The HarSearch Similarity Map:

Visualising Search Result Sets Using a Maximum Similarity Spanning Tree

Alexander Rodiga

The HarSearch Similarity Map:

Visualising Search Result Sets Using a Maximum Similarity Spanning Tree

Master's Thesis

at

Graz University of Technology

submitted by

Alexander Rodiga

Institute for Information Processing and Computer Supported New Media (IICM),
Graz University of Technology
A-8010 Graz, Austria

May 1997

Advisor: o.Univ.-Prof. Dr. Dr.h.c. Hermann Maurer
Supervisor: Univ.Ass. Dr. Keith Andrews

Die HarSearch Similarity Map:

Visualisierung von Suchergebnissen mittels eines maximalen Ähnlichkeitsspannbaumes

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Alexander Rodiga

Institut für Informationsverarbeitung und Computergestützte neue Medien
(IICM),
Technische Universität Graz
A-8010 Graz

Mai 1997

Diese Diplomarbeit ist in englischer Sprache verfaßt.

Begutachter: o.Univ.-Prof. Dr. Dr.h.c. Hermann Maurer
Betreuer: Univ.Ass. Dr. Keith Andrews

Abstract

As the size of an information space grows very large, finding specific information becomes increasingly difficult and ever more matches are returned by search queries. Numerous approaches have been proposed to make sense of search result sets.

This thesis describes HarSearch, an extension to the Hyperwave client Harmony, which utilises the retrieval mechanisms of the Hyperwave server. HarSearch provides a Similarity Map which takes an innovative approach by visualising search result sets using a maximum similarity spanning tree. The Similarity Map allows users to interactively explore search result sets, in terms of document similarities.

Kurzfassung

Mit der ständig wachsenden Menge verfügbarer Information wird es zunehmend schwieriger, relevante Information zu finden. Suchabfragen führen oft zu einer unüberschaubaren Anzahl gefundener Objekte. Verschiedenste Methoden wurden bereits vorgeschlagen, um es den Anwendern zu erleichtern, die für sie interessanten Objekte in einem Suchergebnis zu finden.

Diese Diplomarbeit beschreibt HarSearch, ein flexibles Suchwerkzeug, daß die Suchfunktionalität des Hyperwave Clients Harmony erweitert. Die HarSearch Similarity Map zeigt einen innovativen Ansatz wie Suchergebnisse mittels eines maximalen Ähnlichkeitsspannbaumes visualisiert werden können. Mit Hilfe der Similarity Map ist es Anwendern möglich, interaktiv Suchergebnisse zu betrachten und Beziehungen zwischen den gefundenen Objekten herzustellen.

I hereby certify that the work presented in this thesis is my own and that work performed by others is appropriately cited.

Ich versichere hiermit, diese Arbeit selbständig verfaßt, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben.

Acknowledgments

I would like to thank my supervisor Keith Andrews for his help and advice in writing and correcting this work. Bernhard Marschall, Michael Pichler, Jürgen Schipflinger, and Jörg Faschingbauer for their help in mastering the difficulties of programming. All the people at the IICM and especially the Hyperwave team who gave me the possibility to work in an ambitious team on a great idea.

My most heartfelt thanks go to my family, my parents and grand-parents, for their support, love, and encouragement during the whole time of my studies. I affectionately dedicate this work to them.

Danksagungen

In erster Linie möchte ich meinem Betreuer Keith Andrews für den Rat und die Hilfe beim Schreiben und Korrigieren dieser Arbeit danken. Weiters danke ich Bernhard Marschall, Michael Pichler, Jürgen Schipflinger und Jörg Faschingbauer, die mir in allen programmiertechnischen Fragen hilfreich zur Seite gestanden sind, sowie allen Mitarbeitern des IICM, die es mir ermöglicht haben, in einem ambitionierten Team an einer großartigen Idee mitzuarbeiten.

Ganz besonderer Dank gilt meiner Familie, vor allem meinen Eltern und Großeltern, denen ich diese Arbeit widme, für ihre Unterstützung während meiner Studienzeit.

Contents

1 Introduction **1**

2 Information Retrieval (IR) **3**

2.1 Introduction to Information Retrieval 3

2.2 Indexing Documents 4

2.2.1 Inverted Indexing 5

2.3 Measuring Retrieval Effectiveness 7

2.4 Relevance, Ranking, and Similarity 7

2.4.1 Document Clustering 10

3 Search Interfaces **13**

3.1 What is a Search Interface ? 13

3.2 Examples of Search Interfaces 15

4 Visualising Search Result Sets **17**

4.1 Visualising Information Spaces 17

4.2 Scatter/Gather 18

4.3 Bead 20

4.4 VR-VIBE 24

4.5 LyberWorld 28

4.6 InfoCrystal 31

4.7 TileBars 34

4.8 Envision 36

5 The Web, Hyperwave, and Harmony **39**

5.1 The Internet 39

5.1.1 Connections via the Internet 39

5.1.2 The World Wide Web 40

5.2 Hyperwave 42

5.2.1 Problems of First Generation Information Systems on the WWW 42

5.2.2 The Hyperwave Server 43
5.3 The Harmony Client 45
5.3.1 Searching in Harmony 46

6 The HarSearch Interface 51
6.1 The Features of the HarSearch Interface 51
6.2 The Interface of the HarSearch Similarity Map 53

7 The Design of the HarSearch Similarity Map 55
7.1 The Idea Behind the Similarity Map 55
7.2 The Maximum Similarity Spanning Tree 56
7.2.1 Prim's Algorithm to Grow a Maximum Spanning Tree 57
7.2.2 Clustering Using a Maximum Similarity Spanning Tree 59
7.3 Visualising the HarSearch Similarity Map 60
7.3.1 The vertical Tree Layout 60
7.3.2 The two-sided Tree Layout 61
7.3.3 The Polar Coordinates Layout 61
7.3.4 Determining the Horizontal Positions of the Vertices 63

8 The Architecture of HarSearch 65
8.1 The Connection to Harmony and Hyperwave 66
8.2 The API Search Tool 66
8.3 The Interface Objects 68
8.4 Graph and Node Objects 68
8.5 The Similarity Manager 69

9 Selected Details of the Implementation 71
9.1 class APISearchTool 71
9.2 The Interface Classes 72
9.3 Classes for the Tree Visualisation 72
9.3.1 class APSNode 72
9.3.2 class visAPSNode 73
9.3.3 class APSGraph 74
9.3.4 class APSSimilarityGraph 74
9.4 class SimilarityManager 76

10 Outlook and Future Work 79

11 Concluding Remarks 81

A HarSearch User Guide **83**
A.1 The HarSearch Search Interfaces 83
A.1.1 Common Elements 83
A.1.2 The Simple Query Interface 85
A.1.3 The Extended Query Interface 86
A.1.4 The Power Query Interface 89
A.2 The Result Window 90
A.3 The HarSearch Similarity Map 91
A.4 XDefaults 97

B The Test File Format **101**

References **103**

List of Figures

2.1 Information Retrieval. 4
2.2 Recall-Precision graph. 8
2.3 Triangular inequality 9
2.4 Five disjunctive clusters of objects. 10

3.1 The advanced query interface of AltaVista. 15
3.2 The HotBot search interface. 16

4.1 A Scatter/Gather example. 19
4.2 A Scatter/Gather application 21
4.3 Bead: point cloud 22
4.4 Bead: landscape 23
4.5 Multiple users exploring a VR-VIBE document space 25
4.6 Labelled documents in VR-VIBE 26
4.7 VR-VIBE: 3D-Relevance Scroll Bar 26
4.8 VR-VIBE: Web Browser 27
4.9 A simple LyberTree 28
4.10 A complex LyberTree structure 29
4.11 LyberWorld: Relevance Sphere 30
4.12 A Venn Diagram transformed into an InfoCrystal. 32
4.13 An InfoCrystal with four criteria. 33
4.14 TileBars: term distribution 34
4.15 Search result list with TileBars 35
4.16 Envision's Query Window 36
4.17 Envision: Search Result Window 38

5.1 Linear text and hypertext. 41
5.2 Hyperwave collections seen through Netscape. 44
5.3 The Harmony session manager. 46
5.4 The Harmony Landscape. 47
5.5 The Harmony Local Map. 47

5.6 The search interface of Harmony. 48

7.1 Examples for maximum spanning trees. 56
7.2 Prim's algorithm 57
7.3 Document clusters displayed in the HarSearch Similarity Map. 59
7.4 The vertical tree layout. 60
7.5 The two-sided vertical and horizontal tree layout. 61
7.6 The principle of the polar coordinate layout. 62
7.7 The HarSearch Similarity Map using polar coordinates layout. 62
7.8 The Priority Layout Method 64

8.1 The architecture of HarSearch. 65

A.1 The simple query interface. 86
A.2 The extended query interface. 87
A.3 The attribute menu of the extended query interface 88
A.4 The power query interface. 89
A.5 The result window. 90
A.6 A HarSearch Similarity Map example 91
A.7 A list of all similarities to a previously selected document. 92
A.8 The options dialog for the HarSearch Similarity Map. 94
A.9 Similarity Map colour specification. 95
A.10 Different Similarity Map layout styles. 96

1

Introduction

Nowadays a vast amount of information is available to us via textual and multimedia databases. Almost anything can be found even on-line on the Internet, but as the quantity of data grows, and the information space becomes overwhelming, so the retrieval of demanded information becomes more difficult and complex.

Various systems have been developed to visualise information spaces, to help the user to retrieve the desired information efficiently, effectively and easily, and to overcome the problem of being "lost in Hyperspace" by using the human's remarkable capabilities of perception and recognition.

Hyperwave represents the next generation of multimedia information systems. In cooperation with Harmony it is a powerful tool to provide, structure and grasp the scope of the available information. It offers solutions to common problems of existing information systems of the first generation.

This thesis describes an application, called HarSearch, which utilises Hyperwave's powerful information retrieval functionality. It offers an efficient user interface and makes sense of search result sets using a maximum similarity spanning tree to visualise document similarities.

Chapter 2 introduces the main principles of information retrieval. Mechanisms to index documents are described as well as document clustering, and measures to determine retrieval effectiveness and document similarity.

Chapter 3 focuses on search interfaces, the interfaces between users and search engines, and Chapter 4 discusses several systems which are directed to the visualisation of information spaces and search result sets.

Chapter 5 describes the basic concepts of the Internet followed by a listing of main problems of current first generation information systems. Solutions to these problems, offered by the Hyperwave server, are described afterwards. Finally, the Hyperwave client Harmony and its search interface is introduced.

The features of the new Hyperwave search tool HarSearch are the topic of Chapter 6. Furthermore, the interface of the innovative HarSearch Similarity Map is presented. The Similarity Map allows users to find document clusters within a search result set and facilitates the recognition of relations between similar documents.

However, delays in the implementation of document similarity in the Hyperwave server meant that only simulated similarity values could be used upto now in the development of the HarSearch Similarity Map.

Chapter 7 is directed to the underlying concept of the HarSearch Similarity Map. The idea and the theory which finally led to the visualisation of search result sets by using a maximum similarity spanning tree are described as well as the algorithms used for the actual layout of the Similarity Map.

HarSearch consists of several components. The overall architecture and the connection to Harmony and the Hyperwave server are described in Chapter 8 while Chapter 9 focuses on the actual implementation details. The most important C++ class declarations and source code parts are presented to offer an orientation aid to anyone who extends and maintains HarSearch.

Finally, Chapter 10 is an outlook on extensions of HarSearch which are not yet implemented. Primarily, real document similarities from the Hyperwave server should be integrated as soon as they are available, and usability evaluations can provide feedback for further improvements.

A detailed user guide is provided in Appendix A. It should help users to start searching with HarSearch and to find the requested information of interest in the Hyperwave space.

2

Information Retrieval (IR)

In this chapter the term information retrieval is explained and techniques for storing, structuring, and retrieving information are described.

2.1 Introduction to Information Retrieval

Techniques to store knowledge and to search for information are very old and can be found in any library. The most common way to find a book is to look up a few index terms such as the author, the title and some subject headings in an index.

Nowadays the traditional library index has often been replaced by a computer system. These systems have changed the way of storing and searching for information. The number of index terms is no longer limited to a few "hand-selected" terms, but may be generated automatically for all the terms within a document or even a document collection. An index which contains all or at least most of the words of a document collection is called a *full text index* (see Section 2.2).

The subfield of computer science that deals with the automated storage and retrieval of documents is called *information retrieval* (IR). The retrieval itself could be defined as the identification and retrieval of particular matching (textual) documents from an information base in response to an information request (query). In most cases information retrieval means text retrieval which is the retrieval of only text documents, though it might be possible that IR systems deal with pictures and other media as well.

The IR system matches the query terms with the terms in the documents and retrieves a set of relevant documents (see Figure 2.1). Retrieval is a probabilistic process which means that it is possible that some relevant documents are missed and some non-relevant are retrieved in a search task. The result depends on the retrieval technique and the combination of the index terms used to retrieve the documents (see Section 2.3).

The documents of an IR system are stored in a database. Nevertheless, a database management system (DBMS) should not be mixed up with an IR system. Although DBMS also retrieve documents, the retrieval of a DBMS is deterministic.

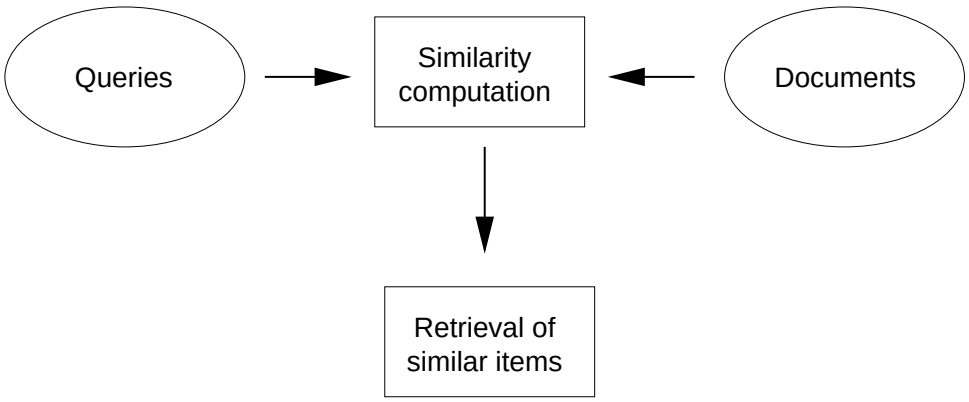


Figure 2.1: Information Retrieval.

The IR systems deal with largely unstructured text documents while the DBMS systems use homogeneous records to retrieve the stored information.

Therefore, there is a distinction between two types of retrieval. The *full text retrieval* of an IR system and the *Boolean retrieval* of a DBMS. In full text retrieval the query consists of terms which are searched in the whole document (or the full text index). The result is a list of ranked documents according to how close these documents are to combinations of the query terms. If the query terms occur more often in the document it is presumed that the document is more similar to the query (see Section 2.4).

In Boolean retrieval a Boolean query is made up of conjunctions, disjunctions, and negations of query terms. Documents are retrieved only if their attributes (e.g. title, keywords...) or contents satisfy the conditions of the Boolean statement. Boolean retrieval produces no ranking among the results because a document either satisfies the Boolean conditions or does not.

However, since an IR system uses a database to store documents and attribute fields like the author, and the title of the documents it may also support Boolean retrieval. Therefore, it is related to a DBMS and can be combined with a DBMS. One combination would be to use the ranking mechanism of full text retrieval to rank a set of documents retrieved by Boolean retrieval.

An IR system must support certain basic operations. There must be a way to enter documents and delete documents from the database, to change documents and to search for them. Finally, there must be a way to interact with the user and to present the search result sets, but this will be discussed in later chapters.

2.2 Indexing Documents

There are two possible approaches to retrieve documents by means of terms contained in these documents:

- Sequential scanning of the documents: this means scanning through every document to find the particular query terms. It is obvious that this technique is very time consuming and would not be feasible for large document collections. However, it can be used for string searching within a single document.

- Indexed documents: an index is available, and can be used to speed up the search. The index size is usually proportional to the document collection size and the search time is sub-linear on the size of the collection since a binary search strategy could be used.

Therefore, the *index* is the most important tool for IR. An index is a collection of terms with pointers to places where information can be found. The terms can be document titles, authors, or words within the documents.

Most people are familiar with the index of a book. It allows to find relevant information without scanning through the actual pages. The more pages the book has the more important is the index. The same can be done for document collections which may contain millions of documents to search for terms in the documents and to locate relevant documents.

2.2.1 Inverted Indexing

An index could be fashioned in many ways. The index as it is found in a book would be very slow and inconvenient for document collections to search in. To find a document, which contains a particular term, the system would have to go through the index of each document to check if that term is contained in this document. Therefore an inverted index is used.

An inverted index contains, for every term in the document database, an inverted index entry that stores a list of pointers to all occurrences of that term in the various documents in which that term appears.

An inverted index would look like as follows:

Term	Documents
1	1,3
2	2,3,4
3	2,4
4	4

It is easy to see that *Term 2* occurs in the documents 2, 3, and 4. The entries are stored in order of increasing document number so that different merging operations can be performed in linear time to the length of the lists. For example an AND-operation which is used to find documents containing the terms 2 AND 4 has as result the documents 2 and 3 while an OR-operation (terms 2 OR 4) retrieves the documents 2, 3, and 4.

The *granularity* of an index is the accuracy to which it identifies the location of a term. A coarse grained index may only identify the document which contains the term while a fine grained index might return even the exact position of the term within a sentence. However, adding precise location information increases the index because more information has to be stored for a single entry.

To improve the inverted index method and to support ranked search results the terms may be weighted for each document. A term-importance weight is assigned to each term. The weight depends on the frequency of the term in the document and the location of the term. This means that if the term occurs in a heading it is weighted more than if it occurs within a simple paragraph. The advantage is that the retrieved documents can be ranked in order of their relevance to the query but query matching becomes more complicated.

An inverted index with additional information can consume considerable space, and might occupy 50-100% of the space of the stored text itself. Therefore various compression methods are used to reduce the size of the index.

Beside compression other considerations should be taken into account when building an index:

Case Folding

The problem that queries fail because of a case insensitiveness of the index can be avoided by using *case-folding*. That is the replacing of all uppercase characters in a term with their equivalent lowercase characters. For example, "It", and "it" are both folded to "it" and indexed accordingly. The query terms are also case-folded before they are matched with the index terms. A case sensitive search could lead to false matches, unless some post-processing is performed.

Stop Words

It is questionable if words such as *it* should be indexed at all. Since words like *it*, *the*, *to*, and some others occur very frequently in almost every English document. Words that are "stopped" and not indexed are called *stop words* and the set of stop words is referred to as the *stop list*. In an uncompressed inverted index a stop list can save a substantial amount of space which may range up to 40% of the index size.

When the inverted index is compressed the benefit from stop words is less clear. The stop words are exactly the words which are compressed at the highest rate due to their high frequency and therefore the amount of space which could be saved becomes rather small in comparison to the remaining index size.

Furthermore, omitting words can cause problems if users are interested in finding exactly these words. Another problem with stop words is that some of the most common words may have different meanings. For example the removing of the word *may* would also remove any reference to the month of May. Therefore, the generation of an appropriate stop list is not as easy as it looks at first glance.

Stemming

Another process to reduce index size is *stemming*. Stemming involves stripping one ore more suffixes off a word to reduce it to the root form, converting it to a neutral term that is devoid of tense and plurality. For example the words *stemming* and *stemmed* are simply stored as *stem*. Since a single stem typically corresponds to several full terms, using stemming can achieve a compression factor of over 50%. However, reducing words to a root form and omitting suffixes can cause ambiguous search results and the generation of stems requires a detailed knowledge about the language of the documents which have to be indexed.

2.3 Measuring Retrieval Effectiveness

Since the retrieval of an IR system is a probabilistic process, it retrieves relevant and non-relevant documents and a measure is needed to estimate retrieval effectiveness. Many different measures have been proposed. The most common measures are *precision* and *recall*.

Precision P_r for a certain cutoff point r is the ratio of the number of relevant documents retrieved over the total number of documents retrieved:

$$P_r = \frac{\text{number retrieved that are relevant}}{\text{total number retrieved}}$$

Recall R_r of a method at some value r is the fraction of relevant documents retrieved for a given query over the number of relevant documents for that query in the database:

$$R_r = \frac{\text{number relevant that are retrieved}}{\text{total number relevant}}$$

Except for small test collections, this denominator is generally unknown and has to be estimated by sampling or some other method. Both precision and recall take a value between 0 and 1. The relation between precision and recall can be visualised using a recall-precision graph (see Figure 2.2). It shows that recall and precision are inversely related. That is, when recall goes up, precision typically goes down and vice versa.

A combined measure of recall and precision, E, has been developed by van Rijsbergen (1979). The evaluation measure E is defined as:

$$E = 1 - \frac{(1 * b^2)PR}{b^2P + R}$$

where P = precision, R = recall, and b is a measure of the relative importance, to a user, of recall and precision [FBY92].

2.4 Relevance, Ranking, and Similarity

Since IR systems retrieve sets of documents which are relevant to queries a measure for this relevance is required. The relevance could be considered as the similarity or

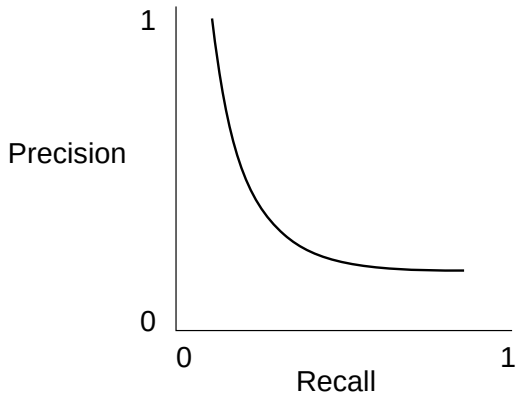


Figure 2.2: Recall-Precision graph.

the distance of a particular document to the user's information request (the query). If the document fulfils the information request it is close to what the user is searching for.

