to study how manipulate information in order to get better *distance* values, and which ones are better.

First of all we define *complexity* in this scenario and how it is determined. The second stage is to introduce the concept of *information distance* and other common synonyms or similar approaches. Then we present the *normalized information distance* and the importance of normalization in order to compare different measures. After presenting NID and complexity in terms of compression, we introduce the *Normalized Compression Distance* (which is the concept on what this work is based on) and analyze why it is important and its quasi-universality. A few methods of its implementation and results are also put together to emphasize the importance of NCD in the IR domain. After understanding these concepts, we are ready to present our approach and results.

3.1 The Kolmogorov Complexity

Li and Vitanyi (1993) affirmed that "Kolmogorov complexity measures the absolute information *content* of individual objects". For data mining and clustering (a way to do information retrieval) in becomes interesting to find a way to measure the absolute information *distance* between pairs of objects (Cilibrasi and Vitanyi, 2005). A notion to understand information distance between two objects X and Y is the minimum amount of information need to transform the *X* object into *Y* object. We provide a high-level definition of Kolmogorov complexity.

Let us suppose some transformation **φ**, a string input ***p*** and a string output ***x*** and such that $\varphi(p)=x$. We can say that the string p is a description of x in the description

language φ. The shortest description p is called the complexity of the string x in terms of φ. A more general view of complexity adds another string y. So the conditional complexity of x with y is defined as:

$$C_{\varphi}(x \mid y) = \min\{\ell(p) : \varphi(y, p) = x\} \tag{3.1}$$

Where $\ell(p)$ is the length of the string p, and $C_{\varphi}(x) = C_{\varphi}(x \mid \varepsilon)$ the unconditional complexity of x.

Clearly the complexity of a string depends on φ. A distinguished function φ_0 is the shortest function to define the string, which assigns the possible lowest value for all φ. So, there is a value c that for all φ:

$$C_{\varphi_0}(x \mid y) \leq C_{\varphi}(x \mid y) + c \tag{3.2}$$

for all x and y.

We denote C_{φ_0} as C later on. A helpful intuition for understanding C is to regard $C(x|y)$ as the length of a shortest computer program, in any popular language, that outputs x on input y. Regarding C we might think that we must be able to concatenate two C_1 and C_2 in order to obtain effectively y from x. That is what we call the Kolmogorov Complexity and denoted by $K(x|y)$ that is at the same time only a constant number of bits larger than the concatenation of the original two programs. Then we have:

$K(xy) \leq K(x)+K(y)+O(1)$.

$K(x)$ function has some properties that are formalized in Li and Vitanyi (1993). The most important for our purpose is the *subadditivity* property that stands:

$$K(x, y) = K(x)+K(y|x) = K(y)+K(x|y) \tag{3.3}$$

Proofs are omitted but are available in the above reference.

With the notion and first formalizations of the Kolmogorov Complexity we are ready to introduce the *information distance* and its properties.

3.2 Information Distance

Intuitively the information distance between two strings x and y might be the shortest program that converts x from y and y from x. In order to let the *complexity calculation program* be machine-independent, we must formalize the problem.
For a function φ let

$$E_{\varphi}(x, y) = \min\{\ell(p) : \varphi(p, x) = y \wedge \varphi(p, y) = x\}$$

There is a universal function ψ_0 such that for every ψ and all x, y,

$$E_{\psi_0}(x,y) \leq E_{\psi}(x,y) + c_{\psi} \tag{3.4}$$

Where c_{ψ} is a constant that depends on ψ but not on *x* or *y*.

So, for every two universal prefix functions φ_0 and ψ_0, we have for all *x*, *y* that $|E_{\varphi 0}(x, y)-E_{\psi 0}(x, y)| \leq c$, with *c* a constant depending on φ_0 and ψ_0 but not on *x* and *y*. Thus the following definition is machine-independent.

Fixing a particular universal prefix function ψ_0, *information distance* is defined as

$$E_0(x,y) = \min\{\ell(p) : \psi_0(p,x) = y \wedge \psi_0(p,y) = x\} \tag{3.5}$$

It has also been proved (Li and Vitanyi, 1993) that

$$E(x,y) = \max\{K(x \mid y), K(y \mid x)\} + \mathrm{O}(\log \max\{K(x \mid y), K(y \mid x)\}) = E_0$$

As a *distance metric*, information distance satisfies the following conditions (Chen et al., 2009)

- $E(x,y) \geq 0$ (non-negativity)
- $E(x,y) = E(y,x)$ (symmetry)
- $E(x,z) \leq E(x,y) + E(y,z)$ (triangle inequality)
- $E(x,y) = 0$ if and only if $x=y$ (identity of indiscernibles)

This finishes the formalization of complexity, Kolmogorov complexity and distance. It is clear that, following the above definitions, information distance does not apply to single characteristics of objects. Thus give us a notion that a non-keyword searching has sense. Even a non-text search has sense, although it must have any *string* representation as Kolmogorov requires. Now we might want to quantify how much some given objects differs one from the other with respect of a specific feature like the length of the objects in bits, the number of beats in music pieces or the number of occurrences of some base in the genome. It turns out that *E* is minimal up to an additive constant among all such distances. Hence, it is a *universal information* distance that accounts for any effective resemblance between two objects.

Let us consider the distance between two *A, B* identical pictures. Take pictures that are digitalized in black and white with binary strings. There are many possible ways to measure their distance, for example the Hamming distance and the Euclidean distance. If we change some bits of one of the pictures, they remain as similar as before, and Hamming distance proves it. But what happens if we turn all bits of one picture? Hamming distance turns maximal, while information distance proves the opposite. We must only define a program *K* that for the given picture *A* outputs *B* in less distance that Hamming. So, classical measures are not always useful to solve all problems.

Another example can bring us to an error. Consider two binary strings of 1,000,000 bits that differs in 1,000 bits. We can say that they are very similar. But if we have two binary strings of 10,000 bits each one and differ in 1,000 bits, we are close to think that they are very similar. Another real example shows that this is not always true. For example, *E.Coli* bacteria have a genome of 4.8 megabase long. A sister-specie called *H. Influenza* has 1.8 megabase long. Another, but very remote specie, *A. Fulgidus* is 2.18 megabase long. A trivial calculation of distance can tell us that *H. Influenza* and *A. Fulgidus* are much closer than *H. Influenza* and *E.Coli*. Yet this is not true. This occurs because of the predominance of the length over the information. In order to deal with such problems, we need to normalize.

The objective is to normalize *E(x,y)* in order to obtain a *similarity* value between 0 and 1, with the lower value, the more similarity. That is what is called *Normalized Information Distance* (NID) and is defined by Li et al. (2004)

$$e(x,y) = \frac{\max\{K(x \mid y), K(y \mid x)\}}{\max\{K(x), K(y)\}} \tag{3.6}$$

3.3 Normalized Compression Distance (NCD)

The Normalized Compression Distance (NCD) is the kernel of our document. We can define it as *the practical implementation of the NID*. From the moment that the NID is not computable (Maciejowski, 1979), it becomes impractical in order to obtain any kind of result.

Formalization

Normalized Information Distance depends on the incomputable function *K*, and this makes NID to become incomputable. First we observe that using *K(x, y) = K(xy)+O(log min{K(x),K(y)}* and property 3.3 we obtain

$$E(x,y) = \max\{K(x \mid y), K(y \mid x)\} = K(xy) - \min\{K(x), K(y)\} \tag{3.7}$$

up to an additive logarithmic term *O(logK(xy))* which we ignore.

Then we must try to remove all conditional terms, replacing them by non-conditional ones *K(x), K(y)* and *K(xy)*. This comes handy if we interpret *K(x)* as the length of the string *x* being maximally compressed. An obvious idea is to approximate *K(x)* with the length of the string *x* after being compressed with any real-world compressor. Any correct and lossless data compression program can provide an upper-bound approximation to *K(x)*, and most good compressors detect a large number of statistical regularities. Using the *Z* compressor (LZ, gzip, bzip2, PPMZ) and rewriting expression 3.6 with the numerator of 3.7, we obtain

$$e_z(x,y) = \frac{Z(xy) - \min\{Z(x), Z(y)\}}{\max\{Z(x), Z(y)\}} \tag{3.8}$$

where *Z(x)* denotes the binary length of the compressed version of the string *x* compressed with compressor *Z*. The better *Z* is the closer e_z approaches the normalized information distance, the better the results are expected to be.

Under mild conditions on compressor *Z*, the distance e_z is computable, takes values in [0, 1]. More formally, a compressor *Z* is *normal* if it satisfies the axioms

- $Z(xx) = Z(x)$ *and* $Z(x) = 0$ (identity)
- $Z(xy) \geq Z(x)$ (monotonicity)
- $Z(xy) = Z(yx)$ (symmetry)
- $Z(xy) + Z(z) \leq Z(xy) + Z(yz)$ (distributivity)

up to an additive *O(log n)* term, with n the maximal binary length of a string involved in the (in)equality concerned.

Informal experiments (Cilibrasi and Vitanyi, 2005) have shown that these axioms are in various degrees satisfied by good real-world compressors like bzip2, and PPMZ, with PPMZ being best among the ones tested.

4. Proposed approach

In this section we introduce the proposed approach to NCD application to any common search engine. We try to justify the development of a content-based search engine. *The goal of this proposal is to retrieve whole documents that are referred to a given query*. There are several observations that explain how to improve results in this type of search. Most important ones are the *database structure* and *ranking based on parameters*. But we also introduce new concepts like *information overlap* and *retrieval based on outlier removal*. All explanations are proposed in order of implementation. We conclude this section by giving a brief overview of the developed software in a real SQL and JAVA-based environment.

4.1 Database Structure

Our database structure is the consequence of the NCD definition and its practical properties. It has been proved that NCD gives better results when comparing information that has sense (Martinez et al., 2008). Let us remind the expression of NCD

$$NCD(x,y) = \frac{K(xy) - \min(K(x), K(y))}{\max(K(x), K(y))} \tag{4.1}$$

Suppose we are comparing a *classical* query with a whole document. Let us say that *length(x)* is much higher that *length(y)*. Note also in that both texts are written in *Natural Language* and contain a *common*[12] text. So we can conclude that

$$K(x) >> K(y)$$

If we combine x and y we can say that

$$K(xy) \approx K(x)$$

And obviously $\min\{K(x), K(y)\} = K(y)$ and $\max\{K(x), K(y)\} = K(x)$. Replacing in the expression 4.1 we obtain

$$NCD(x,y) \approx \frac{K(x) - K(y)}{K(x)} \qquad \text{and} \qquad K(x) - K(y) \cong K(x)$$

So, if y is enough smaller than x we get the following conclusion

[12] By common text we are meaning a non-repetitive document that has much other information that will be probably not related to the query introduced.

$$NCD(x,y) \approx \frac{K(x)}{K(x)} \approx 1$$

The above expression says that, given a big document comparing to the query, it does not matter their content, because NCD will give a value near 1. This says that query and document will be always *different* whatever their information is. But we want to be able to implement a search engine that is able to suite that limitation. In order to do that, we propose a particular database structuring. All documents are *splitted* in different sized blocks and organized in different sized databases.