In everyday life we often refer to objects only as similar and dissimilar, which are highly subjective terms. A stricter definition of similarity is needed for automated calculations. The similarity between two objects o_i and o_j can be defined as a function $s = s(o_i, o_j) = s_{ij}$ which reflects our imagination of similarity, that is a greater value represents greater similarity. The function s is defined on the interval

$$s_0 \leq s \leq s_1$$

where $s_{ij} = s_1$ means most similar and $s_{ij} = s_0$ means least similar (usually $s_0 = 0$ and $s_1 = 1$). Function s satisfies the following conditions:

$$s_{ij} \leq s_1$$

$$s_{ij} = s_{ji}$$

$$s_{ii} = s_1$$

If also the conditions

$$s_{ij} = s_1 \Rightarrow o_i = o_j$$

$$|s_{ij} + s_{jk}|s_{ik} \leq s_{ij}s_{jk}$$

are satisfied then the function s is called a metrical similarity function. Equally, the dissimilarity or distance d can be defined as

$$d = d(o_i, o_j) = d_{ij}$$

with

$$d_{ij} \geq 0$$

$$d_{ij} = d_{ji}$$

$$d_{ii} = 0$$

and with the additional conditions

$$d_{ij} = 0 \Rightarrow o_i = o_j$$

$$d_{ik} \leq d_{ij} + d_{jk} \text{ (trinagular inequality, see Fig. 2.3)}$$

it is called a metrical distance function.

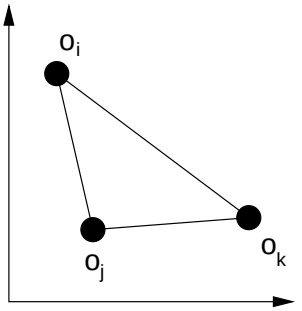


Figure 2.3: Triangular inequality: $d(o_i, o_k) \leq d(o_i, o_j) + d(o_j, o_k)$.

Now a similarity measure has to be defined for documents and queries. Beside probabilistic models and other attempts, the *vector space model* [FBY92, WMB94] is often used to determine the similarity of a document to a query or another document. In the vector space model each document D_i is described by a vector of n terms $\{a_{i1}, a_{i2}, ..., a_{in}\}$. This vector is high dimensional because its length is equal to the number of unique words contained in all the documents of the whole document space. Each component of the vector reflects the occurrence of the corresponding word in the document. Using a binary scale the value is 1 if the particular word occurs in the document and 0 otherwise. The value of each a_{ik} may also be the weighted frequency of occurrence of each term k in the document D_i (see Section 2.2.1). The query is also represented as an n–dimensional vector Q. The similarity between a query and any particular document can now be calculated by vector algebra. A simple measure for similarity is, for example, the reciprocal Euclidean distance:

$$d(Q, D_i) = \frac{1}{\sqrt{\sum_{j=1}^{n}(a_{q,j} - a_{d,j})^2}}$$

Since the number of words contained in a document is usually much greater than the number of query terms this measure discriminates against long documents. Therefore the cosine of the angle between the two vectors (i.e. the difference in direction) is taken as a similarity measure. The cosine of the angle θ is

$$\cos\theta = \frac{X \cdot Y}{|X||Y|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{(\sum_{i=1}^{n} x_i^2)}\sqrt{(\sum_{i=1}^{n} y_i^2)}}$$

where $X \cdot Y$ is the vector inner product. $|X| = \sqrt{(\sum_{i=1}^{n} x_i^2)}$ is the Euclidean length of X and $|Y|$ the Euclidean length of Y respectively. Hence, the *cosine measure* for

similarity can be defined as:

$$cosine(Q, D_i) = \frac{Q \cdot D_i}{|Q||D_i|} = \frac{1}{W_q W_i} \sum_{j=1}^{n} a_{q,j} a_{d,j}$$

where $W_i = \sqrt{\sum_{j=1}^{n} a_{i,j}^2}$ is the weight of the document D_i and $W_q = \sqrt{\sum_{j=1}^{n} a_{q,j}^2}$ is the weight of the query. Instead of a query, another document may be taken to compute the similarity between two documents.

Knowing the similarity of documents to a query and to other documents, the ranking of these documents according to their relevance can easily be accomplished, and the relations between n different documents in an $n \times n$ similarity matrix can be determined.

However, reducing distances in a multi-dimensional space to single representative values always involves a loss of information. (see Section 4.7 and [WMB94])

2.4.1 Document Clustering

Cluster analysis or classification is well-known in statistical mathematics and is used in many scientific fields, such as medicine, biology, or economics, to find groups of similar objects out of an unordered set of objects with different properties. Figure 2.4 shows 5 disjunctive clusters, though it might be possible that there is an overlap of clusters if one object belongs to several groups, depending on the problem specification.

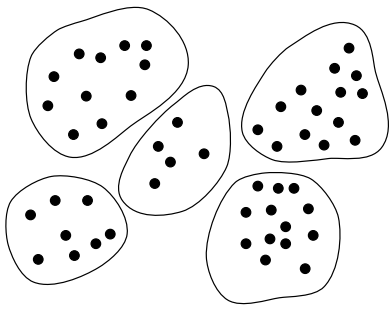


Figure 2.4: Five disjunctive clusters of objects.

Computers made it possible to use these classification methods also for many fields in computer science. Automated classification is used, for example, in pattern recognition and can also help to structure the information stored and retrieved by IR systems.

A document clustering algorithm tries to find clusters of similar documents within the document space using a similarity measure like the cosine measure described in Section 2.4. In [Bot93] three characteristics of clustering algorithms are given. First, the algorithm has to be stable, i.e., it should not be affected by changes in details. Second, a "reasonable" number of clusters should be generated. Finally, documents in a cluster should be highly related.

In general there are two approaches to clustering. The *agglomerative* approach at first considers every document as a single cluster. Then the most similar clusters are merged to bigger clusters in a greedy manner until a specified number of clusters remains. A *partitional* strategy starts with one cluster that contains all documents and splits this cluster, and the resulting clusters into smaller clusters until again a specified number of clusters is generated.

Clusters may be organised hierarchically. In this case a cluster is defined recursively as either an individual document or as a cluster of clusters which are also clustered hierarchically.

Many clustering methods are based on a pairwise coupling of the most similar documents or clusters. If there are n documents the running time of global clustering algorithms is $O(n^2)$, because every document pair has to be considered. Therefore, clustering of large document collections can be very time intensive. For that reason approximation algorithms have been developed in the recent years.

In [CKPT92] two algorithms are presented which achieve a considerable speedup. Both algorithms are based on the hierarchical approach and have rectangular running time, i.e. $O(kn)$ where k is the number of clusters. The algorithms are called *Buckshot* and *Fractionation* (see Section 4.2). They are used to find the initial centres for the seed-based clustering of the document space. In seed-based clustering a number of *seeds* is chosen from all documents according to the number of desired clusters. Then each document is assigned to the closest seed document. After some refinement this strategy results in an approximate partition of the document collection.

3

Search Interfaces

This chapter is directed to the interface between an information system and the user who wants to find information of interest. It is related to Chapter 4 which pays more attention to the presentation of search result sets while in this chapter the interaction with the user, which has to take place to search for documents, is in the foreground.

3.1 What is a Search Interface ?

As mentioned in Section 2.1, one of the basic operations of an IR system is the interaction with the user searching for documents. Therefore the IR system has to provide a search interface of some kind.

The search interface of an information system, or a search engine on the Internet (see Section 5.1) is important to find information of interest. There are different ways to search for information. One is following hyperlinks (see Section 5.1.2) between particular documents or navigating through a hierarchical document structure with collections and groups of documents. Another one is to issue a specific query, filtering the document space by means of query terms.

Browsing from link to link and the navigation through a document hierarchy are both query free. Navigating through the document hierarchy can be compared to the perusal of the table of contents while browsing from link to link is more like following cross references within a text. Both strategies may be used to get an overview of the available information. Finally, querying a system is similar to a look up in the index of a book to find specific information.

However, the standard mechanism to retrieve information from an IR system presumes a query specified by the user. The query reflects the user's information need by means of different query terms. The next step is to search in the document space for documents which match this query. There may be situations where it is not possible to formulate a query precisely. This often results in far too many found documents or none at all. Users may not be able to describe the topic of interest because they may not be looking for anything special, but rather want to get a general overview of the available information. It is also possible that the user is not

familiar with the appropriate vocabulary to describe the topic or is interested in several topics at once. Furthermore, the words used to describe a topic may not be the words used in articles of interest, and therefore, the query may fail to retrieve these articles. Even if some words are used in discussions a query may fail due to the use of synonyms.

On the other hand displaying the whole information space becomes problematic as its size increases. In this case a query may be more suitable, at least to filter the amount of data according to query terms. It also may not be possible to display the whole information space at all.

Typically, the usual information retrieval task includes searching as well as browsing. The first activity of the user is probably to explore the information space to get an overview of the available topics. The next task is a search task where the user tries to find articles and documents of interest. If appropriate documents have been found, a further browsing task may be applied to find related information.

Queries and query languages are often very complex, and therefore, the task of the search interface is to help the user to enter queries in a more understandable way, without bothering the user with query syntax and the internal organisation of the system. The user may enter simple text, select check boxes, or can choose from items in a list and the search interface composes the actual query.

Since search interfaces are often command or form based, it is difficult for the user to judge the relevance of the different query terms and it is not easy to change the query appropriately if the result is not satisfying.Some systems try to overcome this problem by offering ways to generate queries graphically and with more flexibility (see Section 4.5 and Section 4.6).

The search interface also presents the search result set and supports further search activities. The presentation of the search result set may be a simple list, but can also be more sophisticated to convey more information about the retrieved documents. A more detailed description of the visualisation of search result sets is given in Chapter 4.

One point that is important to the design of every user interface is the user profile. In general there are three types of users:

- Novice users
- Knowledgeable intermittent users
- Frequent users

Novice users need a much simpler interface than frequent users who know how to use the particular interface. In the case of a search interface, it is reasonable to assume that the users know about the information they are searching for but nothing about the system itself. A four-phase frame work for the design of search interfaces is introduced in [SBC97].

3.2 Examples of Search Interfaces

All search interfaces of the big search engines on the Internet are form-based. Figure 3.1 shows the advanced query interface of the AltaVista search engine [ALV]. The user can choose between a simple query interface and an advanced query interface. The result is presented as a ranked list. Since the underlying IR system contains millions of World Wide Web pages (see Section 5.1.2), a simple query often results in very many matches. For that reason, the advanced query interface is used for more accurate query specification.

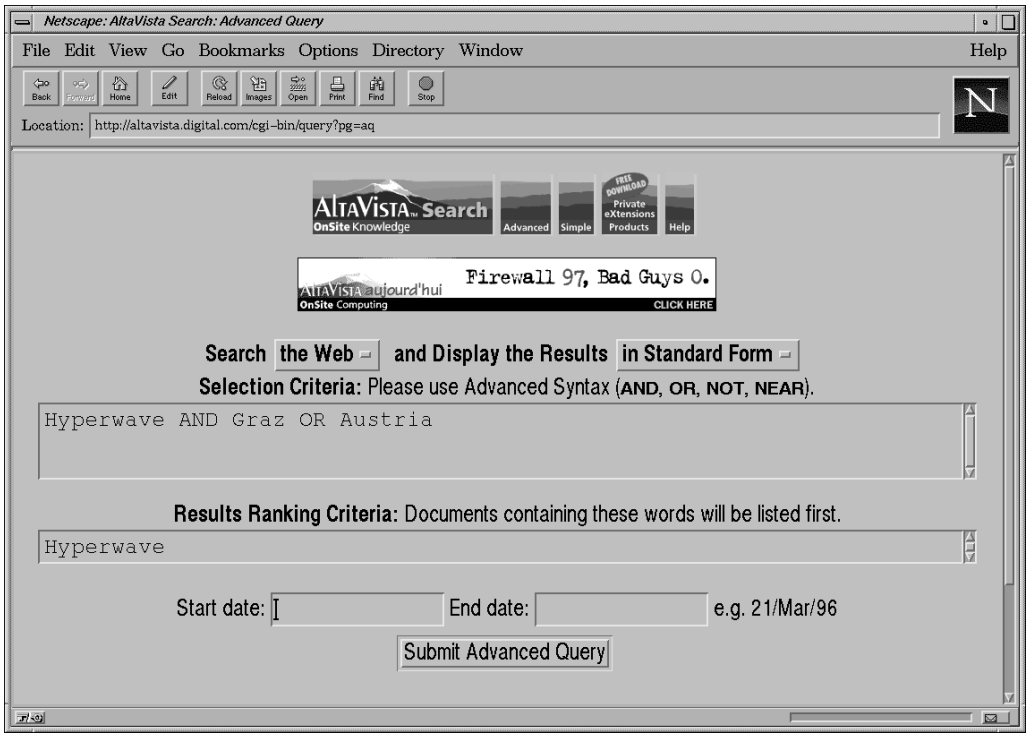


Figure 3.1: The advanced query interface of AltaVista.

Another search interface of an Internet search engine is presented in Figure 3.2. The HotBot [HBT] also indexes millions of WWW documents. Instead of two separate interfaces the user may open and close sections of the search interface dynamically. In the *MODIFY* section the user can specify single words which must or should be contained in the retrieved documents. The *DATE*, *LOCATION*, and *MEDIA TYPE* sections are further possibilities to extend the query interface.

The third search engine described is InfoSeek [IFS]. Its search interface is rather simple. At the main page the user has the possibility to enter query terms in a text field or to choose one of the offered document groups to browse through the information base. Once a query has been issued the result is presented as a ranked list. The remarkable feature is the possibility to search again in a previously retrieved search result set, which allows users to separate relevant documents from non-relevant documents by refining the query in several steps. The user can go back

one step and does not need to reformulate the whole query if the search result was not satisfying.

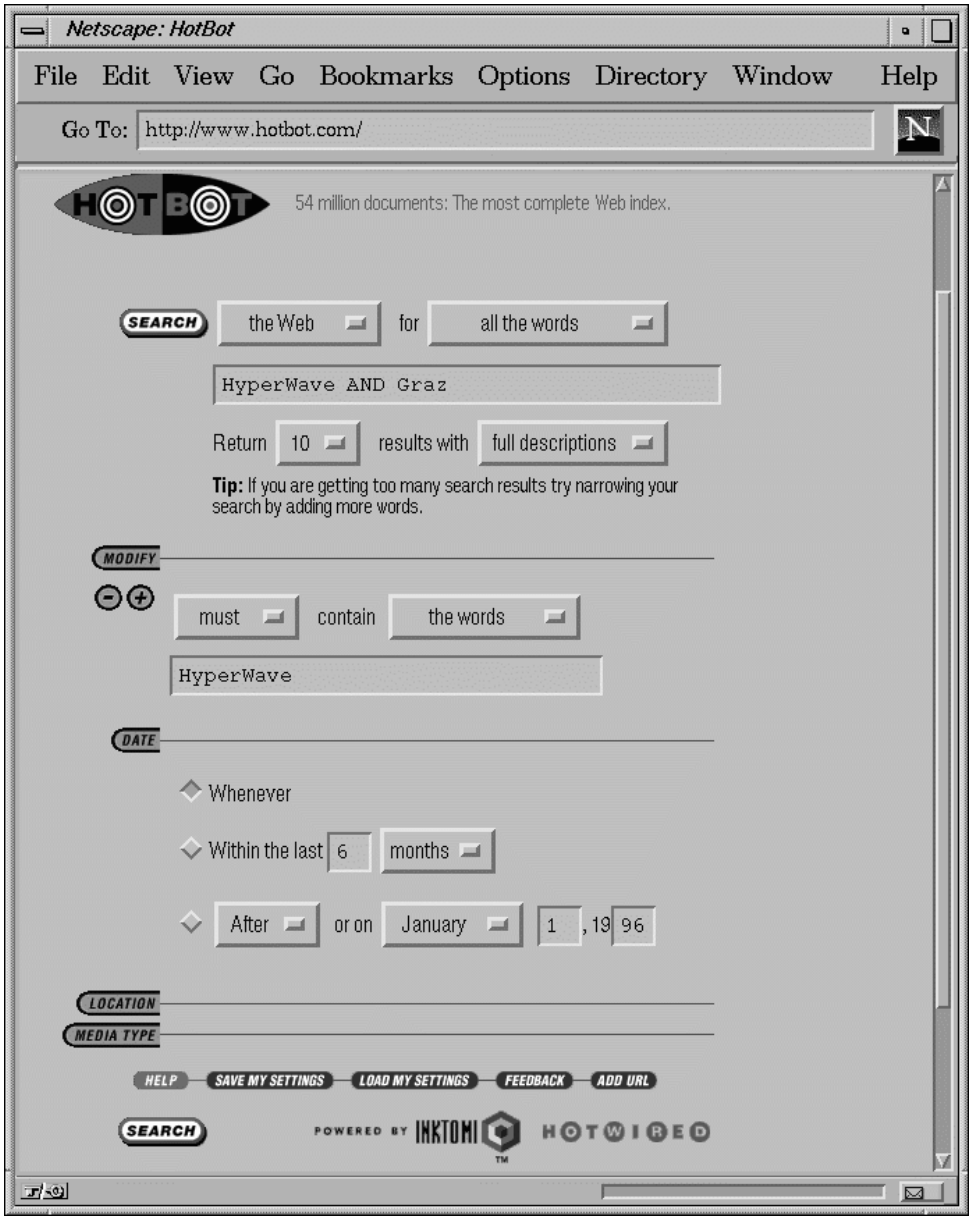


Figure 3.2: The HotBot search interface.

4

Visualising Search Result Sets

How several information search systems visualise a search result set is discussed in this chapter. Also the search interfaces of these systems are described, since in most cases the search interface is hardly separable from the actual search result visualisation.

4.1 Visualising Information Spaces

One goal of every information system is to present the information space to the user as understandably as possible.The user must be able to navigate through the abstract information space to access the presented documents.

In terms of the vector space model (see Section 2.4) every document can be considered as a point in this space. Since this vector space is high dimensional, it is not possible to display it directly and it is not easy to imagine either. It has to be mapped into a lower dimensional space which is more familiar to the user. In many cases this is our 3D world. The mapping procedure is called *multidimensional scaling* (MDS) [CC92].

Furthermore, the information space is often visualised using spatial metaphors. These metaphors are derived from objects of the real world and support the user building up a mental model of the information space. They can be simple geometric forms like spheres, cubes or more complex like trees, landscapes (see Section 4.3), or even cities.

The cognitive load on the user is decreased while interacting with more familiar objects or navigating through them. When visualising search result sets, spatial distance often corresponds to the similarity of documents to query terms or to other documents. Trees or acyclic graphs can be used to visualise document hierarchies.

Several other visual coding principles are used to convey information within visualisations to improve usability. The most popular ones are:

- Proximity Coding: as already mentioned, spatial distance corresponds to the similarity between documents or documents and query terms.

- Size Coding: is used to visualise quantitative information like relevance or the number of elements contained in an object. In three-dimensional visualisations size coding may sometimes be ambiguous due to perspective effects.
- Brightness and Saturation Coding: another way to represent quantitative information is to increase and decrease the brightness and saturation of elements.
- Texture and Colour Coding: also indicates quantitative information or it represents a particular attribute of the object, e.g. the type of the object.
- Shape Coding: different objects are displayed with different shapes making distinctions easier.

Of course the exact meaning of these principles depends on the actual use in the various systems. The list above provides only a general overview.

4.2 Scatter/Gather

Scatter/Gather is more an information access tool than a search system. It proposes a browsing paradigm for large document collections. Initially the whole document space is *scattered* into small clusters which are then presented to the user with short summaries. From these summaries the user selects topics of interest for further studies. The selected groups are gathered together to form a sub-collection. That sub-collection is then again scattered into a small number of document groups. With each step the groups become smaller and more detailed. Finally, when the groups become small enough the user reaches the document level and can access individual documents.

A Scatter/Gather Example

Figure 4.1 shows a typical Scatter/Gather session over a text collection consisting of articles posted to the *New York Times News Service* during August 1990. To simplify the figure only single-word labels are assigned to the cluster summaries.

It is supposed that the user wants to get a general overview of the news of this month. The big stories are obvious from the initial scattering: Iraq invades Kuwait and Germany considers reunification. This leads the user to focus on international issues: he selects 'Iraq', 'Germany' and 'Oil'. These three clusters are gathered together to a cluster of 'International Stories'. The smaller cluster is then scattered into eight new clusters containing a subset of the articles. The articles on the Iraqi invasion and some of the 'Oil' articles have now been separated into clusters discussing the U.S. military deployment, the effect of the invasion upon the oil market, and one which is about hostages in Kuwait.

The user feels his knowledge of these big events is adequate, but he wishes to find out what happened in other corners of the world. He selects the cluster named Africa and the 'Pakistan' cluster, which also contains other foreign political articles.

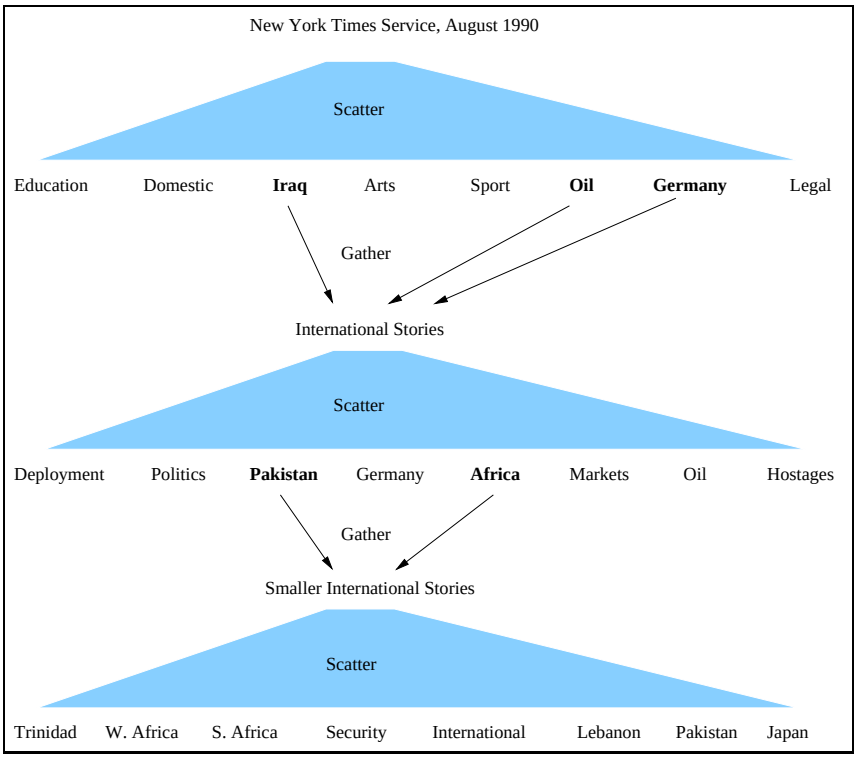


Figure 4.1: A Scatter/Gather example.

This reveals a number of specific international situations as well as a small collection of miscellaneous international articles. The user thus learns of a coup in Pakistan, and about hostages being taken in Trinidad, stories otherwise lost among the major stories of that month [CKPT92].

Requirements

Since the collections are clustered on the fly fast clustering algorithms are essential to achieve tolerable interaction times. For very large document collections this cannot be accomplished without preprocessing the data. Even this preprocessing step has to be efficient. To accelerate Scatter/Gather the developers suggest a fine grained clustering of the document collection and hypothesise that documents similar enough to be clustered in a fine-grained clustering will be clustered together in a coarse-grained clustering. This is called the *Cluster Refinement Hypothesis* [CKP93].

Clusters of this fine grained clustering are considered as *meta-documents*, i.e. nodes, in a *cluster hierarchy*. Each meta-document consists of a set of related documents and meta-documents. Every group of meta-documents represents the union of all groups of documents contained in these meta-documents. Therefore the cluster hierarchy can be recursively described as a tree in which a node either corresponds to a single document, or a tree whose subtrees are cluster hierarchies. Since the meta-documents form a *condensed representation* of the collection, a constant time bound can by maintained using the cluster hierarchy during Scatter/Gather Browsing.

In addition to fast clustering algorithms a method for automatically summarising document groups is required. Such a technique, called *cluster digest*, is introduced in [CKPT92]. The summary to describe a cluster is generated from the words which appear most frequently in all the documents contained in this particular cluster.

Scatter/Gather Clustering

The actual clustering of Scatter/Gather is performed by three different algorithms on the base of seed-based partitional clustering. Seed-based partitional clustering algorithms have three phases:

1. Find k centres.
2. Assign each document in the collection to a centre.
3. Refine the partition so constructed.

The result is a partition P of k disjoint document groups. The *Buckshot* and *Fractionation* algorithms are both used to find the initial centres. Both algorithms assume the existence of some algorithm which clusters well, but which may run slowly. *Group average agglomerative clustering* is used for this subroutine. Buckshot applies the cluster subroutine locally to a small random sample to find centres. Fractionation uses successive application of the cluster subroutine over fixed sized groups to find centres. Since Buckshot is significantly faster it is used for the clustering during an interactive Scatter/Gather session. Fractionation can be used to generate the initial partition of the whole document collection. In [CKPT92] the algorithms are described in detail.

Discussion

Scatter/Gather shows that document clustering can be used as an effective information access tool even for very large document collections. It is not intended to be used to find particular documents but it could be helpful in situations in which it is difficult to specify a query. The Scatter/Gather browsing paradigm may also be used to organise the results of word-based queries that retrieve too many documents (Figure 4.2 shows a screen dump of such an application). Fast clustering algorithms are essential to achieve a tolerable interaction time. Furthermore, preprocessing the data is necessary for very large document collections. The computation of the cluster hierarchy results in a linear storage overhead.

4.3 Bead

Bead is a prototype system for graphically-based exploration of information [CC92]. It is one of the early systems which visualise the whole information space using spatial proximity to display document similarity. Similar documents are placed close

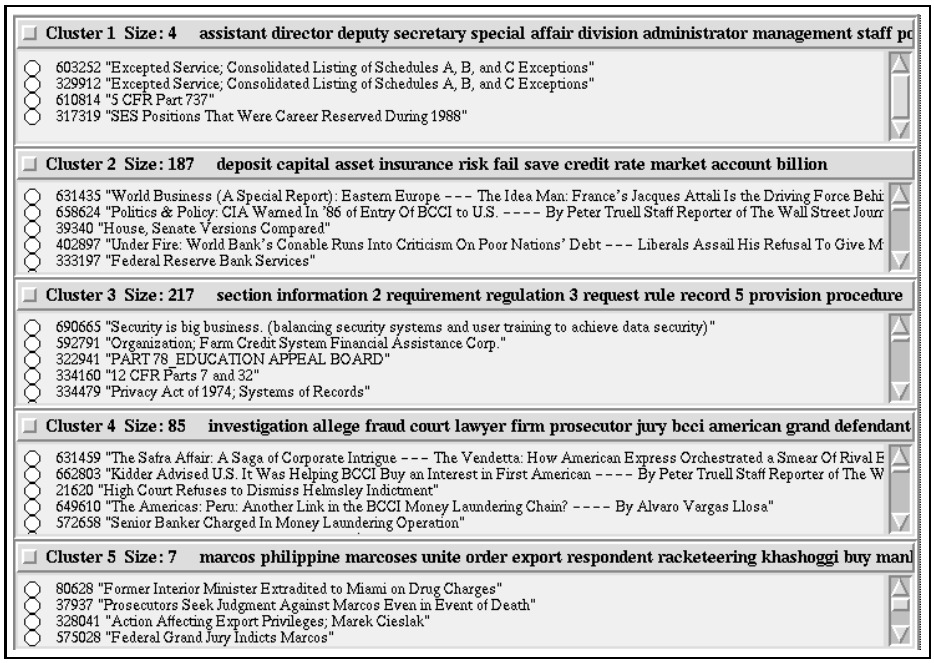


Figure 4.2: Clusters displayed by a Scatter/Gather application. Extracted from [SCB].

to one another and dissimilar ones further apart. The high-dimensional information space is mapped into 3D-space using a concept based on a physical model in which each document is associated with a particle in space. The aim is to retrieve information more graphically by navigating through the information space without knowledge of query languages and database material itself.

Underlying Model and Point Cloud Visualisation

The physical model which is taken for the documents' placement is the model of a damped spring in order to generate forces of attraction and repulsion between particles. When particles are too close, the spring pushes them apart. When they are too far apart, they are drawn towards each other. In each case the spring works to regain its 'rest distance' which is used as a metaphor for the document distance. For n documents this can lead to $n(n-1)$ interactions. That *n-body problem* is well-known in computational physics where interactions due to gravitational or Coulombic fields are often of exactly this $O(n^2)$ complexity. Approximate solutions of lower orders of complexity have therefore been developed. These methods employ the principle of superposition whereby a cluster of particles can be approximated by a single 'meta-particle'. These meta-particles are document clusters which are organised hierarchically very similar to the meta-documents in Scatter/Gather (see Section 4.2).

The particle or document placement is then calculated in order to minimise the unbalanced force on each particle. The outcome of this process is a three dimensional

point cloud constructed from all documents in the document space. Once the point cloud is calculated it can be visualised and explored using a number of different tools.

In the first version of Bead only two dimensional visualisations were possible. The *grid viewer* displayed three orthogonal plots (in XY, XZ and YZ) and a perspective view of the scene. The different documents were represented by there ID number. After a search, matching the query terms were highlighted. Figure 4.3 shows an example of such a point cloud. Since it is only a two dimensional visualisation the documents lie very close together, although they may be quite far apart in 3D-space.

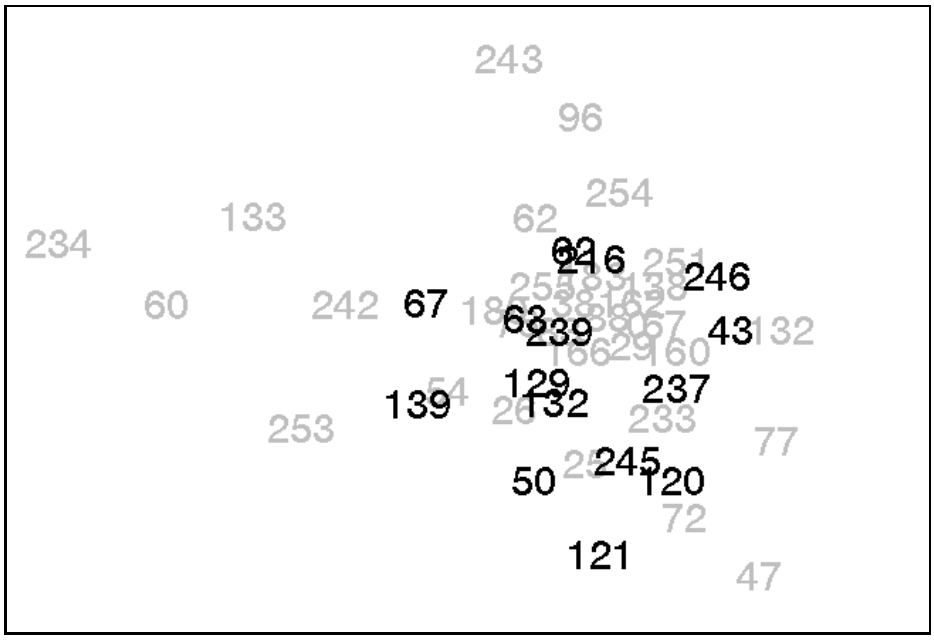


Figure 4.3: A point cloud displayed by Bead. Extracted from [Blu96].

Users could zoom in on a chosen document in order to see neighbouring, and therefore highly related, documents. To improve the 3D impression the user may adjust a radius to define a *sphere of interest* centred on the chosen document. The colours of documents outside that sphere were slightly reduced.

Enhanced Visualisation

Nevertheless, it was often the case that the complexity of the 3D patterns led to a cluttered display. The results of usability tests showed that users found it difficult to orient themselves and navigate within the space, and consequently did not build up a useful mental model of the corpus. Instead they found occasional items of interesting data, but had difficulties in assessing their relevance or significance in the wider context of the corpus [CC93].

To advance the design of Bead's information display the developers decided to move away from strongly 3D structures towards a landscape-like (2.1D) structure.

Of course, this meant a loss of exactitude of relative distances which can be gained in full 3D but promised greater accessibility and familiarity for the user. Using the landscape metaphor also implied a change of the modelling process.

Individual documents are displayed as coloured markers placed within the setting of the landscape and they consequently produce collective patterns of density and locality. A shore delimits the corpus and is usually made up of documents which are less strongly associated with any central topic of the corpus.

User can move freely over the landscape and select the individual documents with the mouse. They can zoom in or zoom out to get an overview of the entire set. After a search for various query terms the according documents are highlighted (see Figure 4.4).

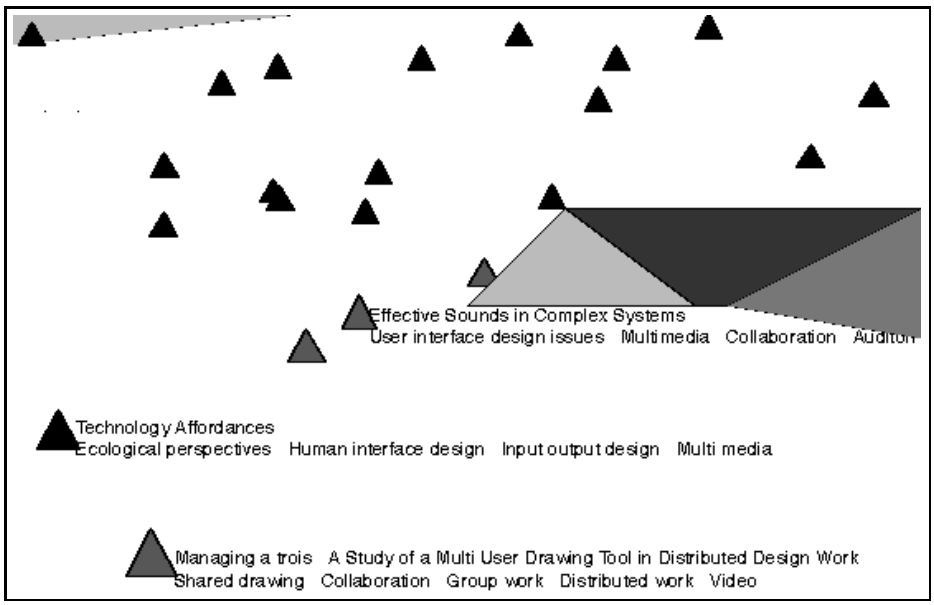


Figure 4.4: Zoom in a Bead landscape. Extracted from [Blu96].

The modelling process of Bead sometimes generates peaks and valleys which indicate areas where the system could not find a good 2D layout. A useful side effect of these areas is that they serve as landmarks which are important for orientation and navigation on the landscape.

Discussion

The design of Bead focuses on the approach to retrieve information in an exploratory way. It supports the user in building up a mental model of the information space. Furthermore, the model of a landscape provides overview and browsing, as well as the possibility to locate documents which are related to the keywords of a query. Therefore it offers an alternative to traditional search systems, although it might be difficult to visualise very large document collections.

4.4 VR-VIBE

VR-VIBE is a search system which visualises search result sets in relation to queries. It is a virtual reality extension of the VIBE System [OKS+93], and supports cooperative information retrieval. To implement the system a multi-user virtual reality environment, called DIVE, was used. The essence of VR-VIBE [BSG+95] is that multiple users can explore the results of applying several simultaneous queries to a corpus of documents and can search for information of interest together.

Document Visualisation

The corpus is visualised in 3D and the spatial position of each document depends on the relative attraction of this document to the different queries. The relative attraction derives from the relevance of the particular document to the query. The more relevant the document is to the query the higher is the attraction. Users may then arrange the queries within a spatial framework. A query with an associated position is called a *Point of Interest*(POI). The system places the document objects at their relative centre of attraction between these POIs. The relative centres of attraction are given by the sum of vector positions of the POIs weighted by their relative attractions to the documents. Thus, a document which is attracted by two POIs is placed on a line between these POIs according to the relative attraction of these POIs; a document which is relevant to 3 queries is placed in a triangle, and so on. Such a placement leads to a problem in a two dimensional visualisation because a document that is weakly attracted to each POI would be placed in the same position as one that is highly attracted to each POI, and which is therefore of greater relevance. Another problem is that an unambiguous placement in a three dimensional representation is only possible up to four different POIs (i.e. beyond four POIs it is ambiguous which POIs influence a particular document position).

Nevertheless, users can define as many POIs as they want. A query may consist of a single or several keywords. To overcome the problem of ambiguity these POIs may be moved around freely, to determine the effect of a single POI on the document corpus. Users also can dynamically switch POIs on and off or create new POIs. Changes in the POI arrangement cause a redraw of the document space.

Two visualisation styles are available:

- The POIs are placed in a two dimensional plane so that the position of each document shows the relative attraction to each POI. The third dimension is used to present the overall relevance of the documents. The higher a document is above the plane the more relevant it is.

- In the 3D-layout POIs are positioned freely in the three dimensional space and the documents are placed according to the relative attraction to these POIs. In this case the overall relevance is visualised using *shade-* and *size coding*. Documents with a higher relevance score are displayed larger and brighter.

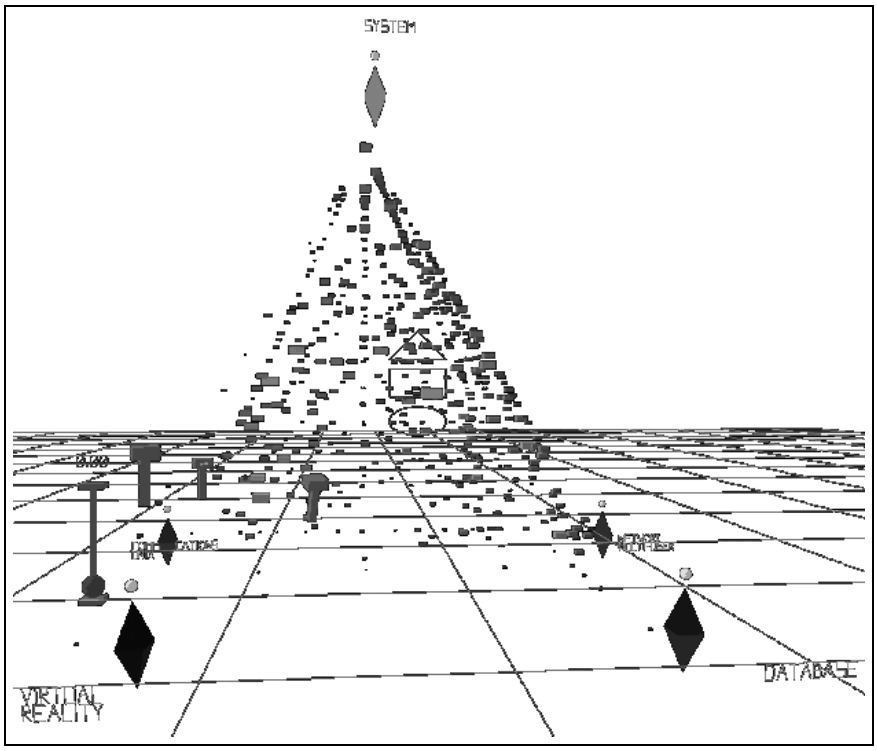


Figure 4.5: Multiple users exploring a VR-VIBE document space. Extracted from [VRV].

Both styles show the overall significance of each POI by varying its shade.

Figure 4.5 shows a visualisation using the 3D-layout style. Five POIs (displayed as octahedrons labelled with their keywords) have been placed at the corners of a pyramid. The size and the shade of the documents (represented as blocks) show the overall relevance. On the left side close to the plane three users are visible exploring the document space. For the embodiment of the users the standard DIVE embodiments, called blockies, are used. The arrangement of an oval, rectangle and triangle in the middle of the screen is also a DIVE feature for standard mouse navigation.

User Interaction

Users may freely navigate in the presented document space. They can select objects or may drag POIs around and thereby change the structure of the corpus. When an object is selected its title appears in the display as a text label and the colour of the document icon changes (see Figure 4.6). User can also mark documents by attaching labels for later reference or for other users. Furthermore, the users communicate with each other over a live audio channel. A three dimensional scrollbar shown in Figure 4.7 allows the users to specify a *relevance threshold* to filter the search result set by means of overall relevance.

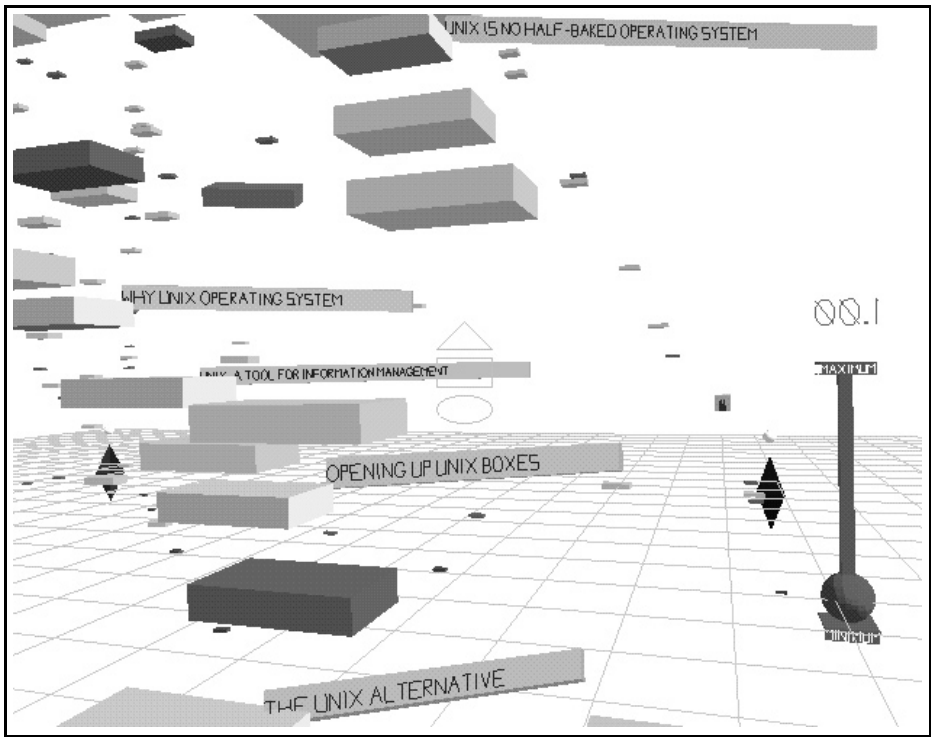


Figure 4.6: Labelled documents in VR-VIBE. Extracted from [VRV].

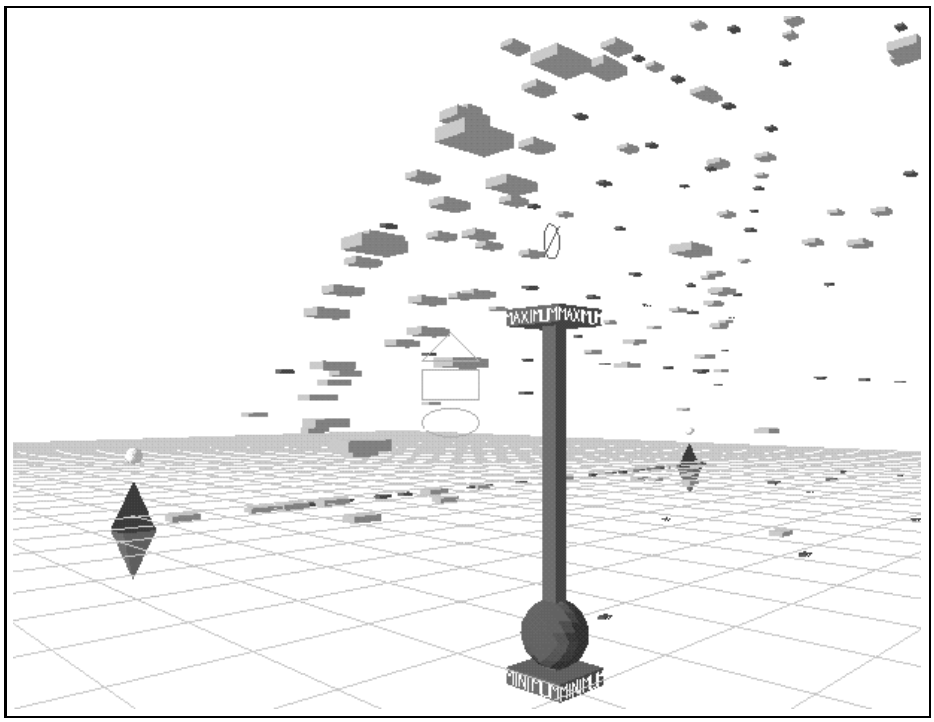


Figure 4.7: 3D-scroll bar to specify document relevance. Extracted from [VRV].

Once an interesting document has been found the user can view the whole document invoking an appropriate viewer. VR-VIBE is capable of invoking a World Wide Web browser to access documents that are available via the WWW [BSG+95]. Figure 4.8 shows an example of the NCSA Mosaic browser being used to inspect a web page.

Finally, instead of browsing through the document corpus, and selecting objects which look interesting, users may also input traditional key queries to find documents within the corpus which match the specified keywords. The matching documents are highlighted.