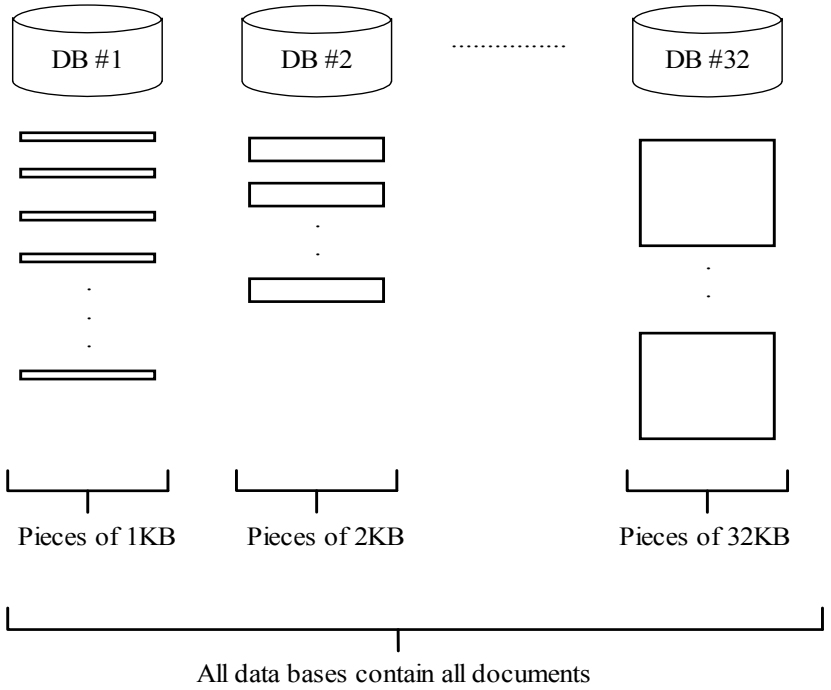


Figure 4.1- Database structure

We have defined 32 databases. Their structure is similar in all cases, but each DB can store different amount of information. *Database #1* can store blocks of text of 1KB long. *Database #32* can store blocks of text of 32KB long.

For each document we do the following:

1. Split it in blocks of 1 KB each one.
2. Store each block of text in the *1KB Database*.

…
(do the same, but splitting it in 2KB…32KB blocks)

What we obtain with this method is to have 32 replicas of each document, stored into different sized data bases. Figure 4.1 shows the structure of our database.

4.2 Information overlap

The goal of this concept is to provide better results for every block stored in the database. The main idea is to avoid loosing the *structure* of the document. Imagine we have the following two blocks of information that are partially related (NCD value is acceptable) to a given query, and both blocks belong to the same document and are adjacent. We have the following blocks

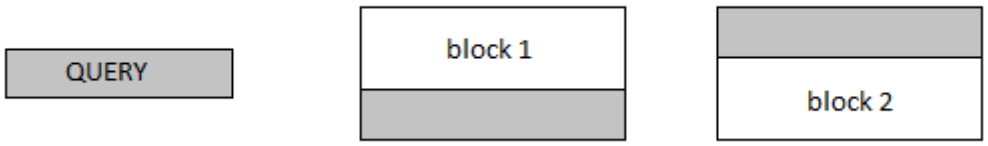


Figure 4.2- Block 1 and 2 share partially information with the query

As shown in Figure 4.2, both blocks share information with the query (highlighted area). We have seen before that all blocks were produced by the *splitting* algorithm and stored in the corresponding database. We might think that both combined blocks will give probably better results that any single one. Even better, we might think that is a good idea to merge only *shared* information into a single block. How can we do that? *Overlap information* between adjacent blocks. With this idea what we are doing is to replicate information, so we must be careful when selecting the *amount* of information we want to *overlap*. Figure 4.3 shows how overlap modifies the structure shown in Figure 4.2.

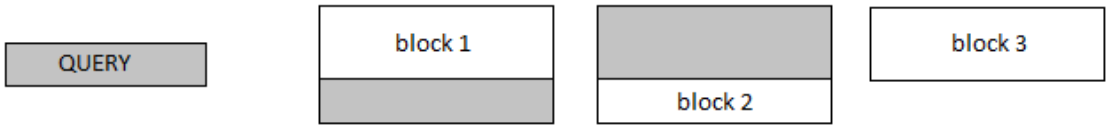


Figure 4.3- Structure after overlap

We can see that *shared* information has been merged into a *new* block. Intuitively the new block must have now better NCD (nearer 0) than former blocks 1 and 2, and better than new block 3. The information overlap can be decided at the *pre-processing* stage. Overlap is input as a percentage of each block. If we want to overlap 25% of each block, it means that 25% of block 1 will be added to the block 2. Block 2 then will be longer, so we must to reduce it. As block 1, the block 2 we will retain 25% of its information in order to add it to block 3, and so on. As you see, information is replicated. The higher the overlap is, the bigger the database is. The overlap decision must be an agreed value between performance, efficiency and effectiveness. If we take a value of 100% overlap, it will degenerate into an infinite *splitting* process. If we take a 0% value, we are meaning that documents will be strictly *splitted*.

Final database will contain blocks with shared information, organized by documents and sizes.

4.3 Outlier removal

The goal of this research is to justify the development of a content-based search engine. Freire et al. (2007) proved the power of the NCD applied to *plagiarism* detection. This is not the objective of this work. The main focus of our research is to provide most related documents, yet not identical ones. In order to address this issue we use a statistical feature of every dataset called *outliers*. After search process (see Section 4.4) we get a collection of NCD corresponding one-to-one with every single block of each database (this is not exactly as this, see *Note 1*). We want to detect which blocks can be detected as *identical* to the given query. The outliers of our set of results are considered as marginally identical and not taken into account in the retrieval results.

4.3.1 The Notion of Outlier

The result of performing similarity calculations is usually a large (quadratic in the number of submission in the corpus) number of pair wise distances, a number of which will be very low in case of similarity occurrence. The goal of an information retrieval tool is to flag these distances as the ones connecting identical blocks.

When based only on the bulk of distance values, the process of evaluation relevant candidates depends on the determination of a similarity threshold value. If a pair of documents have a similarity distances which falls below this threshold, it will be marked for inspection. Otherwise, the pair will not be marked.

However, the task of locating a good initial threshold has received little attention in the literature, and is therefore left completely to the personal decision of the search engine designer.

Under certain conditions, it is possible to quantify the amount of surprise presented by a distance value within a sample of distances. This problem is completely equivalent to the one of finding outliers in a data sample, a classical problem in statistical data analysis. Quoting Barnet and Lewis (1994)

> *"We shall define an outlier in a set of data to be an observation (or subset of observations) which appears to be inconsistent with the remainder of that set of data."*

The term *surprisingly small distances* is thus equivalent to the term lower outliers as used in statistics. In other words, the set of distances which should be marked as connecting identical blocks in a query corresponds to what a reasonable statistical test will consider as lower outliers within the full set of distances. As far as the authors are aware, this is the very first application of outlier detection techniques to Information Retrieval, despite its natural analogy.

4.3.2 Outlier detection method (Hampel identifier)

The outlier detection method consists of identify the *threshold* NCD value, below which data will be considered as *surprisingly* small respecting to the whole set of data. This

***threshold* is represented by α.** The way of finding outliers is by using the *Hampel identifier.*

The Hampel identifier is maybe the most extensively studied outlier-finding method for normally distributed data sets. Additionally, there is empirical data proving that it outperforms other tests in many applications (Pearson, 2002; Wilcox, 2003).

The Hampel identifier works as follows. Let *X(1),X(2), . . . ,X(N)* be the ordered distances X_1,X_2,X_3,X_N. Let *M* be the median of the sample, and *S* be the median absolute deviation form the median of the sample. The Hampel identifier, adapted to lower outliers, is a rule which identifies as relevant all distances of the sample *X* satisfying:

$$(M - X)/S > g(N;\lambda) \tag{4.2}$$

where the function $g(N;\lambda)$ serves for standardizing the identifier in the following way (see Davies and Gather, 1993, p. 783, standardization (4)):

$$\Pr(\textit{no outliers in sample}) = \Pr\left(\frac{|X_{(N)} - M|}{S} < g(N;\lambda)\right) = 1 - \lambda \tag{4.3}$$

The function $g(N;\lambda)$ does not have an analytic form and we estimate by means of a Montecarlo simulation. This method can handle a large number of outliers, and is resistant to the problems appreciated in non-parametric approaches (Freire et al., 2007). The designer may choose which threshold to use, being $\lambda = 0.01$ and $\lambda = 0.05$ the most usual ones.

4.3.3 Outlier removal from set of distances

The Hampel identifier is used in order to determine the NCD value that acts as threshold. This value, denoted by α is the first X_i value of the set of distances that satisfies

$$\Pr\left(\frac{|X_i - M|}{S} < g(N;\lambda)\right) = 1 - \lambda$$

Only data values above α will be considered for the retrieval stage, as shown in Figure 4.4. This NCD set extraction is explained in Section 4.4.

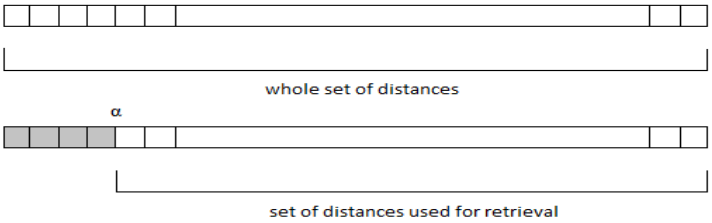


Figure 4.4- Set of distances used for retrieval after α-identification

4.4 Search method

First step for searching is having one *query* to find out (or find out similar ones) in the 32 databases. Search engine calculates all NCD values with all blocks of text stored in databases. So, it will give a set with size

$$|SET| \geq \sum_{i=1}^{N} \sum_{j=1}^{32} \frac{length(i)}{j}$$

up to the *overlap* parameter, with N being the number of documents we have in our dataset.

All we have to do is order each NCD value.

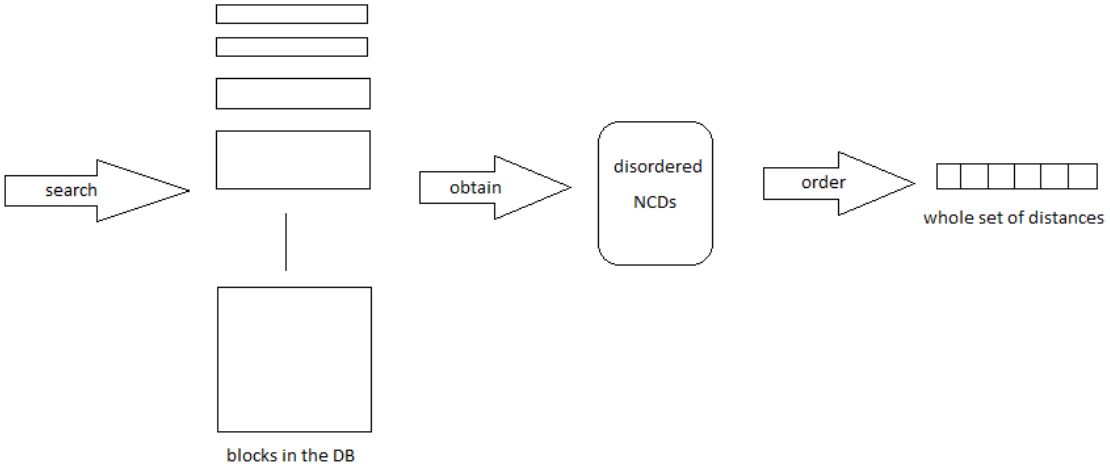


Figure 4.5- Process to obtain the set of distances

Note 1: Due to our experience and NCD properties, we have concluded that best NCD's are the ones that match the *query* length with similar sized blocks of text. So, a better way to reduce time of processing, is avoiding search all over the 32 data bases, and search only from $querySize(KB)-1$ to $querySize(KB)+2$ databases.