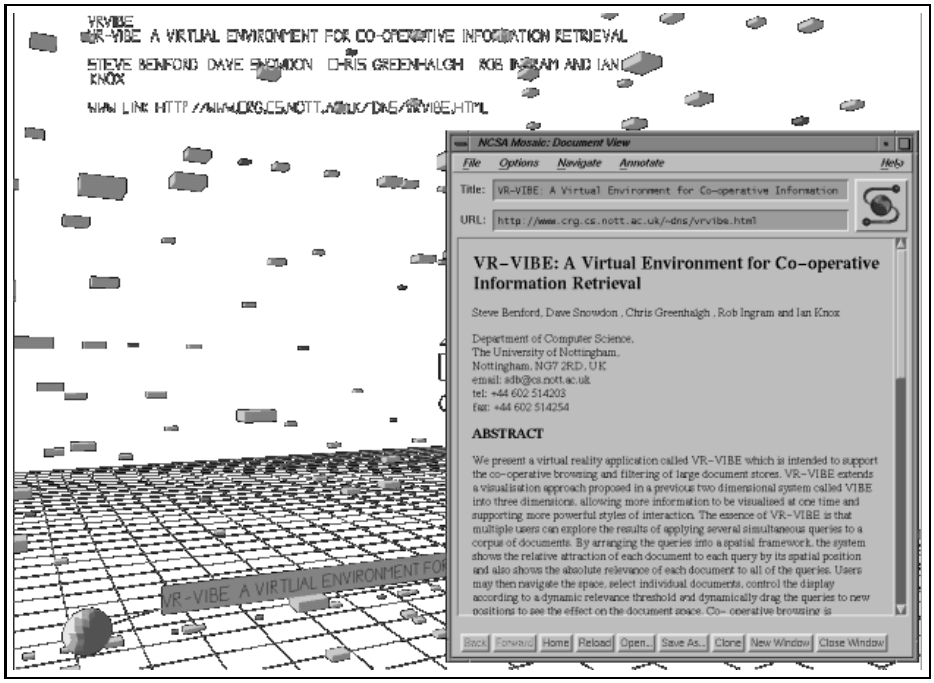


Figure 4.8: Using a web browser to view documents in the virtual reality. Extracted from [VRV].

Discussion

VR-VIBE presents a flexible way to visualise search result sets. Although it is intended to be used in cooperation with other users, it can serve as a single user application as well. Since it depends on the specification of queries and particular query terms the selection of suitable keywords may be as difficult as it is for traditional text based search systems. However, it supports the user in judging the significance of query terms which is a powerful aid to browsing.

In cooperative situations the problem of subjectivity versus objectivity has to be considered. VR-VIBE presents an objective world view to the users; i.e. every user sees the same objects in the same places displayed in the same way. For example, if one user selects a document it is highlighted in the views of all the other users

too. Text labels are problematic because users do not want their display cluttered if another user selects an object, as well as the marking of objects for other reasons than communication. That means that, for example, notes attached by one user for better orientation may not be necessarily useful for other users. The relevance filtering is another subjective issue since users may wish to set their own relevance threshold.

It is not clear if the benefits gained from cooperative information retrieval are worth the efforts which have to be made to provide this additional possibility.

4.5 LyberWorld

Based on an already existing probabilistic IR system a research team at the German National Research Center for Computer Science developed a prototype IR user interface called LyberWorld [Hem93, HKW94]. It is another approach to visualise the multi dimensional relations between documents and query terms. Two different tools support the user finding information of interest.

LyberTrees

In the LyberWorld an explicit query no longer exists. Instead of a query users specify their interests by browsing through the document space along a *content oriented search path*. The visualisation of the document space is accomplished using three dimensional *cone trees* called *LyberTrees*. The user starts at the root of the LyberTree and navigates through the document space by unfolding branches of these trees. An already known document or a specified keyword serves as an initial starting point.

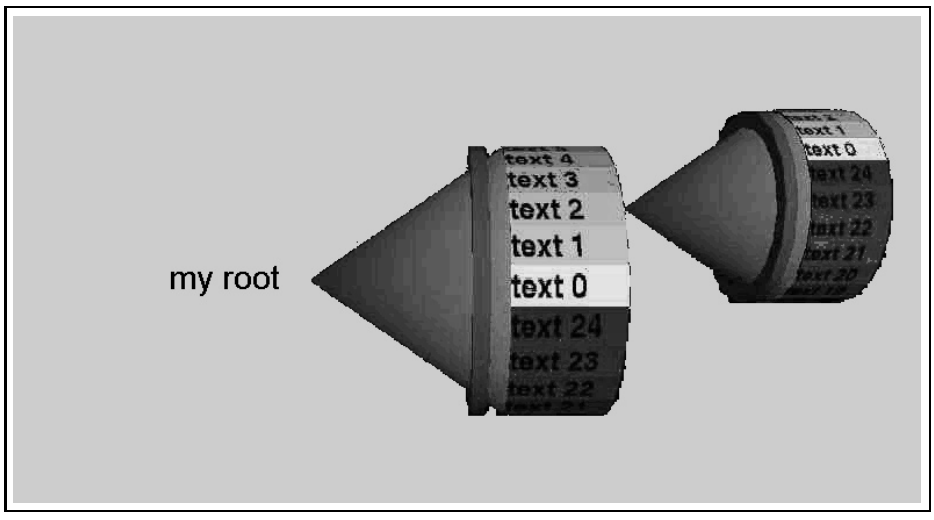


Figure 4.9: A simple LyberTree with two levels. Extracted from [LYW].

If the user types in a keyword the initial LyberTree consist of all the documents which contain that very keyword and if the user selects a document the cone tree is made up of all the terms contained in that document. Therefore two different levels are distinguished within a LyberTree; a word or term level, and a document level. They are displayed in different colours. For further descriptions it is assumed that the user has specified a keyword to start from.

The user may now rotate the initial cone tree and inspect the titles of the presented documents until one found document seems interesting. By selecting this document a branch is unfolded, going out from the document, which consists of all the words contained in that document (see Figure 4.9). The words are ordered according to their relevance to the document. Then again a branch of the LyberTree made up of documents may be opened and so on. The user browses through this tree structure and can move from level to level or the user may unfold several branches from the same level. If the user feels that the unfolded structure becomes to complicated the user can close already opened subtrees.

The so called *current view* is the tree level that is actually visited. If the user reaches a word or a document which already occurred in a previous level an animation moves the point of view back to the first occurrence of that item. This avoids *search loops* and provides a compacter visualisation. A more complex LyberTree structure is shown in Figure 4.10. The radius of a single cone tree depends on the number of contained items. If there are too many items the tree may also be unfolded as a spiral tree in order to save place. Furthermore, it is possible to zoom in and out to get an overview of the generated tree structure.

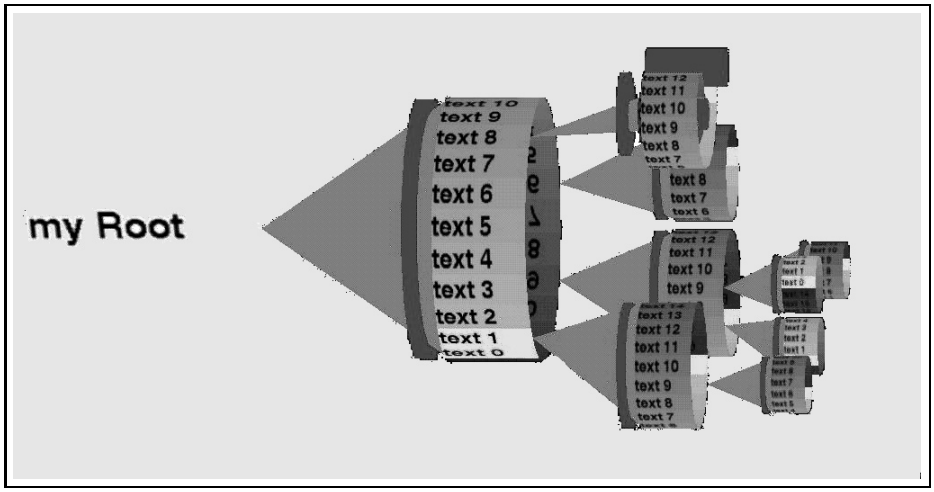


Figure 4.10: An example of a complex LyberTree structure. Extracted from [LYW].

To decide if a found document is relevant the user either can inspect the words contained in the subtree of that document or the user can view the document itself. To view the entire document the user leaves the tree structure and enters the *LyberRoom* which is a 3D visualisation of a room in which the current document is projected onto one of the walls. The room can be left through two doors. One

leads to the next document, and through the other one the user returns to the point where the user has come from.

The Relevance Sphere

At the point where users feel that they cannot find adequate documents of interest anymore, by moving around in the LyberTree structure, the users can use the *Relevance Sphere*. In association with the LyberTrees it is called *LyberWorld*. The system automatically generates a query out of the content oriented search path in the tree structure.

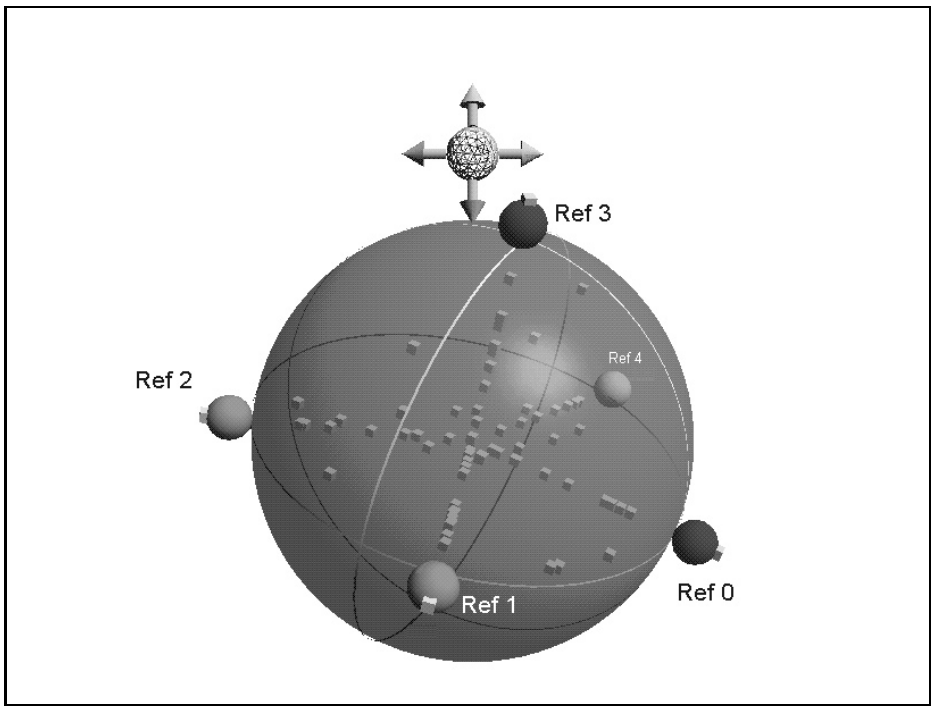


Figure 4.11: Relevance Sphere with 100 documents and five points of interest. Extracted from [LYW].

The retrieved documents are placed in the sphere and the search terms are placed on the surface of the sphere. The spatial position of the individual documents is defined by their relative attraction to the different terms. The mechanism is very similar to the one used by VR-VIBE (see Section 4.4). Again, each search term can be considered as a point of interest (POI). The radius of the sphere is computed with respect to the distances between documents and POIs or other documents. The users can change the position of these POIs on the surface to overcome the problem of ambiguous placement as described before. Furthermore, the Relevance Sphere provides a mechanism to decrease the document density in the sphere and it is possible to increase or decrease the relative attraction of every single POI. This enables users to find out easily which documents are related to a particular POI and

which are not. An example of a LyberWorld with 100 documents and five POIs is shown in Figure 4.11.

A additional partitioning mechanism for a LyberWorld document set is introduced in [HKW94]. It enables the user to specify a scaling factor. If there are clusters of documents close to the surface of the sphere and other groups of documents are more concentrated in the inner regions it is possible to divide the sphere in two disjunctive spheres. If all POIs are positioned on one side of the sphere it is possible to separate the more relevant documents from less relevant documents by using the scaling factor because in this case the most relevant objects are closest to the surface.

The users may immerse into the sphere and inspect single documents. They can rotate the sphere and watch how the movement of POIs influences the document set.

Discussion

LyberWorld is a sophisticated IR user interface which uses a wide range of visualisation techniques nowadays available. The approach to specify interests by browsing through the information space seems to be very intuitive. The necessity of a definite starting point restricts the possibility to get a general overview of the available information. Since the concept of the Relevance Sphere is similar to the visualisation approach of VR-VIBE it leads to the same problems of ambiguity. Therefore, it may be difficult to judge the overall relevance of individual documents which can only be deduced from an appropriate arrangement of the POIs. However, due to additional features, like the possibility to change the attraction of a single POI, it is easier to determine the relations between documents and query terms. Since documents, which are similarly attracted by POIs, are close by means of spatial position a natural clustering of the search result set is supported as well.

It has to be mentioned that all the tools described above can be used in combination. User may enter the LyberRoom whenever they want to view a single document and they can switch between the LyberWorld and the LyberTrees at any time. Hence, the LyberTrees serve also as a *query history tool.*

4.6 InfoCrystal

The approach of the InfoCrystal to visualise abstract information, such as document spaces, without explicit spatial properties, is different to the previous systems. The InfoCrystal visualises all the possible relationships of documents to N concepts using a smart extension of the Venn diagram [Spo93]. It enables the user to retrieve and explore information in an interactive way.

Using the Venn Diagram to Visualise Relationships

Figure 4.12 shows how to transform a Venn diagram into an InfoCrystal. The interior icons have the following Boolean meanings: 1=(A and (not (B or C))), 2=(A and C and (not B)), 3=(A and B and C), 4=(A and B and (not C)), 5=(C and (not(A or B))), 6=(B and C and (not A)), 7=(B and (not (A or C))). A, B, and C represent the query terms to which the retrieved documents are related. These icons are called criterion icons.

Various visual coding principles are used to set the position and the appearance of the icons:

- Shape Coding: indicates the number of criteria the interior icon is associated with; 1− > circle, 2− > rectangle, 3− > triangle, 4− > square, and so on.
- Proximity Coding: The closer an interior icon is placed to a criterion the more related are the contained documents of that particular icon.
- Rank Coding: icons with the same shape are placed on invisible concentric circles. The smaller the radius the more criteria are satisfied by the interior icons on that circle.
- Colour and Texture Coding: is used to indicate to which criteria an interior icon is related to.
- Orientation Coding: the icons are oriented so that their sides face the criteria they satisfy.
- Size or Brightness & Saturation Coding: shows quantitative information like the number of documents represented by an icon.

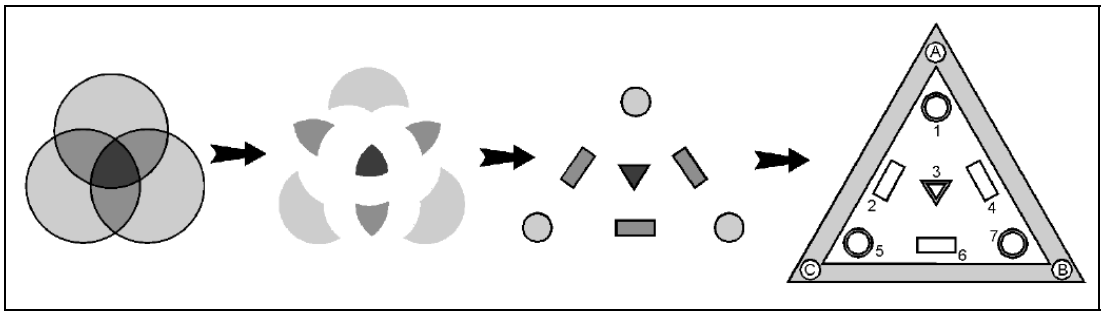


Figure 4.12: A Venn Diagram transformed into an InfoCrystal.

Furthermore, the number of contained documents is written into the corresponding icon. Figure 4.13 shows an example for an InfoCrystal with four criteria.

In addition to this *rank layout*, that places the interior icons which correspond to a higher number of criteria towards the centre of the InfoCrystal, a second layout style, called the *bull's-eye layout* exists. A polar transform is used to place more

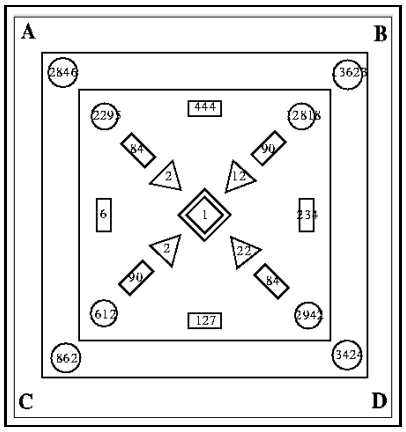


Figure 4.13: An InfoCrystal with four criteria.

relevant interior icons or documents closer to the centre. This approach is similar to the method used by the VIBE System (see Section 4.4). Thus, an interior icon is placed closer to related criterion icons than to not related ones.

Visual Queries

Since each interior icon represents a distinct Boolean relationship they may be used to specify Boolean queries. The user can easily formulate queries graphically by selecting sets of interior icons. Moreover, InfoCrystals may be combined and organised in a hierarchical structure to create complex queries. The output of an InfoCrystal (i.e. the document set represented by selected interior icons) is used as input for the InfoCrystal at the next hierarchy level. The user can ask "what-if" questions by changing the selection of icons in one InfoCrystal and observing the change of the content of the icons in the InfoCrystals which are higher up in the hierarchy.

In the case of full text retrieval the user may adjust the relevance of an individual criterion by a slider with a range from 1 to -1. Negative weights indicate that the user is more interested in documents which do not contain the concept at the particular input and a -1 is equal to a logical NOT. A threshold slider provides the possibility to filter out less relevant documents. If an interior icon contains only documents which are less relevant than the specified relevance threshold it is not possible to select it.

Discussion

Although the InfoCrystal does not visualise the search result set in three dimensions it shows an excellent way to display multiple relations between documents and query terms. Furthermore, graphical composition of Boolean and full text queries is supported, but complex queries might need too much screen space to be effective, even if a hierarchical structure of InfoCrystals is generated.

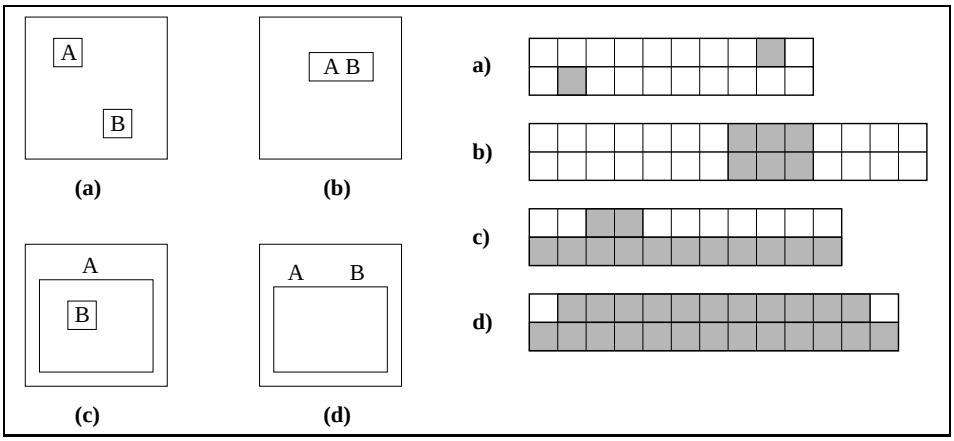


Figure 4.14: Possible term distributions and their corresponding TileBar representations.

4.7 TileBars

The concept of TileBars addresses the problem of query term distribution in text documents. Using only an overall relevance to rank search result lists may be problematic or even wrong especially if individual text documents are reasonably long.

To address this, the TileBars paradigm provides a compact and informative representation of the documents' contents with respect to the query terms [Hea95].

Document Structure and Representation

For a set of short documents it is reasonable to assume that documents in which the query terms occur more frequently are more relevant. However, there are many ways in which a long text can be similar to a query. A long text often consist of many different subsections with different subtopics. These topics may be related or not and therefore the query terms may occur in the same context or not. For that reason the TileBars representation simultaneously displays:

1. The relative length of the document.

2. The frequency of the term sets in the document.

3. The distribution of term sets with respect to the document and to each other.

Figure 4.14 shows four possible term distributions within a text document and the corresponding TileBar representations. In case a) the distribution is disjoint and in case b) both query terms are discussed locally. If term A occurs in the whole text term B may be discussed locally (c) or may also occur globally throughout the text (d).

To determine the internal structure of a text document an algorithm, called *TextTiling*, has been developed. It detects subtopic boundaries by analysing the term repetition patterns within the text.

The relative length of documents is indicated by the length of the corresponding TileBar representation. Since the TileBars are left aligned it is easy to compare the length of one document to the length of other documents. The term distribution is presented by the non-overlapping squares, or Text Tiles, within the document icon. The darker a Text Tile is the more frequent occurs the query term in the particular section. Actually it may not be a single query term but a query term set, in which the terms are combined with a logical OR. Between two query term sets there is an implicit AND. Figure 4.15 shows a screen shot of an application using TileBars to present a search result.

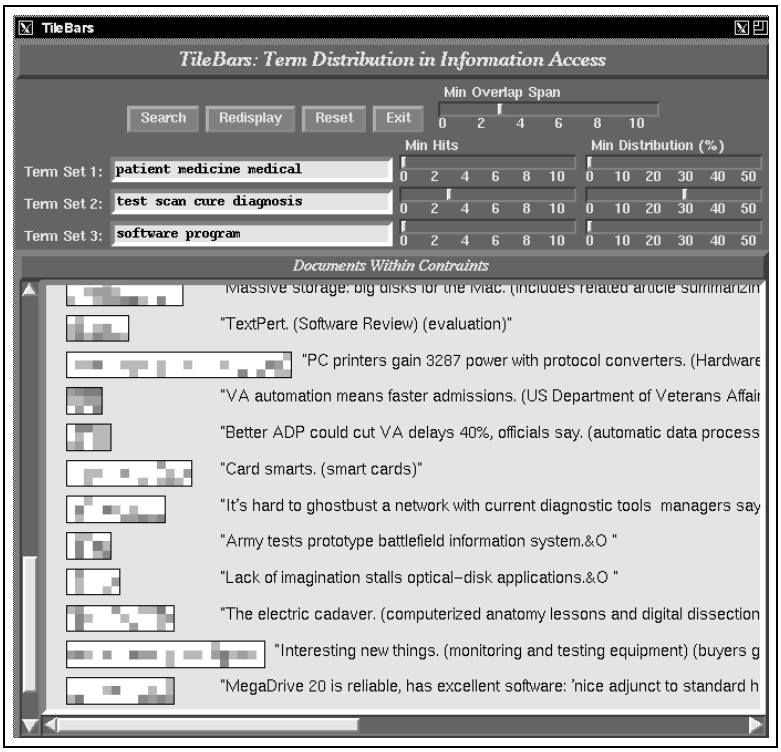


Figure 4.15: Search result list with TileBars. Extracted from [UCB].

Discussion

The TileBars paradigm is directed towards a different aspect of visualising search result sets than the other described search systems described in this chapter. TileBars help the user to estimate the relevance of documents before viewing them. If interesting documents are found the user is enabled to go directly to the relevant sections within the documents.

On the other hand, there are disadvantages which are common to all search systems which present the search result only as a ranked list. The user neither

gets a general overview of the available information, nor is the finding of related documents supported. Furthermore, because it is query based, all advantages and disadvantages of query composition are involved.

4.8 Envision

Envision visualises search results by graphically presenting various document characteristics in addition to ranked query document similarity. It is an application developed for searching in digital libraries. The search result is displayed as a matrix of icons with layout semantics under user control [NFH+96]. It addresses typical user tasks, determined through user interviews, including:

- Identify trends in the literature, spotting emerging topics of research, as well as identifying peaks and valleys of research interest in topics.
- Locate highly influential works which have been frequently cited by others.
- Identify relationships among research topics that were not apparent.
- Discover communities of discourse in which authors regularly cite and respond to another's work to form an ongoing conversation in print.

Envision supports full text retrieval and relevance feedback.

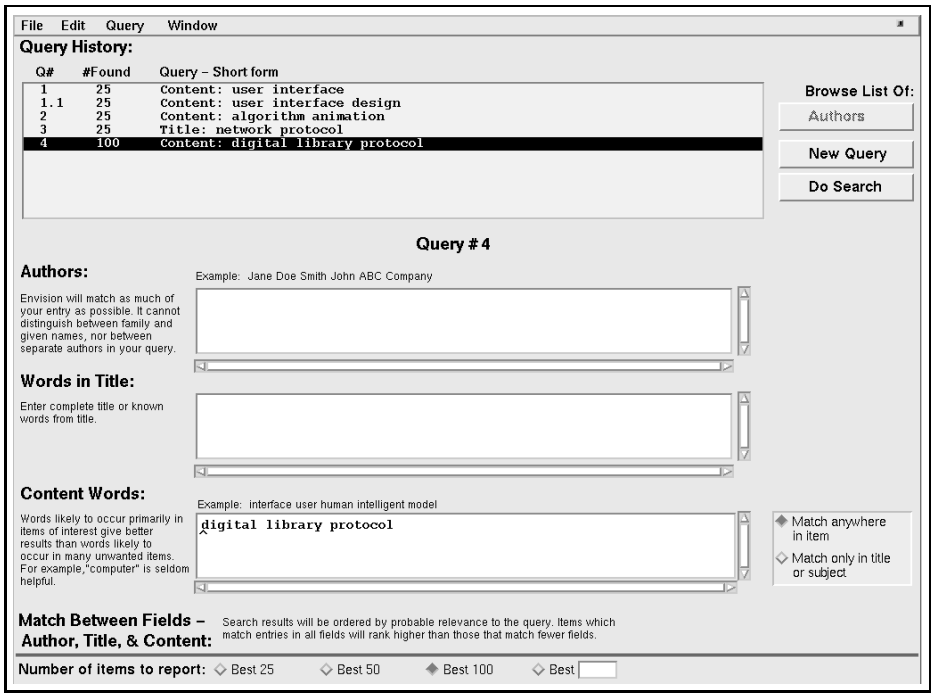


Figure 4.16: Envision's Query Window. Extracted from [EEL].

User Interface and Search Result Visualisation

The user interface comprises three different windows. The *Item Summary Window* displays a text description of documents whose icons are selected or marked as useful in the *Graphic View Window*. Selected icons are indicated by bold icon labels and marked icons are indicated by surrounding boxes. A double-click on an icon or a line in the Item Summary Window invokes XMosaic to access that document entry in the Envision database.

The *Query Window* is used to compose queries by entering query terms into the appropriate fields. It is possible to search for authors, titles and for content words. It also provides a Query History (see Figure 4.16).

The search result set is displayed in the *Graphic View Window* (see Figure 4.17). The users can adapt the visualisation to their needs. They control size, colour, shape and labels of the document icons as well as the semantics of the x-axis and the y-axis. Each axis may indicate estimated document relevance (i.e. overall relevance), author names, index terms, document types, and publication year. The document icons are placed accordingly to the semantics of the two axes. The icon attributes may be chosen as follows:

- Size: may be uniform or may indicate document relevance.
- Colour: may be uniform or indicates document relevance or the document type.
- Shape: shows the document type or the document relevance or may be uniform.
- Label: the icon label either shows the relevance rank or an unique Envision document identifier.

If icons cannot be displayed non-overlapping they are represented by an elliptical cluster icon. The number of contained documents is shown within the icon and the label presents the ranks or the identifiers of the two most relevant documents. The colour of the cluster icon corresponds to the relevance of the highest ranked document in that "cluster".

Discussion

The user interface of Envision is very flexible and the visualisation of the search result set is clearly arranged. The specialisation in dealing with digital libraries facilitates the visualisation of search results because the documents and objects in the database are structured similarly. As discussed in the section about the TileBars paradigm (see 4.7) the *estimated relevance* could be problematic for books and other large documents which are probably contained in a digital library. There exist cluster icons, but that name may be misleading in terms of document clustering, since two documents from the same author published in the same year may not necessarily

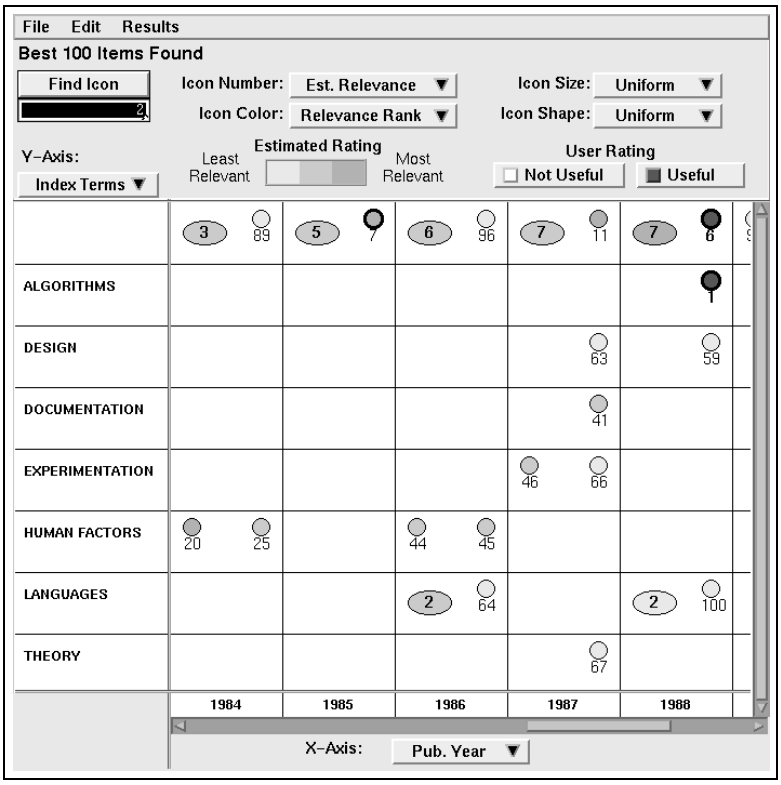


Figure 4.17: A search result in the Graphic View Window. Extracted from [EEL].

deal with the same topic and therefore may not be related. Moreover, relationships between documents are actually not visualised apart from attribute similarities.

Another major limitation of Envision is that it currently visualises only index terms or keywords that have been assigned by authors or editors, especially since the underlying system supports full text retrieval. In addition, Envision currently displays document icons only in relation to one index term and one author although a single document may correspond to several index terms or may be written by two or more authors. It is an issue of future work how multiple relations can be visualised without cluttering the display and irritating the user. Some attributes like the "frequency of citation", as demanded in the specification, and other useful ones like document size, are not yet implemented.

5

The Web, Hyperwave, and Harmony

This chapter focuses on the main concepts of the Internet and introduces Hyperwave, the next generation Web solution, and its client Harmony.

5.1 The Internet

One of the reasons that so much information is available to search for is the Internet, the worldwide computer network with millions of users. It started as a research project of the US Department of Defense's *Advanced Research Projects Agency* (ARPA) and grew to this huge network it is today. The estimated growth rate is between 10 to 15% per month or around 100% per year. The recent popularity of the World Wide Web on the Internet introduced the term *The Web* as a synonym for all the services available on the Internet.

5.1.1 Connections via the Internet

To connect different computers with different operating systems there has to be a standardised protocol which is common to all connected systems. In case of the Internet these protocols are the TCP (Transmission Control Protocol) and the IP (Internet Protocol) which serve also as base for several other protocols.

The Internet is a packet switched network, that is data packages are sent from one computer to another until they reach their destination without a direct connection between the sender and the recipient. For that reason every computer on the Internet has a unique address (or IP-number). These addresses are 32-bit numbers, in four 8-bit parts, for example 129.27.2.3. The IP provides these packages which contain the address of the sender and the receiver. The computers in between know where a packet came from and where to send it. They are also called routers since they send the packages on routes through the Internet. The routes are not necessarily the same for all packages because they are chosen dynamically by the routers.

The TCP builds up a virtual connection. It breaks messages into packages, which are then send via IP, and collects the packages on the receiving side. Since the packages may be send on different routes they may arrive out of order or some packages may even be lost. These problems are covered by the TCP and are transparent to the computer applications using these virtual connections.

The IP addresses are rather long and therefore hard to remember. For that reason a hierarchical naming scheme, the *Domain Name Service* (DNS), is available as an alternatively way to address computers on the Internet. Each level is called a domain. The name of a computer is put together from the domain hierarchy. For example the computer with the name *fiicmal03.tu-graz.ac.at* is an Alpha workstation in the domain *tu-graz* which is in the domain *ac* which itself is a subdomain of the domain *at*. Every domain has a *name-server* which resolves the names (several names may refer to the same computer) into the unique IP-number.

Most Internet services use the client-server concept. Client programs are started by the users on their local machines. The client establishes a connection to a server on the remote machine. If the server is not running no connection can be made.

What kind of data is transmitted if a connection is established depends again on a protocol which is usually based on TCP/IP. The most common used protocols are the *Telnet* protocol which allows a remote login into another computer on the Internet. The *FTP* (File Transfer Protocol) allows, as the name says, the transfer of files from one machine to another. The *SMTP* (Simple Mail Transfer Protocol) is used to send emails and the *NNTP* (Network News Transfer Protocol) is needed to provide network news. News are discussion groups in which the Internet users can contribute to discussion of many different topics. Last but not least the *HTTP* (HyperText Transfer Protocol) is used for WWW connections (see Section 5.1.2). To distinguish between the different services a specified *port number* for each service is used in addition to the IP-address.

5.1.2 The World Wide Web

The World Wide Web (WWW or W3) brought together the concept of hypertext and multimedia on the Internet. It is a *distributed, heterogeneous, hypermedia information system* and is based on three key specifications: *HTML*, *HTTP*, and *URL* [Mau96].

HTML (HyperText Markup Language) defines how documents look like on the WWW. It is an SGML-conformant mark up language; the final presentation of the document depends on the individual WWW client. HTTP is WWW's stateless client-server protocol. The URL (Uniform Resource Locator) specifies the location of a resource. Web pages and various other Internet services can be addressed via a URL.

Hypertext and Hypermedia

Hypertext is text which is not constrained to be linear. Instead of reading page by page in a sequential order the reader can follow hyperlinks within the current document. Words, sentences, or paragraphs may be linked to any other part of the same document or to any other document in the document base. The term hypertext was coined by Ted Nelson around 1965. Figure 5.1 shows the difference between linear text and hypertext.

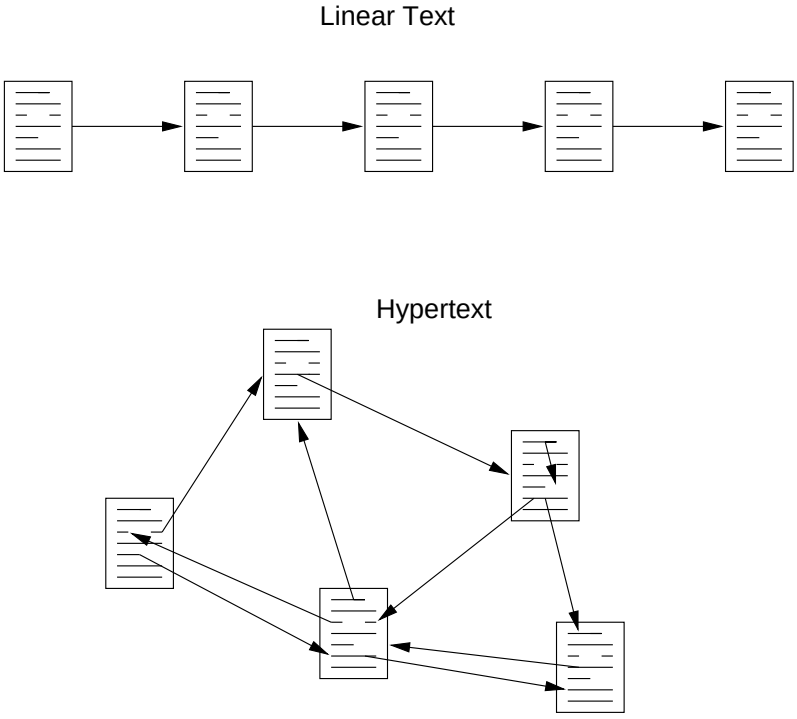


Figure 5.1: Linear text and hypertext.

Hypermedia is the generalisation of hypertext to include other kinds of media: images, audio clips and video clips are typically supported in addition to text. Individual chunks of information are usually referred to as documents or nodes, and the connections between them as links or hyperlinks – the so-called *node-link hypermedia model*. The entire set of nodes and links form a graph network. A distinct set of nodes and links which constitutes a logical entity or work is called a *hyperdocument*; a distinct subset of hyperlinks is often called a *hyperweb* [Mau96].

In the WWW a URL is used to define a hyperlink. The URL can be attached to a word or a part of an image which represents afterwards the starting point of a hyperlink. This starting point is marked as a link and the user can usually follow this link with a mouse-click and access information all over the world. Following these links is often called Web surfing.

5.2 Hyperwave

Hyperwave, formerly called Hyper-G [Mau96], is a hypermedia document management system and provides a next generation Web solution. It is developed at the Institute for Information Processing and Computer Supported New Media at Graz University of Technology. It is called a next generation hypermedia systems since it overcomes several problems which arise from the nowadays large scale of the Internet and the weaknesses in the system architecture of first-generation systems.

5.2.1 Problems of First Generation Information Systems on the WWW

The different problems nowadays on the WWW can roughly be divided into three groups: those that are faced by the consumers of information (the users), those that involve the information provider (the authors), and general weaknesses of the system architecture. The main problems are:

- Getting lost in Hyperspace; following links is easy, but to find the way back to the starting point and to keep the orientation can be very difficult without additional navigation and orientation aids. Another reason for that is, that the hyperlinks only point in one direction, and therefore going back to a previous document is impossible if there is no explicit link back or if the browser has not saved the document where the user has come from. With first-generation systems going back is more like starting again at the beginning.

- Link consistency; due to the fact that links are embedded into the documents and that the URL statically addresses a certain object it often happens that the URLs point to nowhere (dangling links) or even worse point to another object. In case that a document is transfered to another place all documents which contain links to that particular document have to be updated. This is a serious problem for providers who want to reorganise their Web server.

- There is no feedback about how much information 'hides' behind a link.

- A version control is missing. It is sometimes unclear what information has already been seen. Although, most Web browsers mark already followed links to avoid that the user follows a link again and again, there is no mechanism which unmarks a link if the information behind has been changed.

- It is often difficult to separate new information from old information. The user usually has the problem to find the new information while it is even a greater problem for a provider to select the documents, which is not up to date anymore, for deletion. If there is no support from the information system this has to be done 'by hand'.

- There are no practical access control mechanisms. It is hardly possible to provide information only for certain users or user groups. Especially if multiple

authors edit the same documents they should have access rights while normal users should not. Therefore, the information system should provide a kind of user identification.

- Multiple languages are not supported. If an author wants to provide information in multiple languages separate documents and links are required.
- Full text search mechanisms are not supported on a usual Web server or have to be implemented by the provider. There exist huge search engines but these engines have to contact every server worldwide to find new information. This increases the load on the Web considerably.
- The HTTP is connectionless and therefore also stateless, which means that the client establishes a connection and sends a request. The server handles the request and returns the requested information. Afterwards the connection is closed. This is easier to implement than a stateful connection, but on one side very costly, by means of network load, and slow because a new connection has to be established for every request. On the other side all information about a previous request is lost or has to be retransmitted.
- Scalability; if many users address the same URL at the same time they are also addressing the same server and it is difficult to distribute this request among several servers.

5.2.2 The Hyperwave Server

The Hyperwave server architecture is directed to a multi-user concept to provide and manage large scale hypermedia information bases. Hyperwave offers gateways to existing information systems like the WWW or Gopher. However, the most users will access a Hyperwave server using a Web browser like Netscape or the Microsoft Internet Explorer. In this case the Hyperwave server behaves like a "standard" Web server with some additional functionality. Figure 5.2 shows Hyperwave collections seen through Netscape via the WWW gateway. The additional buttons at the top of the Web page are inserted by the Hyperwave server to access the additional functions.

Information providers will typically use a Hyperwave client, which is also an authoring tool, to maintain the server content and structure. For Hyperwave clients the server provides the HG-CSP (Hyper-G Client-Server Protocol [Mau96]). The main properties of Hyperwave can be summarised as follows:

- Every hypermedia document is stored in an object-oriented database. In addition, the documents are inserted into the full text index to support later full text retrieval.
- The documents are structured in a collection hierarchy. Every document must be the member of a collection which could also be a member of one or more parent collections. At top level there is the server's root collection. The

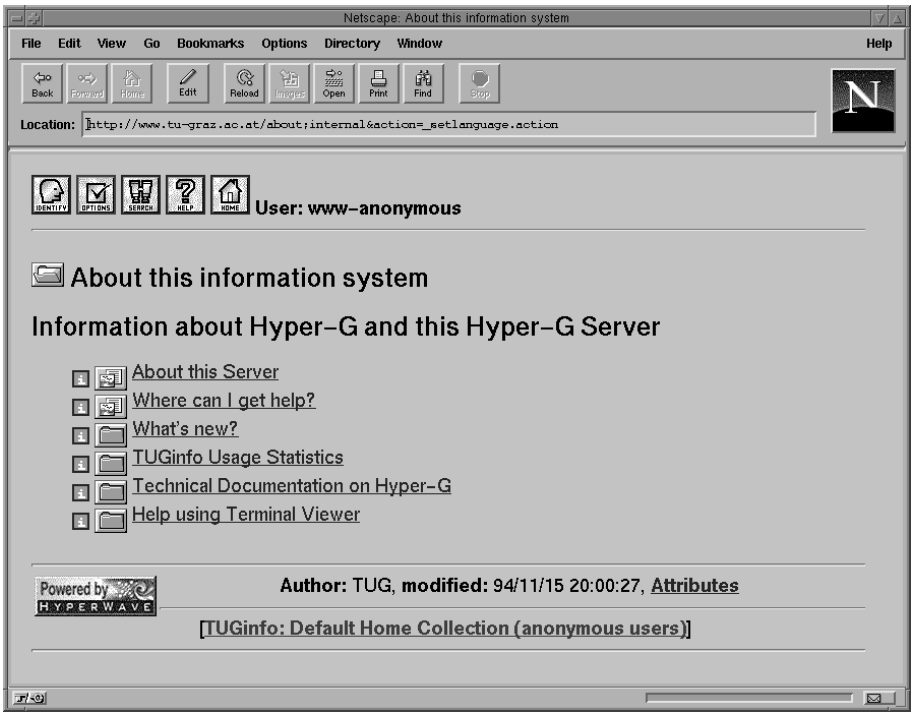


Figure 5.2: Hyperwave collections seen through Netscape.

collection hierarchy is a directed acyclic graph (DAG). It provides continuous orientation and navigation aids.

- Document attributes are stored separately. This support quick attribute search activities.
- The search scope can be limited to a number of collections or the local server which allows more specific searching.
- Links are separate objects and stored in a separate link database. Therefore, links are bi-directional and the link consistency is maintained by the system.
- Links may refer from and to any document type, that is there are not only links from and to texts or images but also to and from video and audio clips, postscript documents, or 3D scenes.
- Hyperwave clients may insert links interactively.
- There exists an access control mechanism which allows to attach access permissions to every document. Certain documents may only be read by identified users with appropriate access right. The users may be members of different user groups which also may have different access rights.
- Documents can be grouped together to clusters. Clusters are subtypes of collections and are used to create compound documents. For example an image and the additional description are presented at once when accessing the

cluster. Furthermore, if there are several images of different image formats the appropriate image is selected according to the display capabilities.

- Clusters support multiple languages. One cluster may contain the same text in different languages. If users access that cluster they only see the document in one language depending on the language settings.
- Collections may also be transformed into a sequence collection. The members of that collection are then presented in a sequential order without the necessity of additional links.
- Hyperwave uses a connection-oriented protocol which results in a more efficient connection but requires a more complex server architecture. The already transmitted data is not lost during a session.
- Hyperwave uses the "proxy" architecture, which means, that a client only connects to one Hyperwave server, the local server, during a whole session. If documents from a remote servers are required the local server connects to the remote server instead of the client. This offers various advantages, such as a reduction of network load, which are described in detail in [Mau96].

Additional information about Hyperwave can be found in [DH96] and on the Internet at `http://www.iicm.edu`.

5.3 The Harmony Client

Harmony is the Unix client and authoring tool for Hyperwave. The main window of Harmony, the session manager, presents the hierarchical Hyperwave collection structure in the form of a tree (see Figure 5.3).