4.5 Parameterized estimation for ranking method

After retrieving the whole set of calculated distances and removal of outliers, we have a set of distances that must be studied to analyze which results fit best our needs and how can they be useful for ranking purposes.

Intuitively, all distances near to 1 are candidates to leave the ranking process. We will see in experimental (Section 5) that this supposition is true. We might also think that values of NCD near to α are the best ones for ranking.

Another fact to have into account is the size of the retrieved blocks. As seen in section 4.1, blocks of similar length to the *query* size fit better the NCD. Another assumption should be that best NCD's will be those that are of the similar size than the given query. However the assumptions are, the parameterized analysis of NCD's will tell us the best way to obtain best values for a given domain or set of documents. Let us define the proposed *parameters* for ranking estimation:

> **Definition 1:** (*Beta* β) is the value that denotes the upper bound subset of distances that fits best the ranking for a given set of documents. It is measured in terms of percentage of the total set of distances.

Given a set of distances extracted in the search method, the β value is the percentage of the whole set of distances counted from 0 (α threshold is taken as 0%). A notion of β is represented in Figure 4.6.

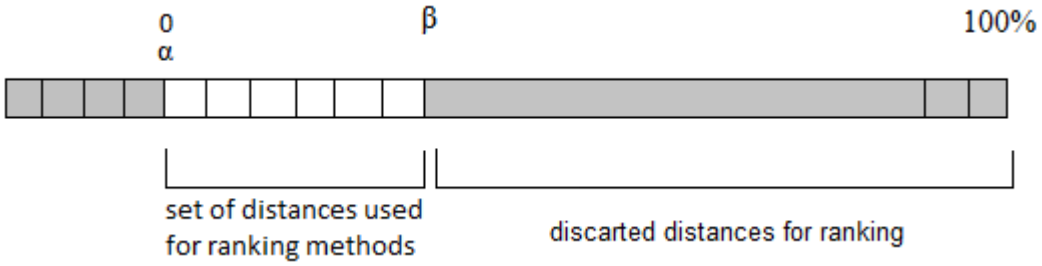


Figure 4.6- β-value selection over the set of distances

The next step is to analyze if the block size has impact over ranking results (in terms of *precision* and *recall*). As we see in Section 5, the size of the block has a lot to do with better results, so we can define it

> **Definition 2:** (*Theta* θ) is the size of a retrieved block. It is measured in kilobytes (KB).

Theta value has sense in order to determine which block size fits best the ranking. Suppose we have retrieved a set of distances corresponding to blocks (after beta discrimination), which sizes are

[1,2,2,4,6,6,6,6,6,6,6,7,8,20]

We might think that blocks of size equal to 6KB (θ=6) have more probability of being positives that blocks of size 2KB, for example (we must define which is positive and which is negative in order to say that θ=6 is really better than θ=2).

As defined above, β has to do with NCD values and θ has to do with block sizes (note that each block has each corresponding NCD value). An experimental agreement between β and θ must be made in order to get best values of each to retrieve the best ranked list possible. These results will be given in the Experimental Section 5.

4.6 Evaluating effectiveness: The Hypothesis Matrix

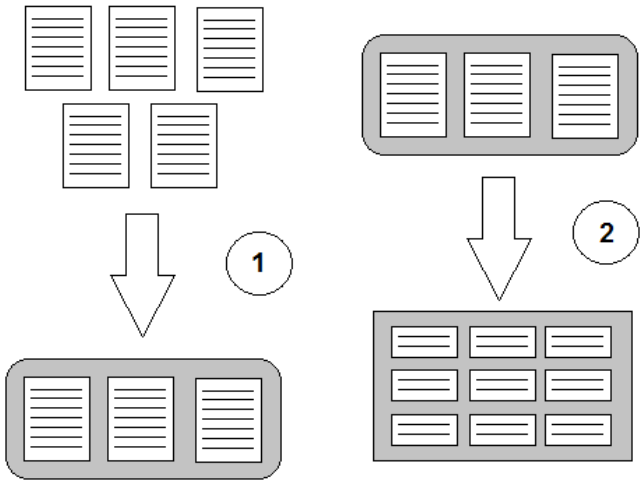
To extract results for the proposed approach, we must provide an evaluation method in terms of *precision* and *recall*. A relevance judgment is required in order to identify *positives* and *negatives* (see Section 2.5) from the search engine.

Due to NCD clustering properties (Cilibrasi et al., 2005) it become interesting to analyze results in terms of *clustering effectiveness*. For that matter we have implemented a *hypothesis matrix*.

> **Definition 3: (***The hypothesis matrix*) contains all blocks stored in the database that are supposed to be considered as *true positives* for a given category.

In order to generate the hypothesis matrix we must configure a set of documents which are representative for each category. Such documents are then used to be the *training set* for the hypothesis matrix. Documents are used as queries and results will compound the hypothesis matrix. The process is the following:

1. Review the documents from a category and choose the most *representative* ones.
2. [Optional] Select blocks of those representative documents as *representative blocks*.
3. Use selected documents (sets of blocks) as queries for the search engine.
4. Retrieved blocks from search engine are considered as *hypothetical true positive* blocks.
5. Store each feature of those retrieved blocks for generating the hypothesis matrix.



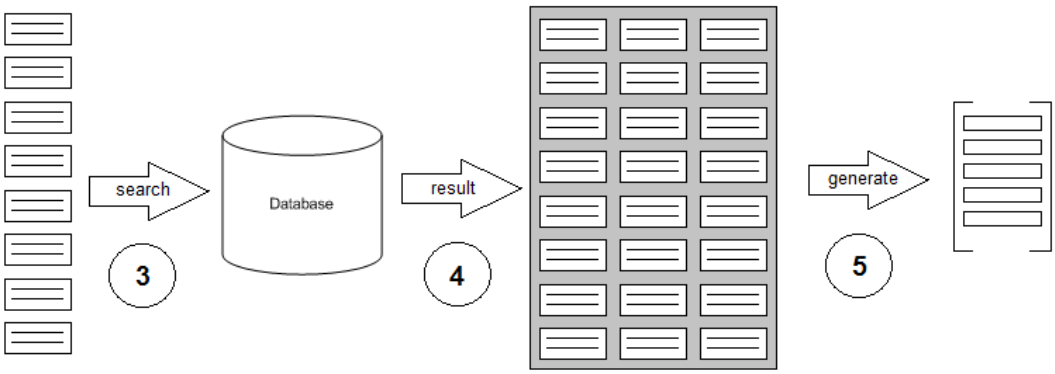


Figure 4.7- Hypothesis matrix generation process

The *hypothesis matrix* is a matrix (vector) of size *Nx1* where *N* is the number of documents available in the database. Each row of the matrix contains a tuple:

{ [id , blocksize0 , blocknumber0] … [id , blocksizeM , blocknumberM] }

Where:

- *Id* is the document identifier which the block belongs to.
- *Blocksize* is the size of the retrieved block (in KB).
- *Blocknumber* is the position of the document in the document.
- *M* is the number of blocks retrieved from the document identified by *id*.

Example: An example of hypothesis matrix can be

$$\begin{bmatrix} \{[1,1,2][1,2,1][1,2,2][1,2,3][1,3,3]\} \\ \{[2,1,1][2,1,2][2,1,5][2,2,1][2,3,3][2,4,1]\} \\ \{[3,1,2][3,2,2][3,2,4][3,2,5][3,2,6]\} \\ \{[4,1,1][4,2,1][4,2,2]\} \end{bmatrix}$$

This matrix contains the information about hypothetical relevant blocks for a given domain or category.

- Relevant blocks of *DocumentID=1* are
  - Block no.2 of 1KB long; Block no.1 of 2KB long; Block no.2 of 2KB long; Block no.3 of 2KB long; Block no.3 of 3KB long;
- Relevant blocks of *DocumentID=2* are
  - Block no.1 of 1KB long; Block no.2 of 1KB long; Block no.5 of 1KB long; Block no.1 of 2KB long; Block no.3 of 3KB long; Block no.1 of 4KB long;
- Relevant blocks of *DocumentID=3* are
  - Block no.2 of 1KB long; Block no.2 of 2KB long; Block no.4 of 2KB long; Block no.5 of 2KB long; Block no.6 of 2KB long;
- Relevant blocks of *DocumentID=4* are
  - Block no.1 of 1KB long; Block no.1 of 2KB long; Block no.2 of 2KB long;

The hypothesis matrix contains information of representative blocks, and database contains 4 documents.

After generating the matrix, we store it in memory (i.e. hard disk) and it will act as a reference for future searches.

When a query is introduced to the search engine, it is compared with the blocks stored in the database (all) and *only* blocks that are present in the hypothesis matrix are considered as positives. Let us call *H* to the blocks stored in the hypothesis matrix, let *D* the retrieved blocks from the database and *R* the result set,

$$R = D \cap H$$

In order to evaluate the effectiveness of our result set *R*, we compare every single block and see if it belongs to the supposed category. Overall precision and recall are then calculated in two ways, the classic way or a granular way.

Classic way: (Precision and recall are related to documents)

$$Precision = \frac{|\, doc(R^+)\,|}{|\, doc(H)\,|} \qquad Recall = \frac{|\, doc(R^+)\,|}{|\, doc(D^+)\,|} \tag{4.4}$$

Granular way: (Precision and recall are related to blocks)

$$Precision = \frac{|\, R^+\,|}{|\, H\,|} \qquad Recall = \frac{|\, R^+\,|}{|\, D^+\,|} \tag{4.5}$$

Where:

- doc(A) is the number of *different documents* that the A set blocks' belongs to.
- A^+ is the subset of *positive* blocks of the set A.

Both precision and recall can also be calculated in terms of β and θ in the following way:

$$P^C{}_{\beta,\theta} = \frac{|\, R_{\beta,\theta}{}^+\,|}{|\, H_{\beta,\theta}\,|} \qquad R^C{}_{\beta,\theta} = \frac{|\, R_{\beta,\theta}{}^+\,|}{|\, D^+{}_{\beta,\theta}\,|} \tag{4.6}$$

$$P^G{}_{\beta,\theta} = \frac{doc(R_{\beta,\theta}{}^+)}{doc(H_{\beta,\theta})} \qquad R^G{}_{\beta,\theta} = \frac{doc(R_{\beta,\theta}{}^+)}{doc(D^+{}_{\beta,\theta})} \tag{4.7}$$

Where:

- P^C stands for *precision* in classic way
- R^C stands for *recall* in classic way
- P^G stands for *precision* in granular way

- R^G stands for *recall* in granular way
- $A_{\beta,\theta}{}^+$ stands for the subset of positive documents at β threshold and θ size of the block. In other words, only blocks of size θ and present in the set A are considered.

These measures are used to evaluate the real effectiveness of our search engine. Note that, at last, human interaction is needed in order to provide real relevance judgment (R^+ and D^+ can only be calculated by expertise).

4.7 The application

We have developed a suitable software application in order to address the above concepts in experimental. This software consists of three stages: *pre-process*, *search* and *experimental*. The first two ones are implemented in the same application, while the third stage has been implemented separately.