Furthermore, two other visualisations of the collection structure are available. The Harmony Information Landscape (see Figure 5.5) presents a three-dimensional landscape view of the collection structure and the Harmony Local Map (see Figure 5.4) is a dynamic, two-dimensional structure map which visualises the local relationships of a particular collection or document (for a detailed description see [And96]).

The user can browse through the Hyperwave space to find information of interest. A single mouse click selects the object and causes the drawing of a frame around that object. A double-click opens a collection or starts an appropriate viewer for the particular document. Icons of already viewed documents are marked with a tick. A selected collection may be activated for focused searching. This means it is added to the set of active collections.

The user gets an immediate feedback for every action in the other views. For example, if a document is selected in the collection browser then it is also framed in the Harmony Local Map and vice versa. Furthermore, the collection browser shows the location of a document which has been selected in one of the other views. This orientation mechanism is called *location feedback*.

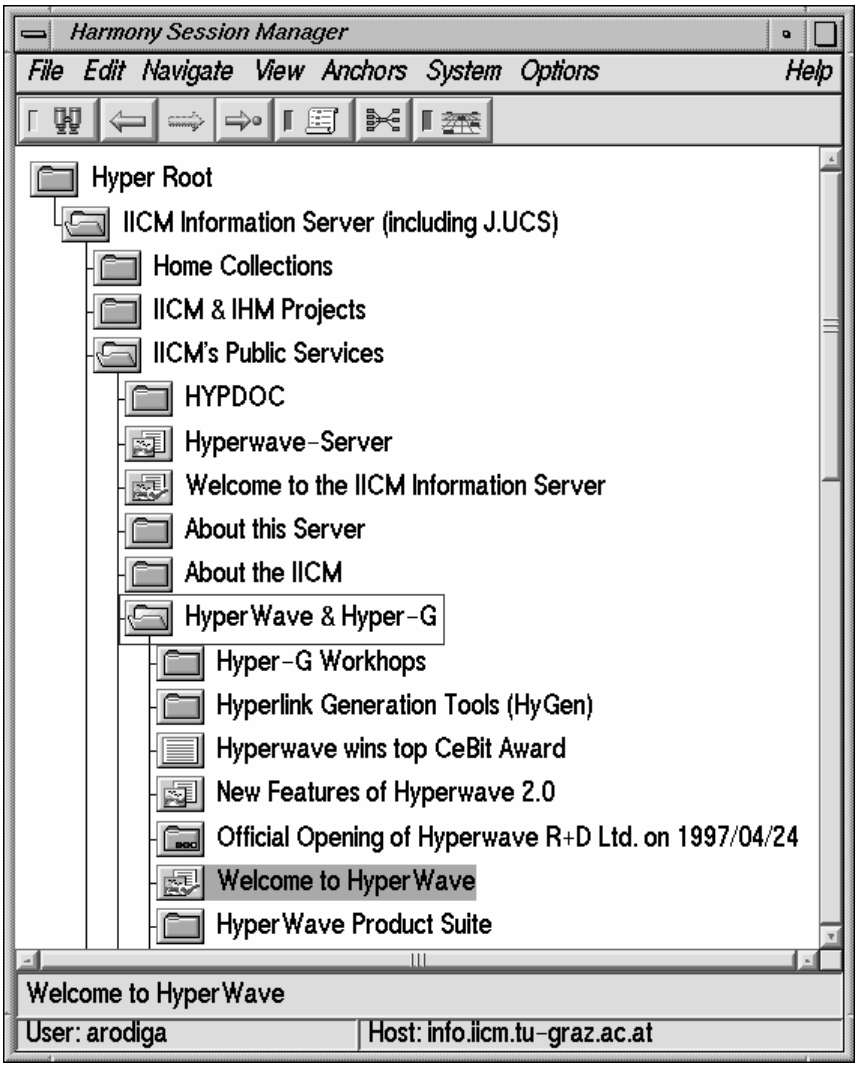


Figure 5.3: The Harmony session manager.

Harmony is not only a browser but also an authoring tool. It can be used to organise and maintain the document structure, to insert documents, to edit documents and documents attributes, to move and copy documents, to assign rights to documents and much more. Harmony is the Hyperwave client that supports most of the features provided by the Hyperwave server. Furthermore, the functionality of Harmony can be extended via an Application Program Interface (see Section 8.1).

5.3.1 Searching in Harmony

Beside browsing through the collection hierarchy the user can search for specific documents. For that reason, Harmony provides a separate search interface. It can be opened and closed via the pull-down menu or by clicking on the button with the binoculars symbol.

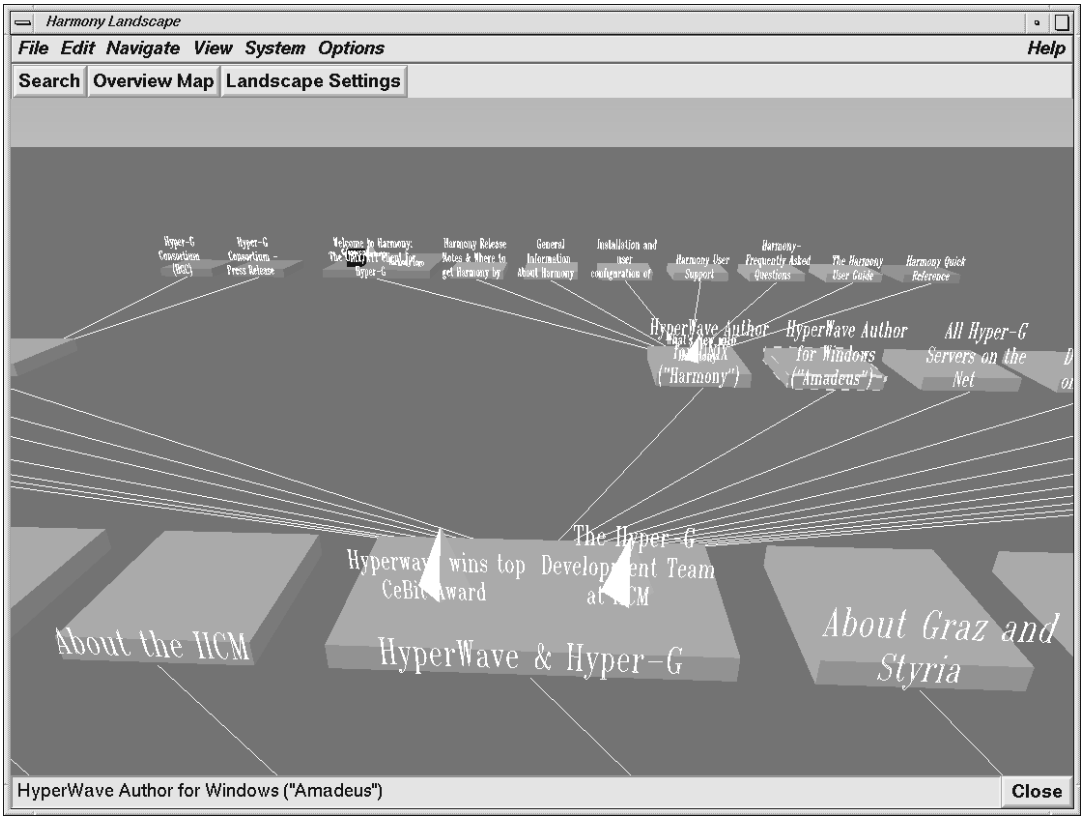


Figure 5.4: The Harmony Landscape.

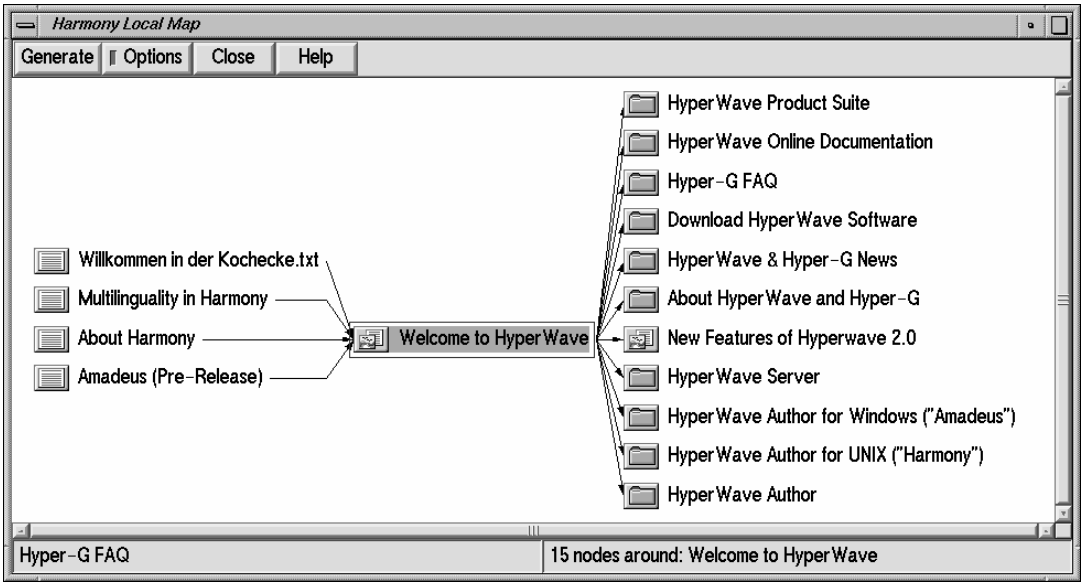


Figure 5.5: The Harmony Local Map.

The search interface is made up of several check-boxes and a few fields in which the users may type in the search terms to specify the documents they are searching for (see Figure 5.6).

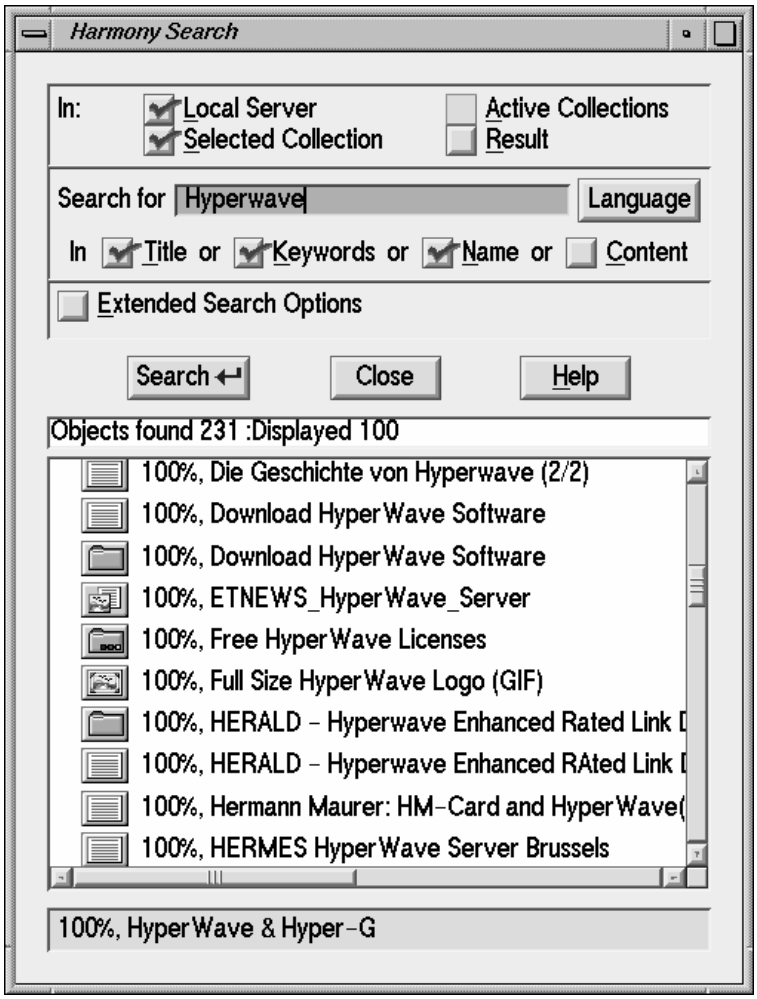


Figure 5.6: The search interface of Harmony.

The check boxes at the top are used to choose where the Hyperwave system should search for documents. It is possible to search in the local server, the currently selected collection, previously activated collections, and in an already retrieved search result set. Searching in the result set excludes an additional search in any collection or the local server, while otherwise every combination can be used.

Under the check boxes which specify the search location is the actual query field. The user may enter several query terms which can be combined with the Boolean operators AND, OR, and ANDNOT. The use of a simple blank to separate the different terms implies a logical AND. Parenthesis may be used to enter more complex queries. The 'Language' button beside the query field opens a dialogue to select the search languages. Since the Hyperwave system is multi-lingual a document may have several titles in different languages.

Hyperwave supports Boolean attribute retrieval and full text retrieval (see also Section 8.2). To use these capabilities in Harmony the appropriate check-boxes in the search interface have to be selected or the query terms have to be filled into the different text fields

To apply an attribute search the user can choose from five document attributes. These are Title, Keyword, Name,Author, and TimeModified. The appropriate check-box has to be selected to search in the titles, the keywords and the name of the documents. The author can be entered into the author field in the *Extended Search Options* section of the interface. Another extended search option is the modification time. It has to be entered in the Hyperwave date format (e.g. 90/01/31 or 95/12/31 12:34:05). It is possible to search for objects which are modified before a certain time, after a certain time, and in a defined time interval. The Extended Search Options section can be opened or closed dynamically by clicking on the Extended Search Options check-box.

A full text search can be performed by selecting the *Content* check-box. Currently, only text documents are full text indexed and hence retrievable through content search. In the future, Microsoft Word and PDF documents may also be indexed.

Clicking the SSearch"button initiates the actual search task. The search result set is presented as a ranked list in the search result window at the bottom of the search interface. The first 100 highest ranked documents are displayed. Since the document attributes either satisfy the Boolean query or not the documents found by an attribute or Boolean search are always ranked with 100% relevance. The documents found by a full text search are ranked according to the relevance to the query terms.

It has to be mentioned that a full text search is only useful to find text documents while an attribute search can be applied to retrieve any type of document (texts, images, movies, audio etc.) and collections and link objects.

All check-boxes can be toggled via keyboard short cuts to speed up user interaction.

6

The HarSearch Interface

HarSearch is a new search tool for Hyperwave. It extends the search capabilities of Harmony and introduces an innovative approach to make sense of search result sets. The interface to HarSearch and its Similarity Map is the topic of this chapter.

6.1 The Features of the HarSearch Interface

HarSearch cooperates with Harmony in retrieving and presenting the search result set. For reasons of consistency, the appearance of the HarSearch interface is similar to the search interface of Harmony. However, HarSearch employs more functionality of the Hyperwave server and offers a more flexible way to search in the Hyperspace. A detailed user guide can be found in Appendix A. The features of HarSearch are:

- HarSearch provides three different interfaces tailored for the different user types (novice, skilled searcher, power user) and different search activities:
  - The simple query interface: for simple queries and users unfamiliar with HarSearch,Hyperwave, and Harmony. The simple query interface is easy to use but hides a lot of Hyperwave functionality.
  - The extended query interface: for extended attribute and full text retrieval. Every indexed attribute may be used to retrieve specific documents, and each additional document attribute can be taken for further filtering of the search result set.
  - The power query interface: offers power users the opportunity to enter plain Hyperwave queries for full text and attribute retrieval as well as for document filtering via object queries (see Section 8.2).
- Controlled scope of searches: search activities may be restricted to regions in the Hyperwave collection hierarchy. Users can search for documents residing on the local server, within the current collection, and also within the active collections.

- HarSearch is multilingual. The interface language corresponds to the language selection in Harmony. English and German are supported at the moment.

- Every language supported by Hyperwave can be used as a search language in HarSearch and is selected in a separate dialog.

- The maximum number of matching documents to display can be configured. All documents of the result set may be inspected but they are presented in subsets of documents with a user defined number of documents.

- Document type selection: a particular document type may be retrieved (e.g. if the user wants to find only images).

- Every query is stored in the query history. This allows the reuse of previous queries and helps the user modifying queries if the search result was not satisfying. The query history shows the query number, the number of found documents, the type of the query (i.e. simple-, extended-, or power-query), and a short description of the query. This allows the judgement of the effectivity of particular queries.

- In addition to the attributes which are already supported by the Harmony search interface (i.e. Title, Keyword, Author, Name, and TimeModified) HarSearch enables the user to search for the document attributes TimeCreated, DocAuthor, DocDate, and UserAtts.

- Special date editors allows the easy input of dates and times.

- The search result set can be filtered by any other document attribute using the query extension field if the particular attribute is not directly supported by HarSearch

- The list with the search result set is displayed in an extra window. The whole document set can be inspected by going through the list using previous and next buttons. The result list shows the document type, the overall relevance of the particular document and the title of the document.

- The result list supports location feedback (see Section 5.3). Documents selected in the result list are also selected in the Harmony collection browser and vice versa. Of course, selected documents in Harmony can only be shown in the result list if the selected document is contained in the search result set.

- The sort order of the result list can be specified. It is possible to sort the retrieved documents by score, by title, by author, by creation time, and by document type.

- Keyboard short cuts are supported to speed up user interactions.

6.2 The Interface of the HarSearch Similarity Map

The *HarSearch Similarity Map* visualises the mutual similarity relations between documents of the search result set. The similarities between documents are visualised in form of a tree in which the documents are displayed as icons and the relations between the documents are displayed as lines.

The Similarity Map interface allows the adjustment and setting of various layout parameters to adapt the layout to the user's preference. The features of the interface are listed below:

- The user may select between four different layout types. The distances between icons are adjustable.
- To represent the documents four different icon styles are available. The icons may be simple, scaled according to the overall relevance of the particular document, coloured according to the overall relevance, and scaled and coloured. The icon colours may be specified by the user using a separate colour dialog.
- The similarities between documents are indicated by lines. The user can choose from four different line styles. Similar to the icon styles the four line styles range from simple lines, width coded lines, colour coded lines, and width and colour coded lines. If the simple line style is chosen every similarity value is indicated by a thin line. Colour coded lines have three different colours according to the degree of similarity. Width coded means that high similarity values are indicated by thick lines, medium similarity values by thin lines, and low similarity values by dotted lines. The width and colour coding combines the last two mechanisms.
- The HarSearch Similarity Map shows clusters of documents in the search result set. The connectivity threshold may be chosen dynamically using a slider.
- Users can define whether documents not belonging to a document cluster, due to a high connectivity threshold, are either displayed or not.
- Another option of the Similarity Map is to show only levels of documents which are maximally related to a selected document. This allows a local exploration around a certain document.
- All similarities of documents, contained in the Similarity Map, to a selected document can be presented as a list in a separate window.
- The HarSearch Similarity Map and also the presentation of the similarities to a certain document support the location feedback mechanism (described in Section 5.3).
- As part of HarSearch the interface of the Similarity Map also supports different interface languages.

- Similarities are currently simulated, since document similarity is not yet supported by the Hyperwave server.

Furthermore, the interface allows the specification of the number of documents which are finally displayed in the Similarity Map. Since a query may retrieve thousands of documents the Similarity Map is restricted to this given number of the first documents in the search result set. The underlying concept of the Similarity Map, and how it visualises search result sets is discussed in Chapter 7.

7

The Design of the HarSearch Similarity Map

The HarSearch Similarity Map uses a maximum similarity spanning tree for document clustering and to visualise relations between similar documents in search result sets. This chapter describes the basic concept of the HarSearch Similarity Map, the idea, the theory, and finally the algorithms used for the visualisation.

7.1 The Idea Behind the Similarity Map

Many approaches have been proposed to visualise search result sets using similarities between documents, and documents and query terms (see Section 4). Studying these systems it can be noticed that all systems assist the user in:

1. Finding the most relevant documents which satisfy the user's information request.
2. Finding groups of documents which deal with the same subject.
3. Finding similar documents if a document of interest has already been found.

These are exactly the points satisfied by the HarSearch Similarity Map. An inexact query often retrieves several highly relevant documents dealing with very different subjects. Therefore, not every highly ranked document is necessarily relevant to the user. For example, searching for the term 'mouse' will very likely retrieve documents about computer mice and also about field-mice. There will be no difference in the overall relevance although the subjects are totally different. Therefore, the user may be more interested in a document with lower overall relevance but about the same subject as the one already found.

This led to the idea to visualise exactly the relation, based on document similarity, of one particular document to its most similar document in the search result set. To do so, the pairwise similarities between all documents of the search result set must

be known. A triangular similarity matrix is used to store all mutual relations. The association to a complete graph, with documents as vertices and pairwise relations between all vertices as edges, is obvious. Since only relations of high similarity are of interest a subgraph of this complete similarity graph is generally sufficient. This subgraph is the maximum similarity spanning tree that represents the HarSearch Similarity Map.

7.2 The Maximum Similarity Spanning Tree

Taking the documents as vertices and the similarities between the documents as edges, the configuration can be modelled as a graph $G = (V, E)$ where V is the search result set, and E is the set of possible interconnections between pairs of documents. For each edge $(u, v) \in E$, the weight $w(u, v)$ is specified by the similarity between document u and document v. The graph is undirected because the similarity measure is symmetric. Furthermore, every document must be connected to the document it is most similar to, and all documents of the search result set have to be connected. The problem is to find an acyclic subset $T \subseteq E$ that connects all of the vertices and whose total weight

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

is maximised. Since T is acyclic and connects all vertices, it must form a tree, which is called a *spanning tree* since it "spans" the graph G. Furthermore, the total weight $w(T)$ is maximised and the tree is therefore a ***maximum spanning tree***(MST).

This problem is equivalent to the well known ***minimum spanning tree problem***. Since, instead of the similarity as weight for the edges the reciprocal of the similarity can be used which implies a minimisation of the total weight $w(T)$ to yield the same subgraph.

The proof that all documents are connected to the most similar ones is trivial. Since the maximum spanning tree T connects all vertices and the total weight $w(T)$ has to be maximised, every vertex is connected to the tree with the edge with the maximum weight, which is the maximum similarity. If a vertex would not be connected with the maximum similarity the maximum spanning tree definition would be violated and T would not be a maximum spanning tree (see [CLR92]).

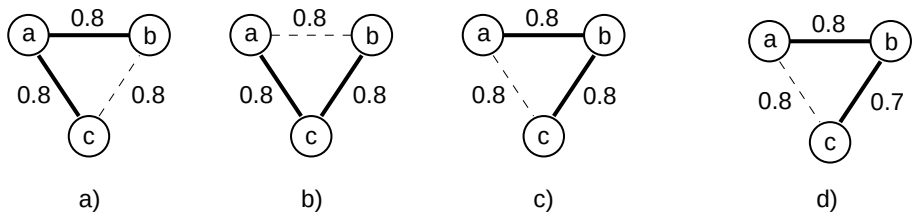


Figure 7.1: a) – c) valid maximum spanning trees. d) edge (b,c) violates the MST definition.