Every application is based on the same *core* software. That *core* contains all methods, algorithms and functions used to search over the database. It supports the implementation and communication of any API-based *Graphical User Interface*.

The *core* software is based on JAVA technology. It supports *pre-processing* and *searching* functions. It also provides SQL-based functions to communicate with the database. It only supports documents in *plain text* format.

The database is intended to be any SQL. In this case we have chosen *MySQL* due to facilities.

In order to run the software appropriately we must do, at least, the following things:

1. Define a Graphical User Interface that must be able to communicate the *core* software with the provided API.
2. Choose the set of documents that the database must contain.
3. Convert every document to *plain text* format.
4. Run the *pre-proccess* method of the software.

Once we have done the above steps, we have our set of documents *splitted* and stored in our SQL database. Then, we can choose to make two things: *search* or *experiment*.

Search

We provide a GUI to make simple proofs with the search engine. It contains required fields for the *query input*, and one are to see results of NCD calculations. It also contains an area that is intended to show final results (not NCD but documents) based on any *ranking* algorithm that the user can define. In Figure 4.8, area 1 corresponds to query input, area 2 contains NCD results and area 3 contains the ranking. Area 4 provides an overview of any clickable result of areas 2 and 3 with highlighted information.

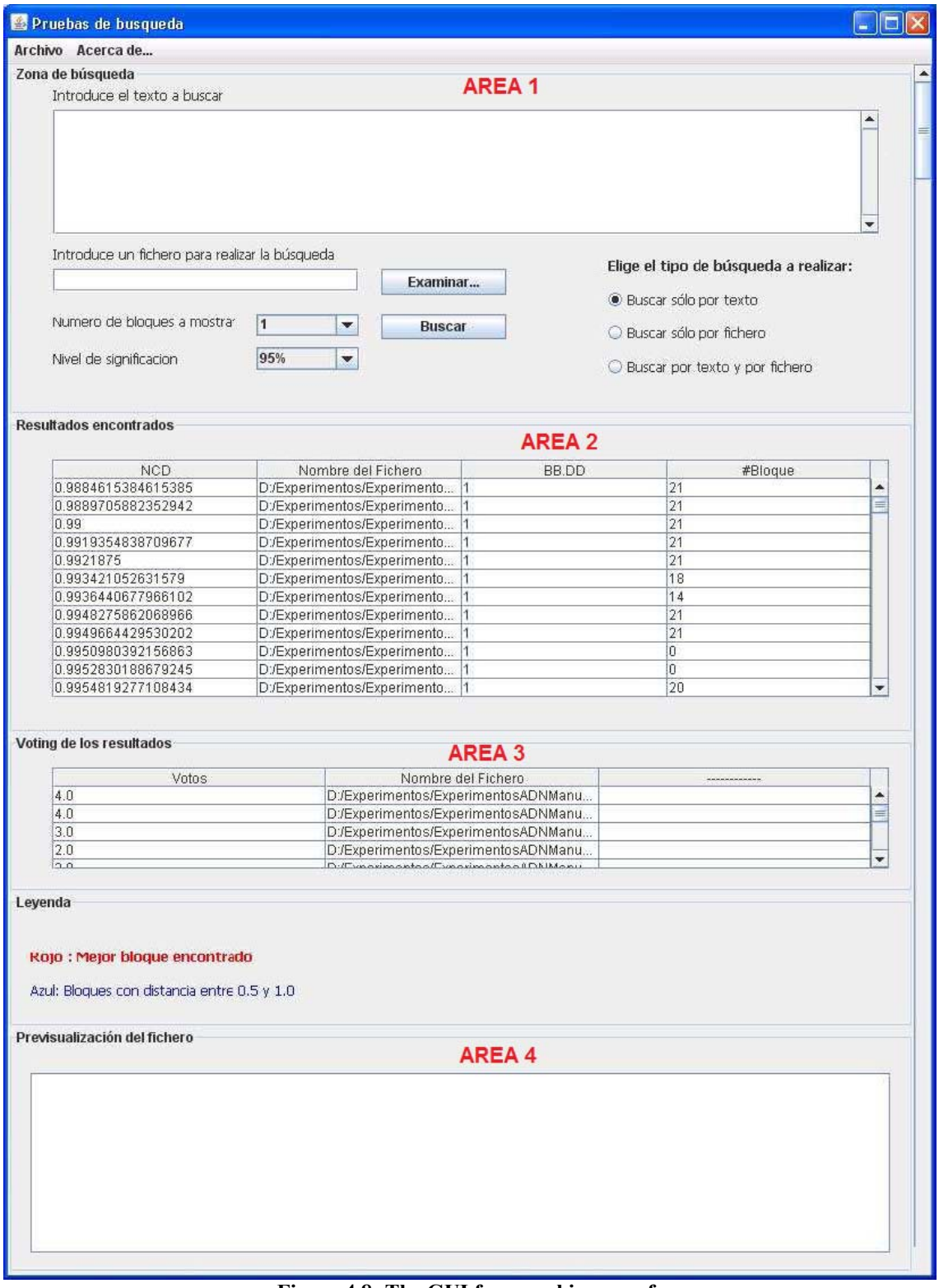


Figure 4.8- The GUI for searching proofs

Experiment

We also provide a simple GUI for test purposes. It contains all needed methods for the hypothesis matrix generation, and β,θ experiments. However, the best experiment tool is the one that any user can define for his test purposes.

5. Experimental

We have developed 3 kinds of experiments. Each one tries to analyze a different fact. Experiments evolve from the very first results to β and θ estimations. The first experiment evaluates the overall *precision* of the simple searching process. This precision is calculated in the *classic* way. The second experiment introduces β and θ estimations for a short set of queries. Finally, we introduce a larger set of queries and a different dataset in order to accurate results. *LZ* is the compressor used in this set of experiments. Datasets and queries for each experiment are defined in Appendix A.

5.1 Experiment no. 1

For this experiment, we have generated a small dataset of 100 computer-science-oriented documents. We have selected a query from a category which contains 19 documents. This experiment extract every document that retrieved blocks belongs to. In other words, we count different documents that are represented by the blocks. We analyze the precision's behaviour in respect of beta percentage of retrieval. In this experiment we do not introduce the *hypothesis matrix*, just analyze results by themselves.

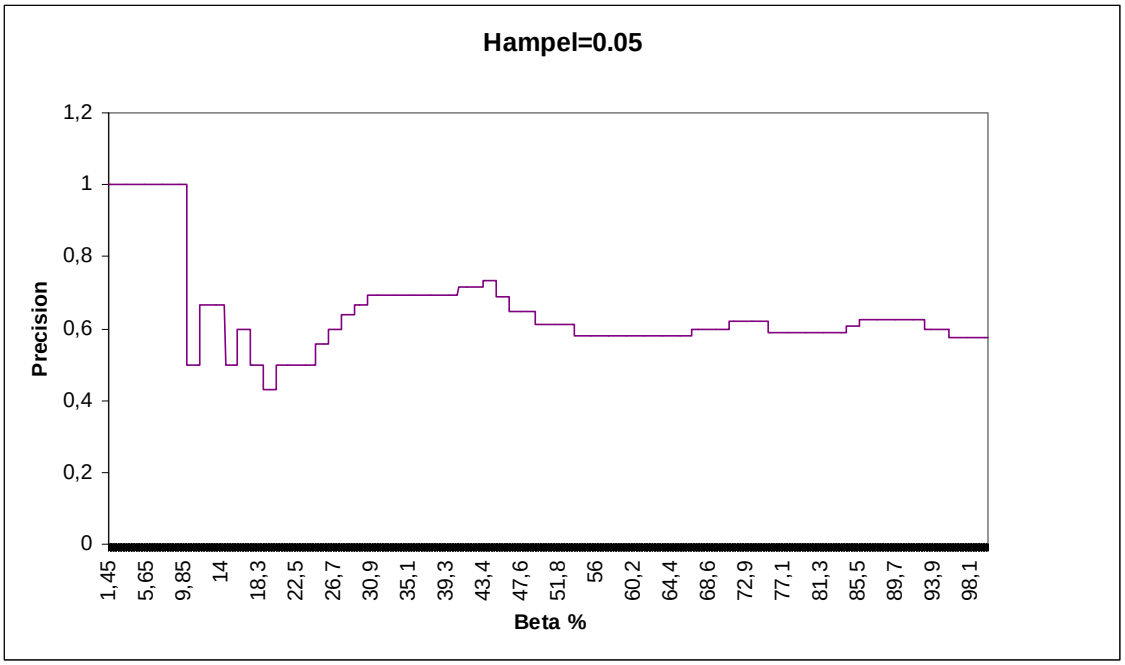


Figure 5.1- Precision at every beta level

Precision is calculated in a *classical* way. *Hampel* identifier for outlier removal is defined at $\lambda = 0.05$. We observed good precision behavior when β is near the α outlier removal threshold, as our intuition said. At very low values of beta, retrieved blocks *only* belong to documents from the specified *category*.

We can not generalize these results, because a single retrieved block from one document does not say that the whole document is related to that *query*. We must examine the amount of blocks retrieved from each document. Intuitively, if we retrieve 10 blocks from a document is more useful that if we retrieve 5.

5.2 Experiment no. 2

In this experiment we introduce the *hypothesis matrix* to accurate precision values. This time we have selected 6 queries from documents of the same category to provide *averaged* results and not just one single query. The dataset used is the computer-science-oriented one. X axis represents the θ value and Y axis shows the amount of blocks of size θ retrieved (in percentage of all blocks retrieved). We split results in *true positives* and *false positives*. By true positives we mean positive blocks that correspond to a positive expertise judgment (and not simply by the hypothesis matrix). The opposite are called false positives.

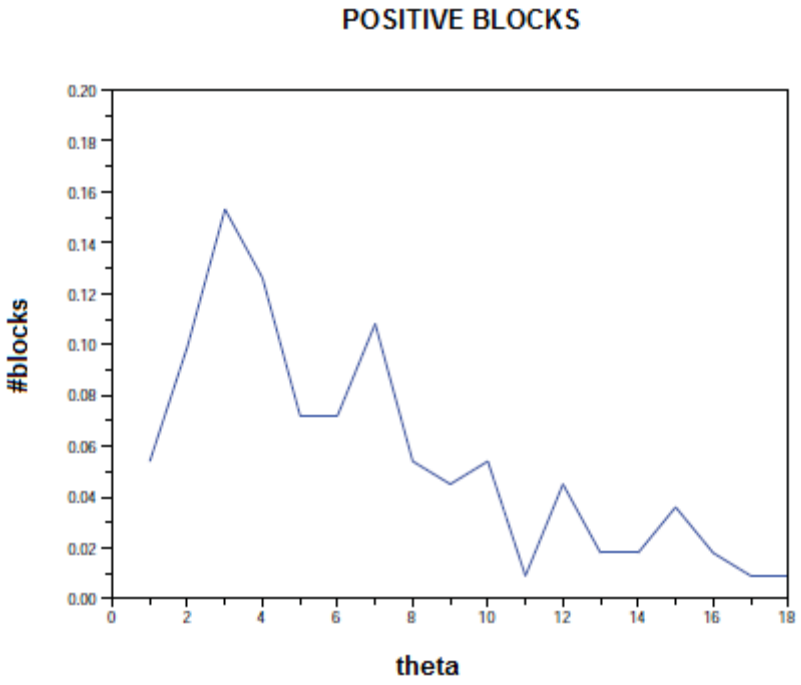


Figure 5.2- True Positive blocks distribution at β=4%

In Figure 5.2 we observe that most blocks retrieved are of a small size (from 1 to 7 KB) when query is 1KB long. But it is also interesting that bigger blocks (from 8KB) are retrieved and classified as positives by expertise. As we see in Section 4.1, better results are usually the ones which length is similar to the query. So, retrieving big blocks has a particular positive effect over the search effectiveness.