In case of equal similarity values the MST is not unique. Figure 7.1 illustrates the three possible maximum similarity spanning tree of a simple graph with three vertices a, b, and c. The similarity between all three vertices is obviously 0.8. Case d) is also a spanning tree but not a maximum spanning tree since the vertex c has to be connected to vertex a to form a MST.

7.2.1 Prim's Algorithm to Grow a Maximum Spanning Tree

HarSearch uses Prim's algorithm [CLR92] to generate the maximum similarity spanning tree. The algorithm uses a greedy strategy to grow the MST one edge at a time. It manages a set A that is always a subset of some MST. At each step, an edge (u,v) is determined that can be added to A without violating this invariant, in the sense that $A \cup (u, v)$ is also a subset of a MST. Such an edge is called a *safe edge* for A, since it can be safely added to A without destroying the invariant.

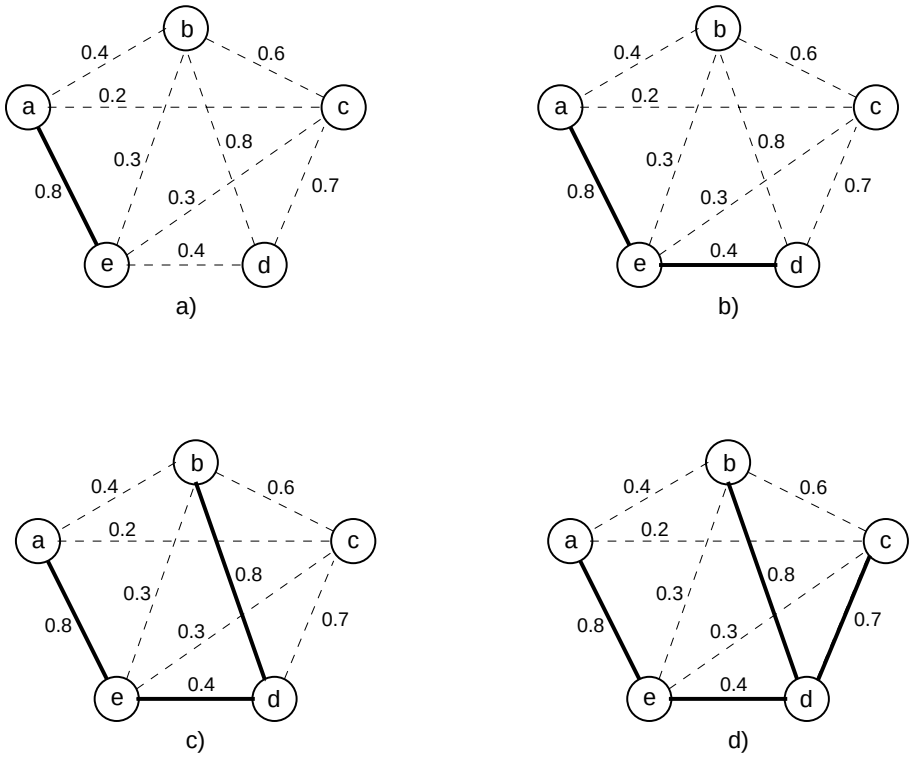


Figure 7.2: The execution of Prim's algorithm on a complete similarity graph with five vertices.

Prim's algorithm is illustrated in Figure 7.2. Starting with vertex a the algorithm adds at each step a maximum edge which connects a vertex 'outside' the MST with a vertex 'inside' the MST. In the second step the algorithm has a choice of adding either edge (a, b) or edge (e, d) since both represent the same similarity value. It has to be noticed that there are two groups of vertices which are among themselves more similar. These are the groups {a,e} and {b,c,d}. They are connected via an edge with a relatively low similarity value.

The key to implementing Prim's algorithm efficiently is to make it easy to select a new edge to be added to the tree formed by the edges in A. In the pseudocode below, the connected graph G and the root r (start vertex) of the maximum spanning tree to be grown are inputs to the algorithm. During execution of the algorithm, all nodes that are *not* in the tree reside in a priority queue Q based on a *key* field. For each vertex v, $key[v]$ is the maximum weight of any edge connecting v to a vertex in the tree; by convention, $key[v] = 0$ if there is no such edge. The field $p[v]$ names the parent of v in the tree. During the algorithm the set A is kept implicitly as

$$A = \{(v, p[v]) : v \in V - \{r\} - Q\}.$$

When the algorithm terminates, the priority queue Q is empty; the maximum spanning tree A for G is thus

$$A = \{(v, p[v]) : v \in V - \{r\}\}.$$

```
MST-Prim(G, w, r)
1  Q ← V[G]
2  for each u ∈ Q
3     do key[u] ← 0
4  key[r] ← ∞
5  p[r] ← NIL
6  while Q ≠ 0
7     do u ← EXTRACT − MAX(Q)
8        for each v ∈ Adj[u]
9           do if v ∈ Q and w(u, v) > key[v]
10             then p[v] ← u
11                  key[v] ← w(u, v)
```

Prim's algorithm works as shown in Figure 7.2. Lines 1–4 initialise the priority queue Q to contain all the vertices and set the key of each vertex to 0, except for the root r, whose key is set to ∞. Line 5 initialises $p[r]$ to NIL, since the root has no parent. Throughout the algorithm, the set $V - Q$ contains the vertices in the tree being grown. Line 7 identifies a vertex $u \in Q$ incident on a maximum edge crossing the cut $(V - Q, Q)$ (with exception of the first iteration, in which $u = r$ due to line 4). Removing u from the set Q adds it to the set $V - Q$ of vertices in the tree. Lines 8–11 update the *key* and p field of every vertex v adjacent to u but not in the tree. The updating maintains the invariant that $key[v] = w(v, p[v])$ and that $(v, p[v])$ is a maximum edge connecting v to some vertex in the tree [CLR92].

The performance of Prim's algorithm depends on the implementation of the priority queue Q. Since the similarity graph is complete each vertex in Q is adjacent to the current maximum u and must be inspected. Therefore, it is sufficient to implement the priority queue as a list.

7.2.2 Clustering Using a Maximum Similarity Spanning Tree

Maximum similarity spanning trees can be used for document clustering. Considering the complete similarity graph G and a similarity threshold s, the removal of all edges with weight $w(u,v) < s$ yields a subgraph $G(s)$ of G. For two thresholds $s_1 < s_2$ the subgraph $G(s_2)$ obviously results from a removal of some edges from subgraph $G(s_1)$. In subgraph $G(s)$ remain groups of documents which are connected by similarities greater than the similarity threshold s.

Since, the maximum similarity spanning tree is a subgraph that connects all nodes with edges of maximum weight, the removal of edges with a weight $w(u,v) < s$ exactly results in these connected groups of documents with a similarity greater s. For that reason, clusters of documents can be found by simply removing edges from the tree with increasing similarity threshold s.

For example, removing all edges with a weight $w(u,v) < 0.5$ from the similarity graph in Figure 7.2 results in two clusters of documents {a,e} and {b,c,d}. Increasing the threshold to 0.8 produces three clusters {a,e}, {b,d}, and {c}.

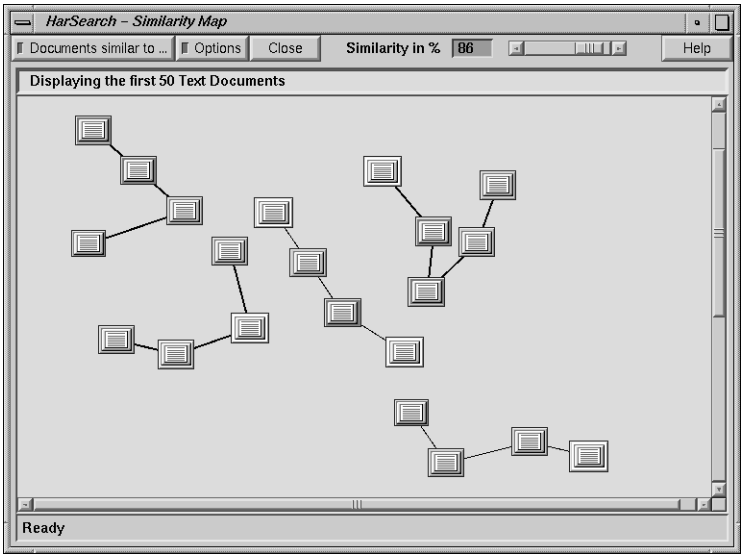


Figure 7.3: Document clusters displayed in the HarSearch Similarity Map.

Within a connected group every document is at least connected to one document to which it is more similar than the specified similarity threshold. The maximum similarity spanning tree contains no explicit information about the similarity between two documents which are not directly connected, though it is not possible that they are more similar to each other than to the documents they are connected to. For example, the information how similar the documents b and c are in the graph in Figure 7.2 is missing. The user of the Similarity Map can generally assume that they are more similar to each other then to the documents a and e since both are more similar to document d. Figure 7.3 shows clusters of documents of the Similarity Map implementation. The similarity threshold may be dynamically adjusted with the slider in the top right corner of the window.

Clustering methods using minimum spanning trees are related to single linkage methods and were already used in the year 1964 [Boc74].

7.3 Visualising the HarSearch Similarity Map

The maximum similarity spanning tree of the HarSearch Similarity Map is a so called *free tree* since it does not represent any hierarchical relationship. The edges only display the similarities between the documents no matter what spatial position the documents finally have.

To draw a free tree the algorithm has to select one vertex to start from. This vertex is called the *root* of the tree and the tree itself becomes a *rooted tree*. HarSearch takes the vertex at the *centre* of the tree as root vertex. The centre is found by counting for each vertex with a degree $d(v) > 1$ the number of vertices in the subtrees emerging from that vertex. Then the vertex is taken as root which has the minimum difference of the two maximum numbers of vertices in the subtrees. For the graph in Figure 7.2 the root is vertex d.

After finding the root HarSearch layouts the vertices on a grid, that means each vertex can only be placed on crossings of this grid. The grid is then transformed to the actual (x,y) coordinates to display the tree on the screen. Therefore, the visualisation of the tree is done in four steps:

1. Find the root of the maximum similarity spanning tree.
2. Set each vertex on a vertical position of the layout grid.
3. Optimise the horizontal positions of each vertex on the grid.
4. Transform the vertex positions to displayable (x,y) coordinates.

7.3.1 The vertical Tree Layout

The "simplest" layout that is provided by HarSearch for the Similarity Map is the *vertical tree layout*. It is the tree layout which is very common in computer science. The root (parent) is placed at the top level of the tree and the children (adjacent vertices) are positioned in the level below. The children of the children are again placed in the next level and so on. This results in a level drawing as illustrated in Figure 7.4.

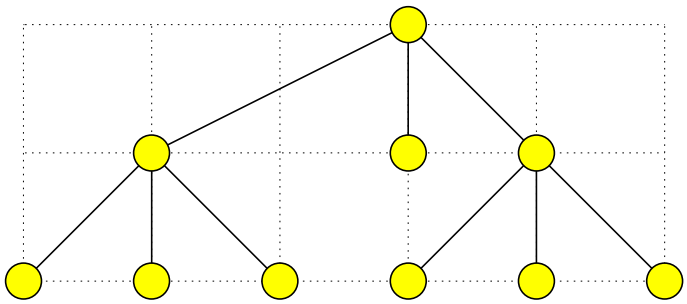


Figure 7.4: The vertical tree layout.

7.3.2 The two-sided Tree Layout

The disadvantage of the vertical tree layout is that the root is very exposed at the top of the tree. This could confuse the user of the Similarity Map, since the root document may appear more relevant then the other documents, although the only special property of the root is the centric position within the maximum similarity spanning tree.

The two-sided tree layout avoids this by splitting the tree into two parts and placing the first half of the vertices above the root and the second half below the root level. This is called the two-sided vertical tree layout. The subtrees are layouted equally to the vertical tree layout. Furthermore, the two-sided vertical tree layout can be rotated to yield the two-sided horizontal tree layout (see Figure 7.5).

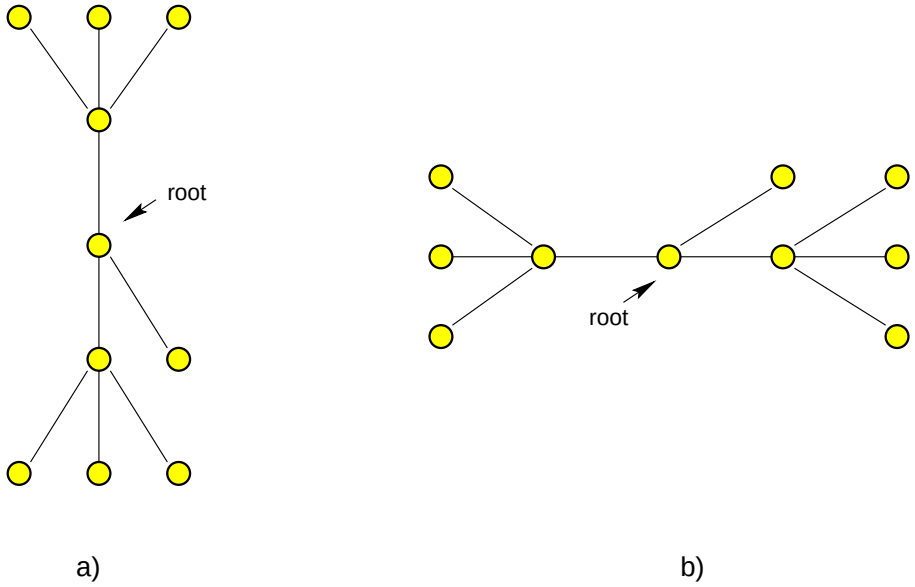


Figure 7.5: The two-sided vertical and horizontal tree layout of the tree in Figure 7.4.

7.3.3 The Polar Coordinates Layout

Another possibility to draw a free tree without exposing the root is the polar coordinates layout. This is achieved by transforming the vertical tree layout using a geometric transformation ($cartesian \Rightarrow polar$) to obtain a radial level drawing of the tree [CT94]. Figure 7.6 illustrates the principle of the polar coordinate layout and Figure 7.7 shows how it looks in HarSearch.

To achieve better results the tree layout which is later transformed into the radial representation is slightly different from the layout used for the actual tree visualisation. This is because a greater horizontal distance on the inner levels of the grid only results in a greater angular distance in the actual visualisation. Therefore it is necessary to spread the vertices in the levels near the root level, and to place the children closer to the horizontal position of their parents in the outer levels.

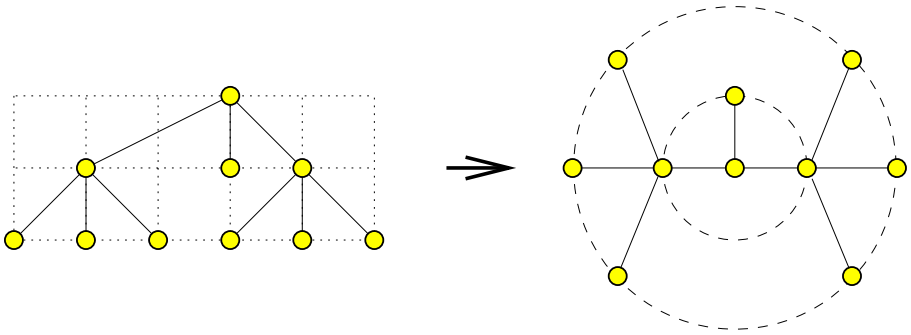


Figure 7.6: The principle of the polar coordinate layout.

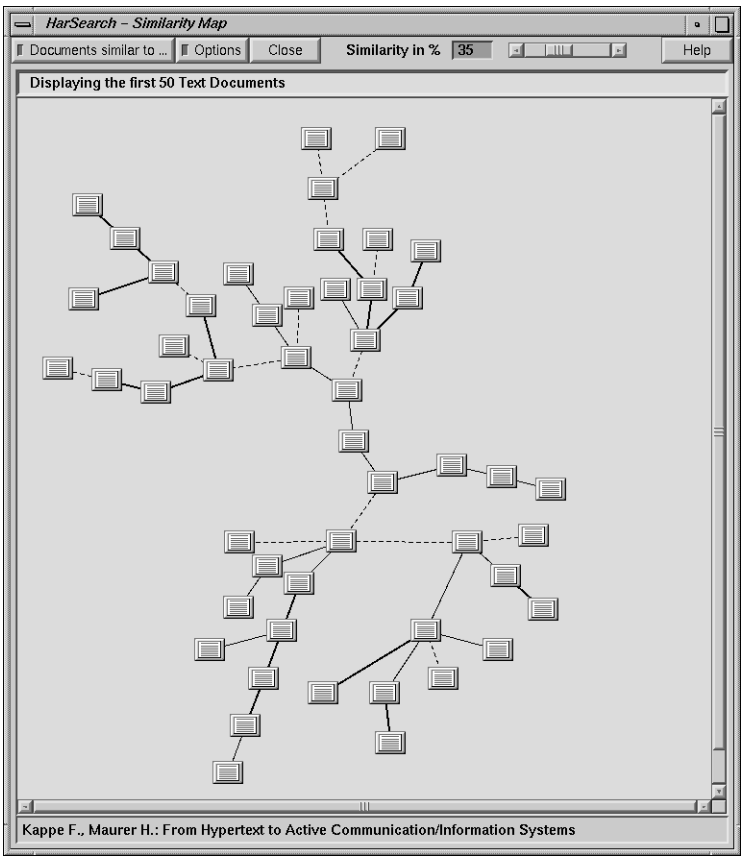


Figure 7.7: The HarSearch Similarity Map using polar coordinates layout.

7.3.4 Determining the Horizontal Positions of the Vertices

The horizontal position is of major importance for the readability of the tree layout. To attain readability some general criteria can be identified:

- No crossings of edges between the particular tree layers.
- Short edges; connected vertices should be drawn close to each other. This also involves short distances between documents of the same cluster, although the Similarity Map does not explicitely use the spatial positions of documents to display similarities.
- A balanced layout; a balanced position for every vertex between its parent and its children is desired.
- No overlap of document icons.

Methods for drawing hierarchical digraphs are used in HarSearch to achieve the criteria above. Although the maximum similarity spanning tree is a free tree, assigning a level to each vertex produces a hierarchical structure and can be considered as a special case of a hierarchical digraph.

Barycentric Method

To avoid crossings of edges between the tree levels the heuristic *Barycentric* method is used [ES90, STT81].

The idea of the Barycentric method is to choose an order of level L_1. Then for $i = 2, 3...n$, the ordering of level L_i is kept fixed while level L_{i+1} is reordered to reduce crossings between level L_{i+1} and level L_i. This is called a DOWN-phase because the algorithm goes down the levels from level L_1 to level L_n. A phase starting with a fixed order of level L_n and going up to level L_1 while reordering levels L_{i-1} is called an UP-phase. During one step the vertices in one level are reordered according to their Barycenters. The Barycenter of one vertex $v \in L_{i+1}$ simply is the average of the horizontal positions of the vertices $u \in L_i$ which are adjacent to v. If the Barycenter is calculated from the positions of the vertices in the lower (upper) level it is called lower (upper) Barycenter.

Minimising crossings between levels of a hierarchical digraph requires several UP- and DOWN-phases to calculate an approximate solution, since the minimise crossings problem is NP-complete.

To order the vertices in the tree levels only one DOWN-phase is needed. Since the children in a lower level only have one parent it is sufficient to order the children by the order of their parents. Starting with the root level this is accomplished with a single DOWN-phase. For the two-sided tree layout the vertices below are ordered by a DOWN-phase and the levels above the root level are ordered by an UP-phase.

Priority Layout Method

While the Barycentric method only reorders the tree levels to avoid crossings without assigning a precise horizontal position to each vertex, the *Priority Layout Method* was developed to compute horizontal positions of vertices that realize a readable layout of a given n-level hierarchy [STT81].

However, the method is similar to the Barycentric method. Instead of reordering the tree levels an improvement of the horizontal positions is determined according to "priority numbers" given to the vertices.

The principle to improve the position of a vertex is to minimise the difference between the present position of the vertex and the upper or lower Barycenter of the vertex in a DOWN or UP-phase. The positions of vertices are determined one by one according to their priority number. The number of adjacent vertices serves as priority number. That means that vertices with more children are positioned first. The vertex position has to satisfy several conditions:

- Only positions on the layout grid are allowed and vertices cannot be at the same position.
- The order of the vertices of each level has to be preserved.
- Only positions of vertices with priorities less than the priority of the vertex, whose position is determined, can be changed. If the position of a vertex has to be changed the distance displaced should be minimised.

In HarSearch several DOWN and UP phases are executed to optimise the tree layout. The positions of the vertices are improved in the order of levels $2, \cdots, n, n-1, \cdots, 1, t, \cdots, n$ with $t = 2, 3, \cdots, n$. In addition, to get a more balanced layout the vertices in the levels $t = 2, \cdots, n/3$ are placed according to the average of their upper and lower Barycenters. The levels of the two-sided tree layout are equally treated in reverse order. Figure 7.8 illustrates the improvement of the positions of the vertices using the priority layout method. Case a) shows the tree after the level ordering by the Barycentric method and b) shows the final tree layout.

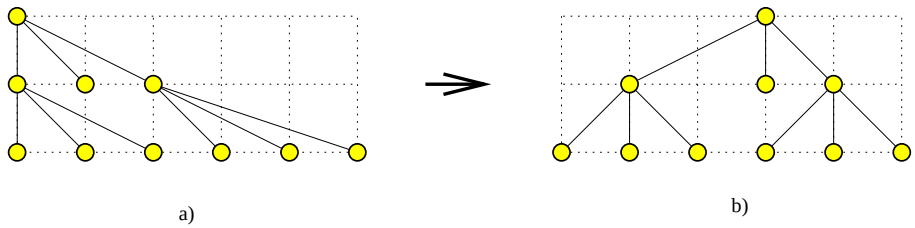


Figure 7.8: Using the Priority Layout Method to optimise the horizontal positions of the vertices.

8

The Architecture of HarSearch

HarSearch consists of several components (objects) which work together to retrieve documents from the Hyperwave server, and to present the search result set. This chapter describes the overall architecture and the interaction between particular HarSearch objects. Figure 8.1 shows an overview of the various objects with arrows representing the interactions and connections between them.