It becomes interesting to analyze the distribution of precision of the true negative blocks in order to determine if big blocks retrieved are negative in any case.

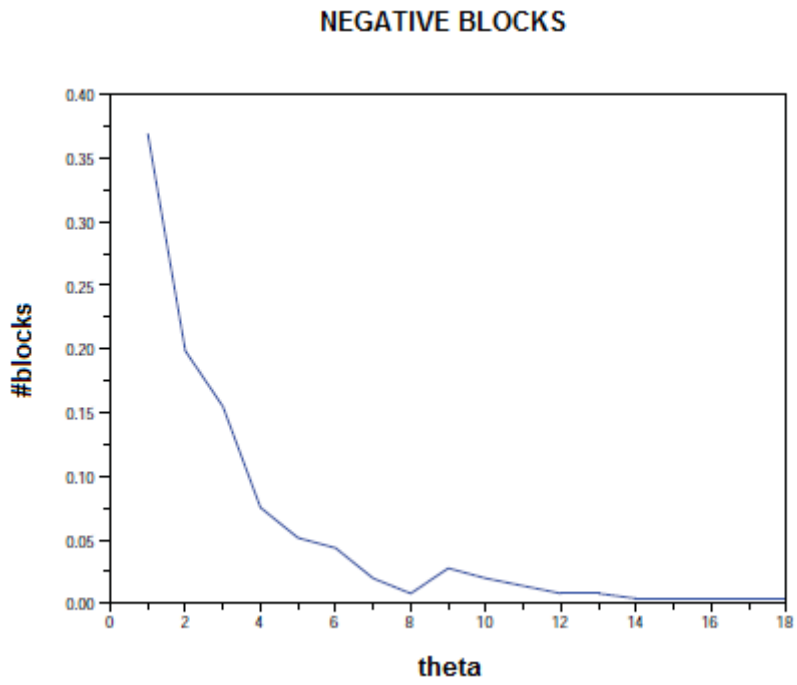


Figure 5.3- False Positive blocks distribution at β=4%

Figure 5.3 shows that big blocks are rarely retrieved as false positives, while small blocks (from 1 to 4KB) have a high probability of being considered as negative. Figure 5.2 shows that 5% of blocks are 1KB long and true positives, while Figure 5.3 shows that 35% of blocks are 1KB long and false positives. So, $5/40 = 12{,}5\%$ of retrieved blocks for θ=1 will be considered as positives.

We aim to find any θ value (if exists) for which precision are reasonable for *Information Retrieval* purposes. Any improvement from 50% begins to be acceptable; however a higher value (around 80%) is required to be considered as good as most TREC systems.

5.3 Experiment no. 3

The goals of this experiment are mainly two:

- Smooth curves
- Clarify possible θ values for which precision saturates.

In order to address the first issue, we provide a higher set of documents and a higher set of queries to search. For the second issue we extract a cumulative curve from the distribution ones.

In this case we select a dataset of 200 documents related to several topics as technical sciences, biology or physics. The positive category will be differentiated from the other ones. In this case, the category to look for is related to religion.

We prepare a set of 20 queries. Five of them are used to generate the hypothesis matrix, while the rest 15 are used to search experiment. All queries are not larger than 2KB. The following results have been obtained (for both *true positive* and *false positive* results).

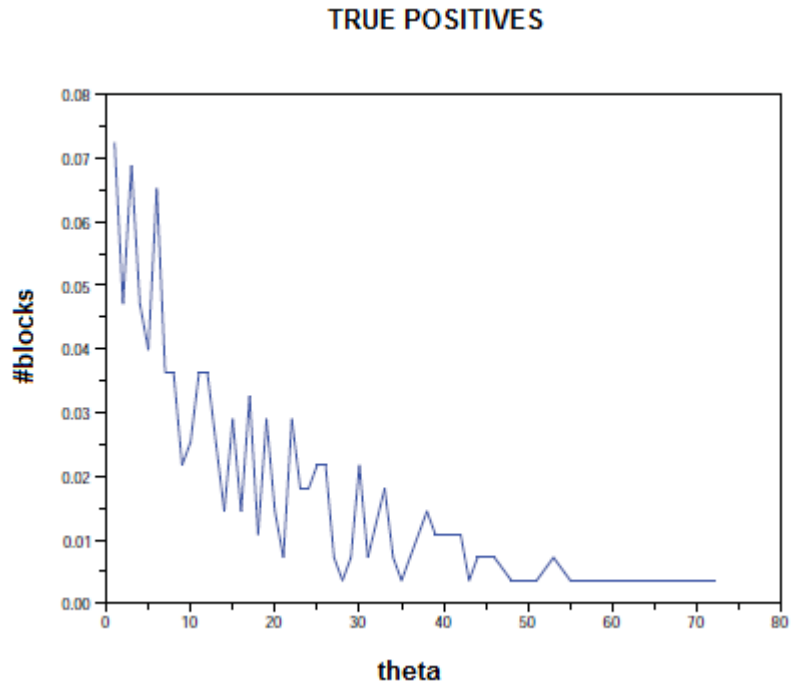


Figure 5.4- Large experiment. True positive blocks distribution at β =4%

This distribution is similar to Figure 5.2 except of oscillations. These oscillations must not be taken into account due the block-oriented retrieval. A false positive analysis is also required for comparing respect to true positives.

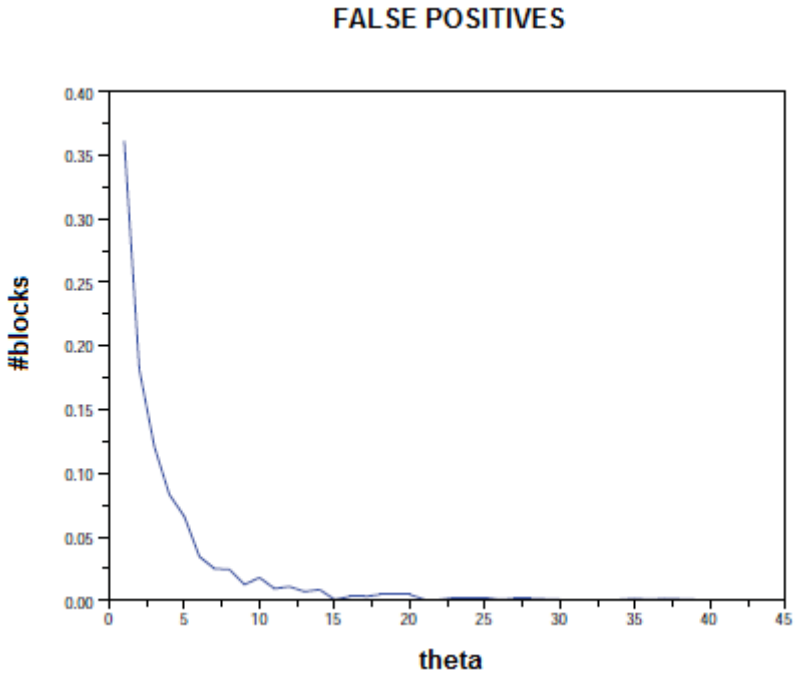


Figure 5.5- Large experiment. False positive blocks distribution at β =4%

As seen in Figure 5.3, false positives distribution is also eloquent. We can observe that big blocks (over 7KB) are rarely detected as false positives, while small blocks (up to 4KB) continue being considered false positives in many cases. A cumulative accuracy figure is provided in order to clarify such suppositions.

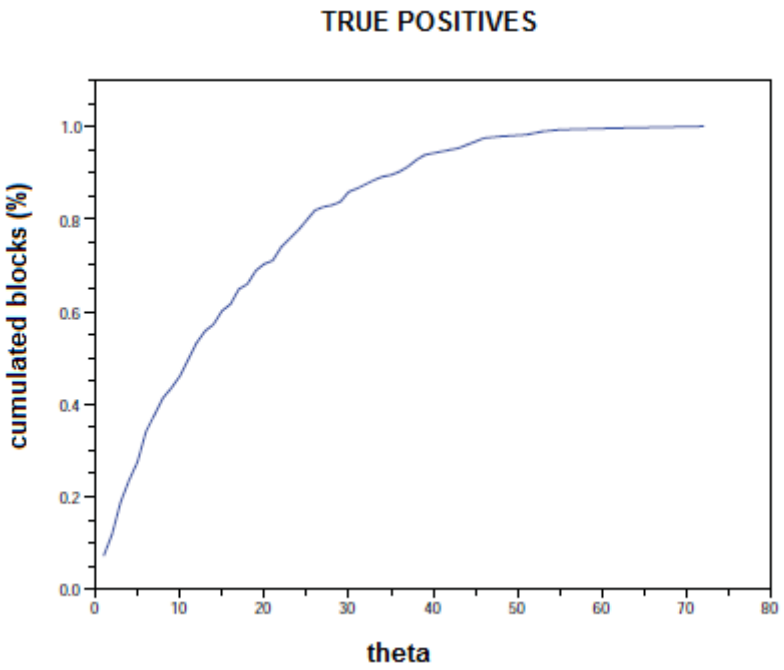


Figure 5.6- Large experiment. True positive blocks cumulative distribution at β =4%

In Figure 5.6 we conclude that θ value begins to be significant from 15KB (with 60% of accuracy approx.). If a retrieved block has a size over 30KB, we can surely conclude that this block will be true positive (in more than 85% of cases). An opposite view can be taken from the true positive cumulative distribution.

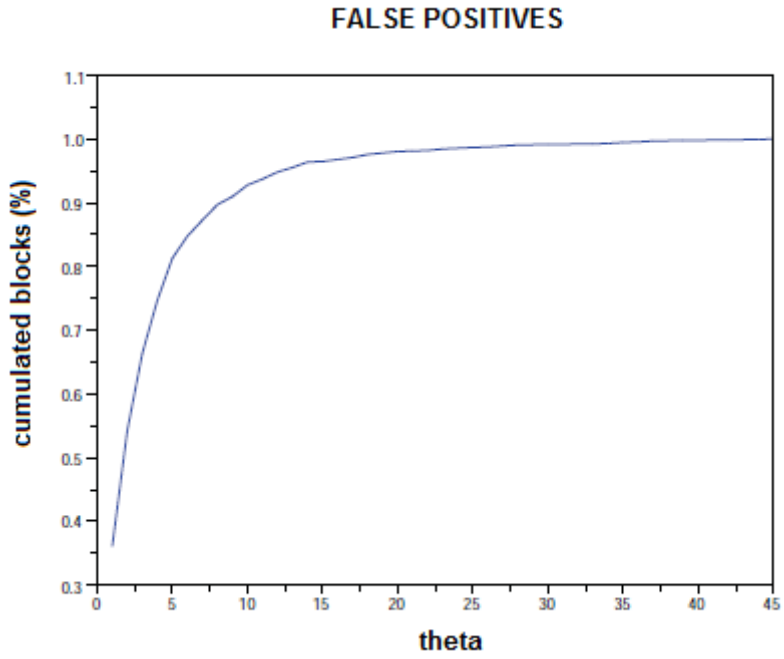


Figure 5.7- Large experiment. False positive blocks cumulative distribution at β =4%

This figure is even more significant. By our experience, almost any block of size $\theta = 7$ or less will be considered in most cases (more than 85%) as negative. In the other hand, blocks of size $\theta \geq 15$ will never (or almost never) be considered as false positives.

5.4 Experiment no. 4 (β and θ analysis)

In the above experiments we have only analyzed results of a single and representative value β in order to extract conclusions about θ. It would be a good idea to put all β and θ results together to have a better vision of their behaviour. The following table represents results for a 10-query (around 1KB long) experiment over the computer-science oriented dataset in terms of β and θ.

θ \ β	1	2	3	4	5	6	7	8	9	10	AVG
1	0,491	0,437	0,310	0,173	0,102	0,054	0,043	0,043	0,028	0,017	0,170
2	0,850	0,542	0,408	0,315	0,289	0,187	0,116	0,063	0,044	0,023	0,284
3	1,000	0,708	0,556	0,461	0,322	0,284	0,218	0,184	0,136	0,083	0,395
4	1,000	0,692	0,581	0,548	0,460	0,373	0,250	0,219	0,195	0,132	0,445
5	1,000	0,692	0,625	0,529	0,500	0,465	0,308	0,293	0,141	0,165	0,472
6		1,000	0,700	0,579	0,600	0,342	0,519	0,355	0,288	0,304	0,521
7		1,000	0,706	0,600	0,556	0,575	0,391	0,400	0,412	0,164	0,534
8		0,000	0,750	0,833	0,429	0,789	0,594	0,389	0,357	0,364	0,501
9		0,000	0,900	0,636	0,588	0,542	0,607	0,483	0,433	0,395	0,509
10		1,000	1,000	0,714	0,667	0,520	0,571	0,455	0,455	0,421	0,645
11	1,000	1,000		0,857	0,900	0,667	0,611	0,621	0,435	0,379	0,719
12			0,000	0,833	0,750	0,769	0,700	0,750	0,524	0,520	0,606
13			0,000	0,600	0,857	0,778	0,538	0,636	0,688	0,600	0,587
14				1,000	0,571	0,500	0,500	0,538	0,706	0,600	0,631
15			1,000	1,000	1,000	0,800	0,636	0,778	0,529	0,500	0,780
16					0,500	0,800	0,750	0,500	0,667	0,933	0,692
17			1,000	1,000	0,750	0,750	0,667	0,778	0,769	0,714	0,803
18		1,000				0,750	0,800	0,500	0,333	0,533	0,653
19			1,000		1,000	0,000	0,800	0,857	0,625	0,375	0,665
20	1,000			1,000		1,000	0,500	0,625	0,667	0,750	0,792
21						1,000	1,000	0,000	0,833	0,667	0,700
22							1,000	1,000	0,667	0,750	0,854
23					1,000		1,000	1,000	0,800	0,333	0,827
24						1,000		0,667	1,000	0,800	0,867
25			1,000			1,000		0,667	0,333	0,500	0,625
26						1,000	1,000	1,000	1,000	1,000	1,000
27		1,000					1,000	1,000		0,500	1,000
28				1,000							1,000
29								1,000		0,000	0,500
30				1,000					1,000	1,000	1,000
31									1,000		1,000
32					1,000						1,000
33					1,000			1,000			
34									1,000		1,000
35						1,000				1,000	1,000

36							1,000				1,000	1,000
37				1,000							1,000	1,000
38												
39					1,000			1,000				1,000
40												
41								1,000	1,000			1,000
42										1,000		
43												
44						1,000						1,000
45											1,000	1,000
46									1,000			1,000
47							1,000			1,000		1,000
48												
49												
50								1,000				1,000
51									1,000			1,000
52										1,000		1,000
53											1,000	1,000
54											1,000	1,000
	AVG	0,906	0,698	0,734	0,734	0,675	0,665	0,647	0,619	0,596	0,558	

Let us call $p_{\beta,\theta}$ to a cell value. The *p* value determines for that beta/theta values, the number of true positives retrieved (blocks) in respect of all retrieved blocks. We consider that a block is *true positive* if it belongs to a *positive* document.

Intuitively, the optimal value for $p_{\beta,\theta}$ is anyone that $\theta > 26$ (where $p_{\beta,\theta} = 1$). But having a deeper view to results over $\theta > 26$, we observe that $p_{\beta,\theta}$ is inaccurate due to the number of retrieved results for that theta value. In most cases, for a very-high theta value, we have a very few results. For example, when $\theta = 30$, we have $p_{\beta,30} = 1$ for every β. Those values stand for 1/1, 2/2 and 3/3. We have not enough statistical data in order to ensure that for every $\theta > 30$ in whatever other experiment will be positive in all cases, since the number of blocks retrieved are 3.

In the other way, when θ is low, we have enough statistical data to ensure that if θ=1, averaged precision will be around 0.170.

As a conclusion we can say that we need enough blocks to get an accurate analysis and approximation to β, θ parameters. For this experiment, we can estimate that optimal parameters are $\beta = [0-4]$ and $\theta \geq 10$. However more experiments are required.

Figure 5.8 represents the number of blocks that are represented for every β value. As we can see, the lowest θ values are contained y lowest β. That is a refutation of our intuition in 4.1 (the lower the length is, the better NCD we obtain). Note that lowest β contains the best NCD values.

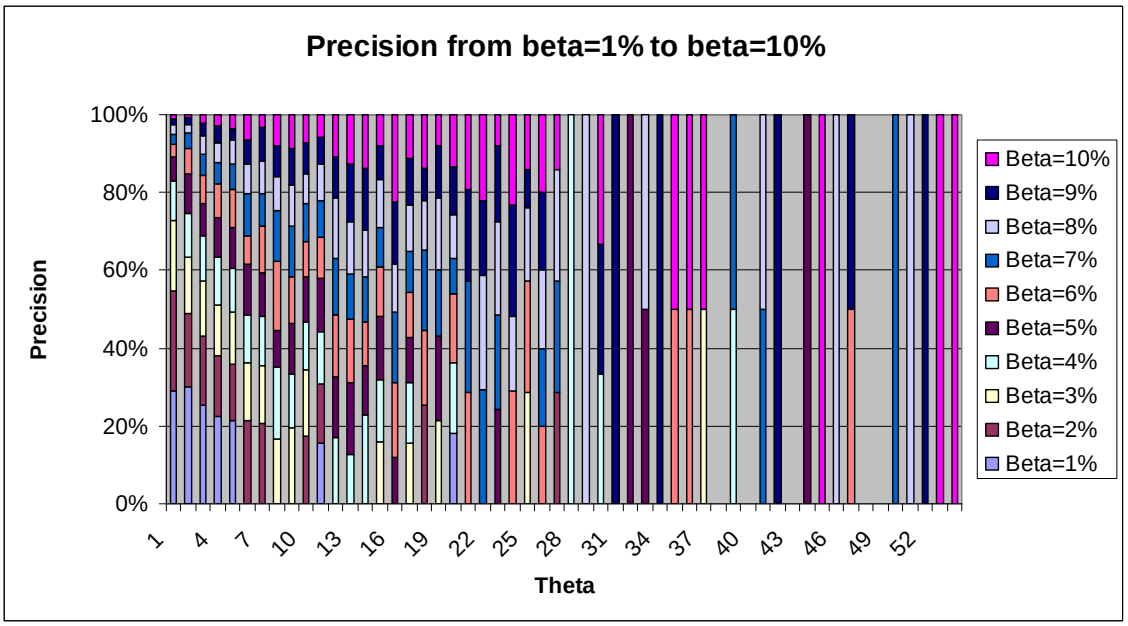
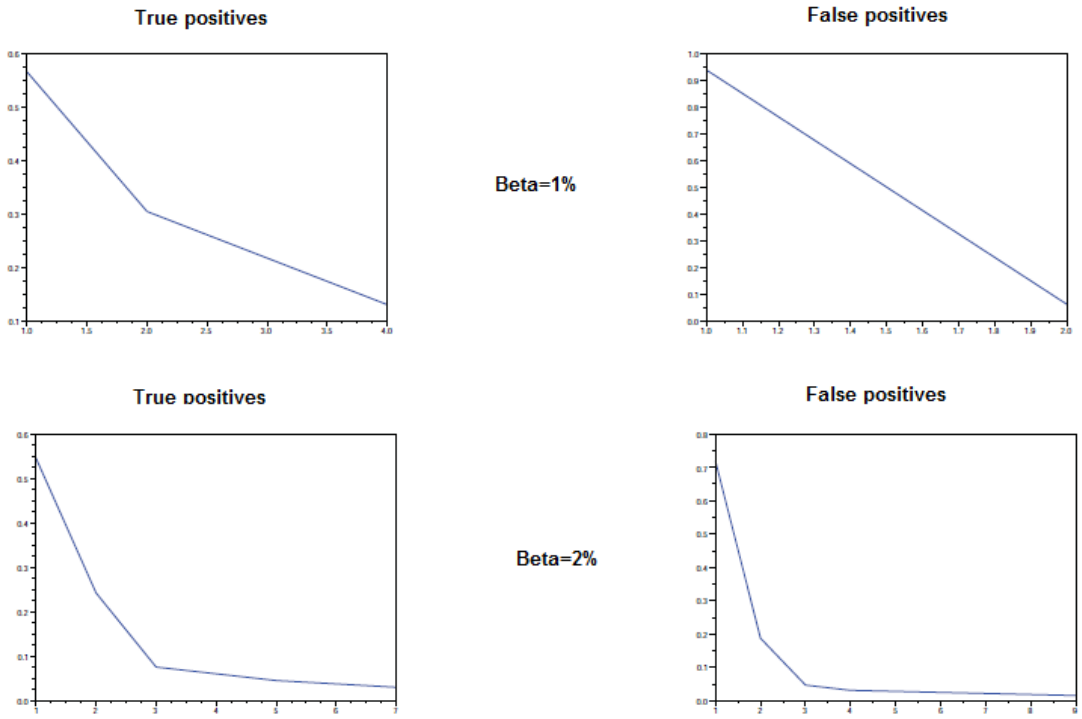
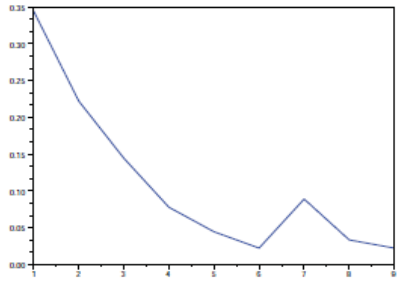


Figure 5.8- Theta proportion in every beta from 1% to 10%

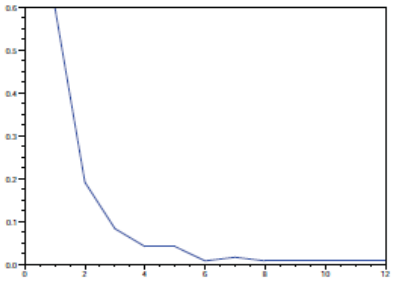
We provide a full set of figures for every single β value and for both *true positives* and *false positives* distribution.



True positives

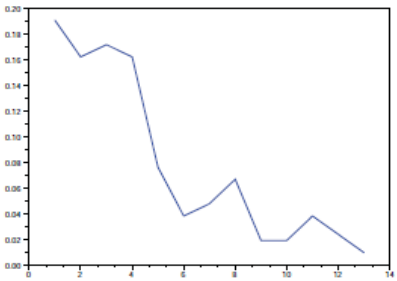


False positives

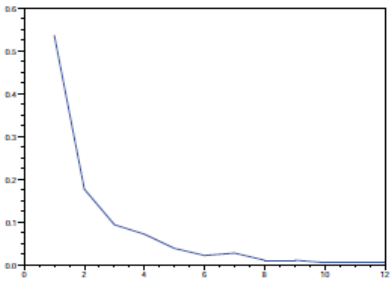


Beta=3%

True positives

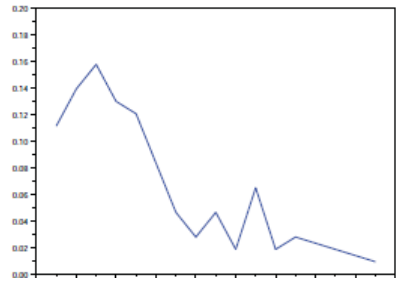


False positives

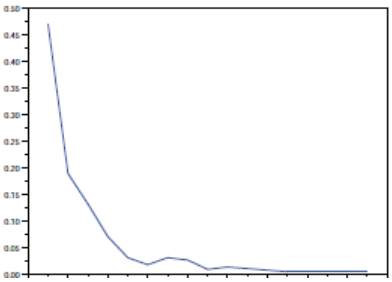


Beta=4%

True positives

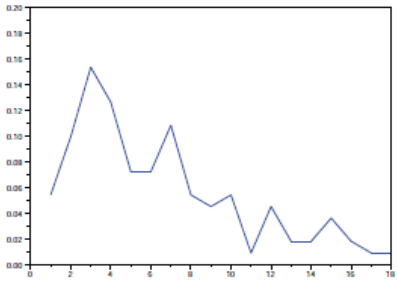


False positives

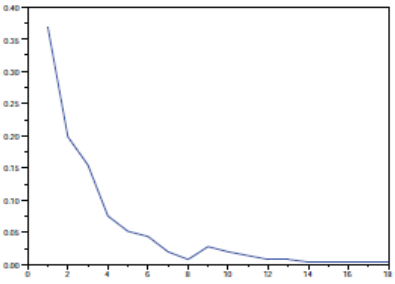


Beta=5%

True positives

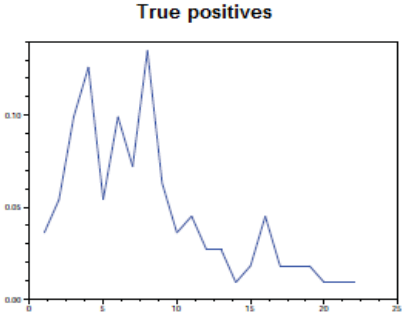


False positives

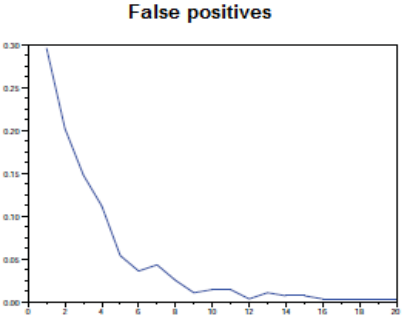


Beta=6%

True positives



False positives



Beta=7%

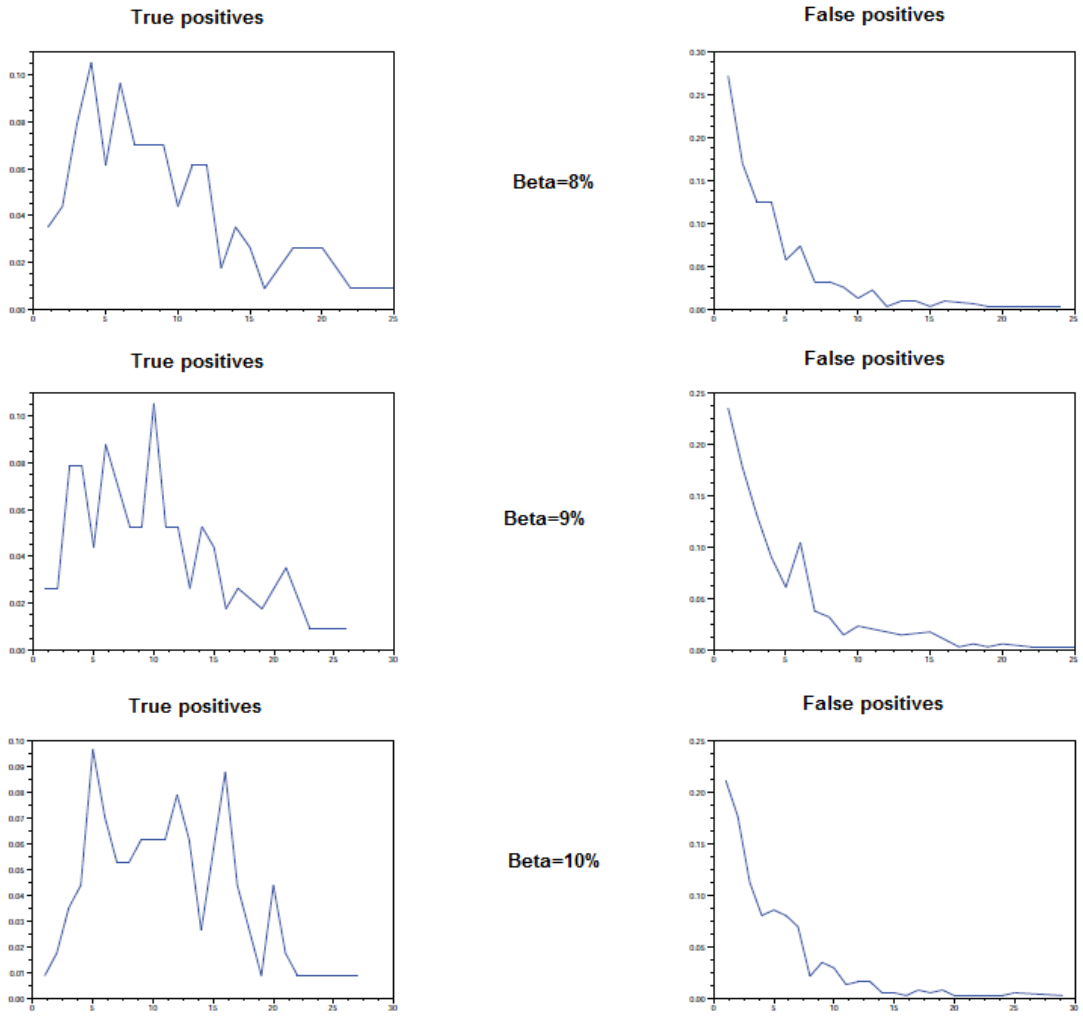


Figure 5.9- Experiment no.4. True/False positive blocks distribution from β =1% to 10%

5.5 Experimental conclusions

As we have seen in the related experiments, it becomes interesting to analyze the behaviour of θ in true positive and true negative blocks. Probably θ values will depend on the domain or the amount of information. False positive results are more eloquent than true positive ones. But we know that ones are the opposite of the others, so they can also be analyzed for search engine effectiveness.

By these experiments we can conclude that the bigger the block is the higher probability it has to be considered as positive. Intuitively range of positive accuracy stands in the range $\theta = [7-15]$ while the range of accuracy for false positives is $\theta = [0-15]$. In the other hand, we can observe that optimal amount of results to be analyzed must be in the range $\beta < 5\%$. Further conclusions can not be taken since more exhaustive experiments in different domains are required.

Note: All queries and datasets are explicited in Appendix A.

6. Conclusions and Future Work

We have concluded that applying Information Distance to Information Retrieval has sense not only for clustering but also for searching. Using the LZ compressor we obtain reasonable buy yet not very concluding results. At first stages we supposed that best results would be the ones where both query and retrieved block have similar size; but experimental proofs demonstrated that, being good results, they provide a huge amount of imprecision due to NCD properties (see Section 4.6). For that matter we tried to address the problem by means of quantity of information retrieved and size of that information (beta and theta thresholds).

Efficiency has been proved to be a critical fact in our approach. We have not covered efficiency terms for our tool, yet we have experimented slowly search processes (increment by means of the size of database) and a big load of memory (around 400MB). The architecture of the provided solution for NCD proofs is not the best and the dataset of documents and relevance judgment are also improvable with more expertise knowledge. Efficiency has been an obstacle in order to increment the amount of experiments.

This work, then, has a large way of study. It can be combined with other techniques like NLP or WordNet aid in order to facilitate the compression process which is tightly related with efficiency. NCD can also be used with other compressors like *gzip*, *bzip* or *PPMZ* since the best compressors give, in Vitanyi's words, the best results. As this work serves only as introduction to NCD applicability, we have decided *LZ* as a proof of concept purpose.

Most researches that have been done and future directions are related with the so called *Multimedia Information Retrieval*. NCD has been demonstrated to be a useful concept for any kind of media retrieval as image or music. Retrieving multimedia by its content (notes, rhythm, shapes, colours, etc.) can be a key concept for future search engine development, since current one are still related to *keywords* or *context*. An evolution of these search engine capabilities has been recently started in our group of investigation. A wide range of experiments with *standard* datasets (see Section 2.5) is also a requirement.

The future of this field is so, wide open. Not many people have initiated an investigation over content-based search engine, much less for private purposes. *Proximic*[13] search engine recently started to apply these concepts, yet there is no information about its implementation. So we expect that these kinds of approaches are going to be important for next generation search engines, at least for domain-restricted purposes.

[13] http://www.proximic.com/

Appendix A. Datasets and Querys

Dataset: Computer-Science oriented

This dataset contains 100 publications of 17 different authors; most of them belong to the UAM-EPS Computer Science Department. Authors are:

David Domínguez (6 documents), Diana Pérez (3), Enrique Alfonseca (6), Estefanía Martín (1), Estrella Pulido (2), Germán Montoro (10), Gonzalo Martínez (4), Guillermo G. de Rivera (3), Javier Aracil (7), Julian Fiérrez (2), Manuel Alfonseca (6), Manuel Freire (6), Mariano Rico (8), Michael Rohs (19), Nicolás Morales (9), Pablo Castells (7), Ricardo Ribalda (1).

Queries used with this dataset

For experiment no.1: Abstract of one publication (stored in the DB) of Michael Rohs.

For experiment no.2: 6 abstracts of different publications (stored in the DB) of Michael Rohs

Dataset: Large non-topic oriented

This dataset contains 200 publications of 10 different categories (20 documents each one); publications have been extracted from the *ScienceDirect* website. Categories are:

- Neural Networks: Cortical circuits for perceptual inference.
- AI: Knowledge forgetting: Properties and applications.
- Broadband Cable Access Networks: Coaxial RF Technology.
- Annual Review in Automatic Programming: Identification of fuzzy rules from learning data.
- Diuretic Agents: The Effects of Diuretics on Calcium Metabolism: Physiologic and Clinical Effects.
- Control in Robotics and Automation: Complementary Sensor Fusion in Robotic Manipulation.
- Gene Expression Patterns: Comparative analysis of conditional reporter alleles in the developing embryo and embryonic nervous system.
- Binary Digital Image Processing: Binary digital image characteristics.
- Flora - Morphology, Distribution, Functional Ecology of Plants: An embryological study of Eriolaena candollei Wallich (Malvaceae) and its systematic implications.
- Religion: The ‘Emerging Church’ in America: Notes on the interaction of Christianities

Queries used with this dataset (only experiment no.3)

For training: 5 abstract of documents of “Religion: The ‘Emerging…”

For experimental: Abstract of each one of the rest 15 documents of “Religion: The ‘Emerging…”

For further information contact to r.martinez@uam.es

References

Akman, V., Surav, M. (1997). *The use of situation theory in context modeling*. Computational Intelligence 13, 3 (1997), 427–438.

Allan, J. (2003). *Challenges in information retrieval and language modeling*. SIGIR Forum, 37, 31–47.

Anick, P.G., & Vaithyanathan, S. (1997). *Exploiting clustering and phrases for context-based information retrieval.* Paper presented at the 20th annual international ACM SIGIR conference on research and development, Philadelphia, PA.

Arkadi Kagan. *Entropy Compression Methods*.
http://compressions.sourceforge.net/Entropy.html

Baeza-Yates, F. y Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley.

Barnett, V.,Lewis, T. (1994). *Outliers in statistical data*. Wiley New York.

Bell, T. C., Cleary, J. G., Witten, I. H. (1990). *Text compression*. Prentice Hall.

Brants, T. (2004). *Natural Language Processing in Information Retrieval.* In proceedings of CLIN 2004 Antwerp, Belgium, 1–13.

Brants, T. (2000). *TnT – A Statistical Part-of-Speech Tagger*, In Proceedings of the Sixth Conference on Applied Natural Language Processing ANLP-2000, Seattle, WA.

Brill, E. (1992). A *simple rule-based part of speech tagger*, In Proceedings of ANLP-92, pages 152–155, Trento, Italy.

Brin, S., Page L. (1998). *The anatomy of a large-scale hypertextual Web search engine*. Computer Networks and ISDN Systems, Vol. 30, No. 1--7. (1998), pp. 107-117

Brisaboa, N.R., Fariña, A.,Navarro, G., Paramá J.R. (2007). *Lightweight natural language text compression*. In Journal on Information Retrieval vol.10, no.1

Burrows, M., & Wheeler, D. J. (1994). *A block-sorting lossless data compression algorithm,* Technical Report 124, Digital Equipment Corporation.

C.E. Shannon. (1948). *A Mathematical Theory of Communication*. Bell System Technical Journal, vol. 27, pp. 379-423, 623-656.

Chen, S., B. Ma and K. Zhang. (2009). *On the Similarity Metric and the Distance Metric*. Theoretical Computer Science 410(24-25): 2365-2376.

Cheverst, K., Mitchell, K., Davies, N. (1999). *Design of an object model for a context sensitive tourist GUIDE*. Computers and Graphics 23, 6 (1999), 883–891.

Crestani, F., Ruthven, I. (2007). *Introduction to special issue on contextual information retrieval systems*. Journal on Information Retrieval (vol. 10, pp. 111-113). Special issue on contextual information retrieval systems.

Daelemans, W., Zavrel, J., Berck, J. and Gillis S. (1996). *MBT: A Memory-Based Part of Speech Tagger-Generator*, In Proceedings of the Workshop on Very Large Corpora, Copenhagen, Denmark.

Davies, L., Gather U. (1993). *The identification of multiple outliers*. Journal of the American Statistical Association, vol. 88, no. 423, pp.782–792.

Dey, A. K. (2001). *Understanding and using context. Personal and Ubiquitous Computing*, Special issue on Situated Interaction and Ubiquitous Computing 5, 1.

Douglass R. Cutting , David R. Karger , Jan O. Pedersen , John W. Tukey (1992). *Scatter-Gather: a cluster-based approach to browsing large document collections*. Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval, p.318-329, June 21-24. Copenhagen, Denmark.

Efthimis N. Efthimiadis. (1996). *Query Expansion*. In: Martha E. Williams (ed.), Annual Review of Information Systems and Technology (ARIST), v31, pp 121–187.

Freire, M., Cebrian, M., del Rosal, E. (2007). *Ac: An integrated source code plagiarism detection environment.* [Online]. Available: arXiv:cs/0703136.

**Ghidini, C., Giunchiglia, F. (2001*)*. *Local models semantics, or contextual reasoning = locality+compatibility*. Artificial Intelligence 127, 2 (2001), 221–259.

Guillermo E. Pedroni (1996). *The Importance of The World Wide Web in Education K-12*. MSE of Southern Illinois University at Edwardsville .

Guy E. Blelloch. (2000). *Introduction to data compression*. Course notes for: Algorithms for the real world.

Halpin, T. A. (2001). *Information Modeling and Relational Databases: From Conceptual Analysis to Logical Design*. Morgan Kaufman Publishers, San Francisco.

Heaps, H. S. (1978). *Information retrieval: computational and theoretical aspects*. NewYork: Academic Press.

Held, A., Buchholz, S., Schill, A. (2002). *Modeling of context information for pervasive computing applications*. In Proceedings of SCI 2002.

Henricksen, K., Indulska, J., Rakotonirainy, A. (2003). *Generating Context Management Infrastructure from High-Level Context Models*. In Industrial Track Proceedings of the 4th International Conference on Mobile Data Management (MDM2003) (Melbourne/Australia) pp. 1-6.

Henricksen, K., Indulska, J.,Rakotonirainy, (2002). *A Modeling context information in pervasive computing systems*. In LNCS 2414: Proceedings of 1st International Conference on Pervasive Computing (Zurich, Switzerland, 2002), F. Mattern and M. Naghshineh, Eds., Lecture Notes in Computer Science (LNCS), Springer, pp. 167–180.

Hernandez, N., Mothe, J., Chrisment, C., Egret, D. (2007). *Modeling context through domain ontologies*, Information Retrieval, v.10 n.2, p.143-172.

Huettner, A. (2000). *Question Answering*. In: 5th Search Engine Meeting.

Huffman, D.A. (1952). *A Method for the Construction of Minimum-Redundancy Codes.* Proceedings of the I.R.E., (pp 1098–1102)

J.M. Maciejowski, (1979). *Model Discrimination Using an Algorithmic Information Criterion*. Automatica, vol. 15, pp. 579-593.

Jones, G. J. F. (2004). *The role of context in information retrieval*. In Proceedings of the ACM SIGIR Workshop on Information Retrieval in Context (pp. 20–23).

Li, M., Chen, X., Li, X., Ma, B., Vitanyi, P.M.B. (2004). *The similarity metric*. IEEE Transactions on Information Theory 50(12), 3250–3264.

Liu, X., Croft, W.B. (2004). *Cluster-based retrieval using language models*. In Proc. SIGIR, pp. 186–193. ACM Press.

Manning, C.D., Raghavan P., Schütze H. (2009, draft). *An Introduction to Information Retrieval*. Cambridge University Press. http://www.informationretrieval.org/

Marchionini, G. (1992). *Interfaces for End-User Information Seeking*, in Joumal of the American Society for Information Science, 43(2):156-163.

Martínez, R., Cebrián, M., Rodríguez, F., Camacho D. (2008). *Contextual Information Retrieval based on Algorithmic Information Theory and Statistical Outlier Detection*. In IEEE Information Theory Workshop (ITW 2008), May 5-9 in Porto, Portugal.

McKeown, K. R., Barzilay R., Evans D., Hatzivassiloglou V., Klavans J.L., Nenkova A., Sable C., Schiffman B., and Sigelman S. (2002). *Tracking and summarizing news on a daily basis with Columbia's Newsblaster*. In Proc. Human Language Technology Conference. 351, 373, 520, 522, 524, 525, 527, 528, 530.

Ming Li and Paul M. B. Vitanyi. (1993). *An Introduction to Kolmogorov Complexity and Its Applications*, Springer-Verlag, Berlin.

Moura, E., Navarro, G., Ziviani, N.,& Baeza-Yates, R. (2000). *Fast and flexibleword searching on compressed text.*, ACM Transactions on Information Systems, 18(2), 113–139.

Paul M. B. Vitanyi, Frank J. Balbach, Rudi L. Cilibrasi, and Ming Li. (2008). *Normalized information distance*, in Information Theory and Statistical Learning, F. Emmert-Streib and M. Dehmer, Eds., pp. 45–82. Springer-Verlag, New-York.

Pearson, R. (2002). *Outliers in process modeling and identification*. Control Systems Technology, vol. 27, pp. 10(1):55–63.

R. Cilibrasi, P. Vitanyi. (2005). *Clustering by compression*, IEEE Trans. Information Theory, 51:4(2005), 1523- 1545.

Ratnaparkhi, A. (1996). *A Maximum Entropy Model for Part-of-Speech Tagging*, In Proceedings of the Conference on Empirical Methods in Natural Language Processing EMNLP-96, Philadelphia, PA.

Salton, G. (1968). *Automatic information organization and retrieval*. McGraw-Hill, Ch. 4.

Salton, G. (1971). *Cluster search strategies and the optimization of retrieval effectiveness*. In The SMART Retrieval System, pp. 223–242. 351, 372, 530.

Salton, G. and Buckley, C. (1988). *Term-weighting approaches in automatic text retrieval*. Information Processing & Management

Salton, G. and M. J. McGill (1983). *Introduction to modern information retrieval*.

Salton, G., Edward A. Fox & Harry Wu (November 1983). *Extended Boolean information retrieval*. Communications of the ACM

Salton, G., Wong A., Yang C.S. (1975). *A vector space model for automatic indexing*, Communications of the ACM, v.18 n.11, p.613-620.

Schmidt, A., Beigl, M., Gellersen, H.-W. (1999). *There is more to context than location*. Computers andGraphics 23, 6 (1999), 893–901.

Spärck Jones, K. (1972). *A statistical interpretation of term specificity and its application in retrieval*. Journal of Documentation

Spärck Jones, K. (1999). *IR lessons for AI*. In Proceedings of Searching for Information, Artificial Intelligence and Information Retrieval Approaches (vol. 7, pp. 1–3). IEEE Special event.

Strang T. and Linnhoff-Popien C. (2004). *A context modeling survey*. In 1st Int. Workshop on Advanced Context Modelling, Reasoning and Management.

Strzalkowski, T., Lin, F., Wang, J. and Perez-Carballo, J.(1999). *Evaluating Natural Language Processing Techniques in Information Retrieval*, In Tomek Strzalkowski (ed.), Natural Language Information Retrieval. Kluwer Academic Publishers, Dordrecht.

Tan, P. N., Kumar, V., Srivastava, J. (2002). *Selecting the right interestingness measure for association patterns*. In Proc. SIGKDD'02, Edmonton, Alberta, Canada, pp.32-44.

Uschold, M., Grüninger, M. (1996). *Ontologies: Principles, methods, and applications*. KnowledgeEngineering Review 11, 2 (1996), 93–155.

van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Butterworths.

Wikipedia (2009). *Huffman coding*. http://en.wikipedia.org/wiki/Huffman_coding

Wilcox, R. (2003). *Applying contemporary statistical techniques*.

Ziv, J., & Lempel, A. (1977). *A universal algorithm for sequential data compression*. IEEE Transactions on Information Theory, 23(3), 337–343.

Ziv, J., & Lempel, A. (1978). *Compression of individual sequences via variable-rate coding*. IEEE Transactions on Information Theory, 24(5), 530–536.