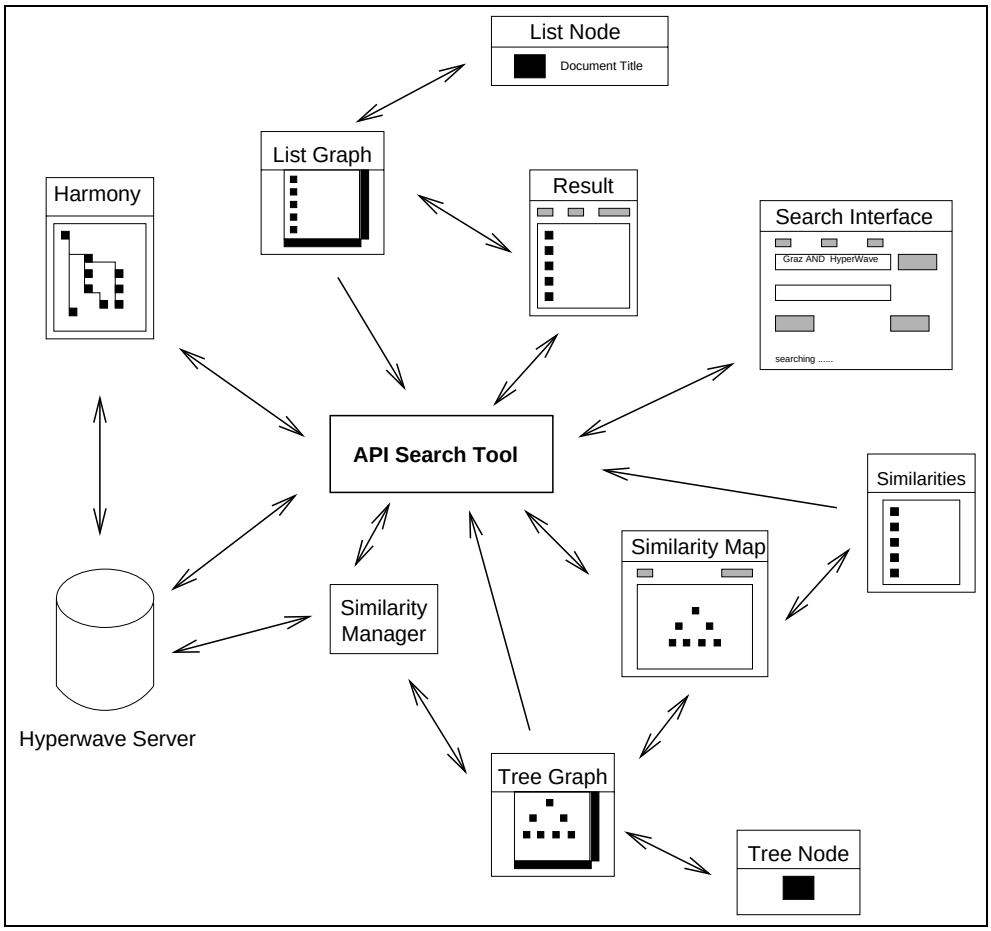


Figure 8.1: The architecture of HarSearch.

8.1 The Connection to Harmony and Hyperwave

Harmony provides an *Application Program Interface* (API) for additional tools and applications which are not directly implemented. HarSearch is such an API tool.

Harmony has a special API port to which an API tool can connect. The API tool may send and receive various messages to and from Harmony via this API connection.

Every interaction between Harmony and the API tool is accomplished over the API connection. The API tool can send a request that Harmony starts an appropriate viewer for a certain document. It can find out the currently selected collection, the active collections, the language preferences and it is notified if several events happen within Harmony (e.g. change of the current object, user identification, and so on).

After an API connection is established the API tool may also get information about the current database connection of Harmony and can use this information to build up its own database connection to the Hyperwave server. After establishing a database connection, the API tool becomes a Hyperwave client and can use all the features provided by the Hyperwave server (see Section 5.2).

8.2 The API Search Tool

The *API Search Tool* is the central object of HarSearch. It controls and synchronises the other components of HarSearch. It establishes the API connection and the database connection. The API Search Tool object communicates with Harmony and does the actual searching.

To retrieve documents the API Search Tool composes the Hyperwave queries according to the user input in the text fields and the check-boxes selections in the search interface. Hyperwave supports three different types of queries:

- The *key query*; for Boolean attribute queries to retrieve documents according to indexed document attributes.

  The formal syntax of the key query is:

```
keyquery   = "(" keyquery ")"
           | keyquery "||" keyquery     ; or
           | keyquery "&&" keyquery     ; and
           | keyquery "&!" keyquery     ; and not
           | keyfield "=" values
           | keyfield ">" simplevalue "<" simplevalue ; range

keyfield   = "Title" | "Keyword" | "TimeCreated"
           | "Author" | "Name" | "UName"
           | "UGroup" | "Host" | "Group" | ...
```

```
values     = value
           | value whitespace values  ; same as "and"-operator

value      = simplevalue     ; normal word
           | simplevalue "*" ; words with prefix simplevalue
           | simplevalue "^" ; all words >= simplevalue
```

- The *full text query*; for full text retrieval.

  The formal syntax of the full text query is:

```
expr       = term 1*(orop term)
term       = factor 1*(andop factor)
factor     = node ["{" float "}"]
node       = word 1*word   |   "(" expr ")"
andop      = "&&" [ optionlist ]
orop       = "||"
optionlist = "[" options "]"
options    = option 1*("," option)
word       = any word composed of validchars
option     = "F" | "f"
validchars = any of
   ab...xyzABC...XYZ0123456789-/
```

- The *object query*; to filter search result set by documents attributes which are not indexed.

  The formal syntax of the object query is:

```
objquery  = "(" objquery ")"
            | "!" objquery            ; not
            | objquery "||" objquery  ; or
            | objquery "&&" objquery  ; and
            | attribute operator value

attribute = "Title" | "Rights" | ...  ; any attribute name

operator  = ">"     ; less than (strings)
            | ">"   ; greater than (strings)
            | "~"   ; regular expression matching
            | "="   ; equal
```

The API Search Tool object combines the different queries for searching. In the case that the user selects full text retrieval as well as Boolean retrieval and enters additional attributes or selects a particular document type, four search tasks are executed. One for the Boolean retrieval, the second for full text retrieval and then each result set is filtered by an object query. Both results are merged together into one search result set. This search result set is then passed to the the different *interface objects* for the actual visualisation.

8.3 The Interface Objects

The **Search Interface** object represents the three different search interfaces of HarSearch (see Chapter 6). It is the main user interface of HarSearch to enter queries. The API Search Tool objects is notified if a new search task should be executed.

The **Result** object manages the presentation of the search result as a ranked list. Changes of the sort order are reported to the API Search Tool object.

The **Similarity Map** object is the interface to the HarSearch Similarity Map (see Section 6.2). After the API Search Tool object has passed the search result set to the Similarity Map object the documents to display are passed the the *Tree Graph* object. Furthermore, it generates the *Similarities* object and controls the presentation of all similarities to a selected document.

The **Similarities** object displays all similarities to a certain document in a list ordered by descending similarity values. It is very similar to the *Result* object. The API Search Tool objects is notified if a document has been selected.

8.4 Graph and Node Objects

The graph objects (**List Graph**, **Tree Graph**) do the actual visualisation of the search result set which is displayed on the screen. They layout the documents, which are passed to them by the interface objects, as list or maximum similarity spanning tree respectively.

A node object (**List Node**, **Tree Node**) represents a single document displayed in the list or the tree. They separate the appearance from the document representation from the layout structure determined by the graph objects. They inform the graph object when they have been selected. The graph objects then directly notify the API Search Tool object to synchronise these document selections among all the other visualisations. This is necessary since any particular document is often present simultaneously in the ranked list, the Similarity Map, and also in Harmony.

Since the *Tree Graph* object generates the maximum similarity spanning tree (see Chapter 7), it contacts the *Similarity Manager* to determine the similarity values between the particular documents.

8.5 The Similarity Manager

The *Similarity Manager* is the interface between the Hyperwave server and the HarSearch Similarity Map whenever similarity values are required. It retrieves the similarities from the Hyperwave server and stores them in its similarity cache to avoid that the similarity between two particular documents is requested from the Hyperwave server more than once.

The current implementation of HarSearch uses the Similarity Manager only to store and provide similarities specified in a test file. Any other document similarities are simulated using random values, since the Hyperwave server does not support document similarities at the moment.

9

Selected Details of the Implementation

HarSearch is programmed in C++. Several C++ classes have been implemented which correspond in the main to the various objects described in Chapter 8. This chapters deals with the actual implementation and is intended to be an orientation aid through the source code for anyone who maintains or extends HarSearch.

The iicmviews class library was used to build the interfaces and an API tool base class as well as a database connection class were taken as a base for HarSearch. The iicmviews class library extends the free InterViews class library which was developed at the Stanford University (see [Kü93, LCI+92]). In addition, several existing C++ classes from IICM, for example the `DLList` (Double Linked List), were used extensively. In the following Sections the most important classes and functions are explained. The notation "[...]" is used as a placeholder for lines not described in the source code.

9.1 class APISearchTool

```
class APISearchTool: public ToolAPIServer{

  public:

    [...]
    virtual void notify(Notify::NotifyEnum);
    void search();

  private:
    [...]
    ObjIdArray seenObjects_;
    DBObjectList resultol_;
}
```

The most important class of HarSearch is the class APISearchTool. It corresponds to the API Search Tool object in Chapter 8. HarSearch is notified about the actions in Harmony via the API function `notify`. The member function `search` does the query composition and the searching.

Beside pointers to the interfaces it contains an `ObjIdArray seenObjects_` which stores all documents that are already seen by the user during a HarSearch session. Furthermore, it contains the last search result in the `DBObjectList resultol_;` which is referenced by all other classes which access the search result set.

9.2 The Interface Classes

There are four interface classes implemented in HarSearch:

- `class SearchInterface`
- `class ListResultInterface`
- `class GraphResultInterface`
- `class CurrentSimilarityInterface`

All interfaces have two important member functions in common. The member function `CreateInterface` builds the particular interface and the member function `setInterfaceLanguage` is called if the language is changed in Harmony. Each interface keeps a reference to the APISearchTool.

The class `SearchInterface` is distinct from the other interfaces because it does not display documents. For that reason, the other three interfaces have in addition:

1. a pointer `graph_` to the graph object that displays the documents within the interface.
2. a function `setDBObjects([...])` which generates node objects which are then passed to the graph object to be displayed.

9.3 Classes for the Tree Visualisation

9.3.1 class APSNode

```
class APSNode{

  protected:
    long id_;   // id of the corresponding Hyperwave document
    float x_;   // x coordinate
    float y_;   // y coordinate
```

```
  private:
    int gid_;     // index in the similarity matrix
    float key_;   // for MST
    int parent_;  // for MST and pushup and placement
    int priority_; // for placement
        int layer_;   // layer of node in the graph visualisation
    int layerpos_; // horiz. position of the node
    float barycenter_; // barycenters for placement

    visAPSNodeList pnodelist_; // parent nodes
    visAPSNodeList cnodelist_; // child nodes
}
```

The APSNode is the base class for every node object in HarSearch that represents a retrieved Hyperwave object. It is only used for inheritance. The variable `id_` stores the corresponding Hyperwave object id. The members `x_` and `y_` are used to store the position where the node is finally drawn.

The other private members are declared for various purposes during the tree layout. Since the similarities between the documents are stored in a similarity matrix the node has a member `gid_` which corresponds to the document's index in the similarity matrix. The variables `key_` and `parent_` are needed for Prim's algorithm (see below and Section 7.2.1). For the optimisation of the horizontal node positions the member variable `priority_` is used in the priority layout method as well as the member variable `barycenter_`(see Section 7.3.4).

Since the nodes are positioned on a grid each node stores its grid position in the member variables `layer_` and `layerpos_` which correspond to the tree level and the horizontal position within this level.

To build a tree with the APSNodes a data structure is used which may also represent a *multilevel directed graph* (a hierarchy). Therefore each node stores its parent nodes, the nodes in levels above the node, in the *adjacency list* `pnodelist_` and its children in the adjacency list `cnodelist_`

9.3.2 class visAPSNode

```
class visAPSNode : public APSNode, public MultiButton {

  public:
    // typical InterViews functions
    virtual void allocate(Canvas*, const Allocation&, Extension&);
    virtual void reallocate();
    virtual void draw(Canvas*, const Allocation&)const;
    virtual void redraw() const;
    virtual void pick(Canvas*, const Allocation&, int , Hit&);
    virtual void drawBorder(Canvas*,const Color*,Brush*);
```

```
    virtual void damageBorder(Canvas*);
    [...]
}
```

While the APSNode is used for the layout and to build the tree the derived class `visAPSNode` (visual APSNode) provides all the functionality which is required to draw the nodes with iicmviews. It is also only used for inheritance because it provides only the functionality but no iconic document representation. For every different document type an extra class is derived from the visAPSNode. The derived classes, for example the class `APSTextNode`, use the functions of the visAPSNode and provide additional icons for the documents.

9.3.3 class APSGraph

```
class APSGraph : public InputHandler {

  public:
    [...]
    enum Sortorder {GID,LAYER,LAYERPOS,BARYCENTER,
                    PRIORITY,DEGREE,SCORE,Default = GID};
    virtual void addNode(visAPSNode*);

  protected:
    [...]
    visAPSNodeList nodelist_;
    visAPSNode *currentnode_;
}
```

Similar to the APSNode the class `APSGraph` is the base class for all graph objects in HarSearch. As base class the `APSGraph` provides the node management without any layout functionality. For every document that has to be displayed a `visAPSNode*` is added to the node list `nodelist_`. This is done using the member function `addNode` which is called from the member function `setDBObjects` of the interface classes.

In case of a document selection the selected node becomes the "current node" of the APSGraph which holds a reference to that visAPSNode in the member variable `currentnode_`. The current node requires special treatment since a border has to be drawn around it in the visualisation.

9.3.4 class APSSimilarityGraph

```
class APSSimilarityGraph : public APSGraph {

  private:
    [...]
```

```
    // functions
    void fillMatrixwithSimilarities();
    void PrimMST();
    void Matrix2Graph();
    void findRoot();
    void layerTree();
    void organizeLayerArcs();
    void orderTreeLayers();
    void optimizeNodePositions();
    void setNodesXYPosition();
    [...]
    // variables
    UGraphMatrix *graphmatrix_;
}
```

The class `APSSimilarityGraph` layouts the HarSearch Similarity Map. The similarities are stored using an instance of the `UGraphMatrix` object. The UGraph-Matrix implements a lower triangular adjacency matrix for the representation of an undirected complete graph.

After the nodes have been added to the node list the graph uses the functions above to layout the Similarity Map. Each function represents one step of the layout algorithm:

1. `fillMatrixwithSimilarities()`; at first the similarity matrix has to be filled with the document similarities. This is done by contacting the similarity manager requesting all pairwise similarity values.

2. `PrimMST()`; the similarity matrix is used to grow the maximum similarity spanning tree using Prim's algorithm. Edges of the maximum similarity spanning tree are simply marked by adding 2.0 to the similarity value. The implementation of Prim's algorithm is presented in Table 9.1.

3. `Matrix2Graph()`; the marked edges in the similarity matrix are used to generate an adjacency-list representation of the tree. Adjacent nodes are added to the adjacency-lists `pnodelist_` and `cnodelist_` of every node contained in the `nodelist_` according to the marked edges in the similarity matrix.

4. `findRoot()`; the root node within the tree is located.

5. `layerTree()`; starting with the root node a tree layer (level) is recursively assigned to each node. The root layer depends on the layout style (see Section 7.3).

6. `organizeLayerArcs()`; since the tree is represented like a hierarchy the edges are organised in a way that all edges are directed downwards. That is all edges directed from a higher level to a lower level.

7. `orderTreeLayers()`; the nodes within a layer are ordered to avoid crossings.

8. `optimizeNodePositions()`; the horizontal node positions are optimised using the priority layout method (see Section 7.3.4).

9. `setNodesXYPosition()`; finally the (x,y) coordinates are determined for each node.

9.4 class SimilarityManager

```
class SimilarityManager {

 public:
   float getSimilarity(RString& goid1,RString& goid2);
   [...]
}
```

The similarity manager caches the similarities between the different documents and contacts the Hyperwave server if a certain similarity relation is not in the cache. The implementation of the similarity manager is rather simple. Nevertheless, the function `getSimilarity` should be mentioned because it has to be extended when the document similarities are supported by the Hyperwave server.

```
void APSSimilarityGraph::PrimMST(){

visAPSNodeList q; // waiting queue
visAPSListNode *u,*v,*max;
float tmpkey;

// put all nodes in the queue
for(u = nodelist_.getFirst();u;u=nodelist_.getNext(u))
  q.addTail(new visAPSListNode(u->getNode()));

for(u = q.getFirst();u;u=q.getNext(u))
  u->getNode()->key_=0;

u = q.getFirst();
u->getNode()->parent_=u->getNode()->gid_;
max = u; // first maximum to remove

while (q.count() > 0){
  u = max;
  q.remove(max);
  graphmatrix_->increase(u->getNode()->gid_,
                         u->getNode()->parent_,2.0);

  max = q.getFirst();
  for(v=q.getFirst();v;v=q.getNext(v)){ //find new maximum
    tmpkey = graphmatrix_->get(u->getNode()->gid_,
                               v->getNode()->gid_);
    if(tmpkey > v->getNode()->key_){
      v->getNode()->key_ = tmpkey;
      v->getNode()->parent_= u->getNode()->gid_;
    }
    if(v->getNode()->key_ > max->getNode()->key_)
      max = v;
  }
}
```

Table 9.1: The implementation of Prim's algorithm: Prim's algorithm is explained in Section 7.2.1. The visAPSNodeList is a derived from the class DLList. Therefore, it does not contain pointers to visAPSNodes but pointers of the type `visAPSListNode*`. This is because the DLList only manages nodes of a type derived from the DLListNode. For that reason, the visAPSListNode is a simple class, derived from the DLListNode, which only stores a pointer to a visAPSNode. Every visAPSListNode has a member function `getNode()` to access the stored visAPSNode*. This explains the frequently usage of the member function `getNode()` since the tree representation is based on DLLists.

10

Outlook and Future Work

The visualisation of document similarities introduced by HarSearch seems to be very intuitive and can be generated within a reasonable time for user interactions. The major problem of HarSearch is that it was never tested with real similarity values, except for small search result sets, due to delays in the implementation of document similarities in the Hyperwave server. The only possibility was to test it with hand-made test files and for larger search result sets with random similarities. Therefore, the most important step will be to extend the partly implemented similarity manager to request the document similarities from the Hyperwave server when they are available. This will hopefully be the case in the near future. Meanwhile random similarities can be generated by the *Random Similarity Generator* which produces a user defined number of random document clusters. The number of clusters can be set in the options dialog of the Similarity Map.

Since HarSearch is a client and the document similarities have to be computed via the server's full text index, another aspect will be to determine how much load is put on the server and the network since the number of similarities is $O(n^2)$ for n retrieved documents. This might influence the interaction time considerably for larger search result sets.

When the real similarities are available usability evaluations will be necessary to find out how efficient and helpful the Similarity Map really is to locate the most relevant documents for the user within large search result sets.

An interesting issue would also be to explore how the polar coordinate layout can be improved. The current layout highly depends on the tree structure being displayed. Especially for large search result sets and dense trees it is problematic to achieve a non-overlapping layout, particularly for the inner tree levels.

Since the development of Hyperwave clients is nowadays more directed to Java applications and to the integration into standard Web browsers an implementation of the Similarity Map in Java is imaginable if usability evaluations indicate high user acceptance.

The implementation of *relevance feedback* would probably be a tremendous improvement in the retrieval of most relevant documents. This means that it would not

only be possible to show document relations between the documents in the search result set but also to find documents, similar to one or several retrieved documents, which are not in the search result set. However, this is more a matter of the server implementation than of HarSearch.

11

Concluding Remarks

HarSearch provides a flexible search interface which extends the search capabilities of Harmony and utilises the Hyperwave document retrieval facilities.

With the HarSearch similarity map an innovative approach to the visualisation of similarities between documents in search result sets has been introduced. The clearly arranged layout of the maximum similarity spanning tree stresses the connection of two documents by their similarity. In addition, it suggests a path for exploration through the search result set.

Nevertheless, there are still two open questions; first, the consequences that result from the usage of real similarity values for large search result sets, and second the usability of the similarity map.

It seems reasonable to assume that there should not be a major difference in the layout and the behaviour of the similarity map if real similarity values are used. However, the actual similarity structure of large search result sets and the implications to the visualisation remain unclear.

Usability evaluations will show if the abstract tree layout does not lead to misinterpretation of the spatial position of the documents since the similarity is only expressed by the connecting edges and not by the spatial distance of the document icons.

Appendix A

HarSearch User Guide

The user guide should help users to enter queries and to search for documents with HarSearch. First the elements common to all three query interfaces are explained and then each interface is described in detail. The result window and the Similarity Map are explained afterwards. At the end, a description of the XDefaults is included.

Starting HarSearch: the tool can be started from the command line with:

```
harsearch -apihost <host> -apiport <port> [-file <filename>]
```

where `<host>` is the computer system on which Harmony is running and `<port>` is the API port of Harmony. Harmony can be started with the parameter "-api" to find out its API port or Harmony displays the API port if the appropriate variable is set in the XDefaults (see Section A.4). Since the Hyperwave server does not support document similarities at the moment, HarSearch may be started with an optional parameter `-file` followed by the name of the test file. The format of the test file is described in Appendix B.

Moreover, it is possible to integrate HarSearch into Harmony and to start it right from the Harmony Tools menu. This can also be accomplished by editing the XDefaults for Harmony (see Section A.4).

A.1 The HarSearch Search Interfaces

HarSearch provides three different search interfaces for simple, extended and power queries. They are described in the following sections.

A.1.1 Common Elements

Menu: at the top of each interface there is a pull-down menu which is made up of four sub-menus:

- The *Program* menu contains only one item to **exit** HarSearch.

- The *QueryType* menu allows to choose between the three search interfaces to enter a **simple** query, an **extended** query, or a **power** query. The interface type changes immediately according to the query type. The query type is also displayed in the window title.

- Several options are set via the *Options* sub-menu. The first item is used to select the **search languages**. The selected search languages are then used to compose the queries to search for titles and document words in the different languages. The menu item *Similarity Map* may be used to open and close the HarSearch Similarity Map (see Section A.3). The other items of the Options sub-menu are only selectable in the extended search interface. If **Time Input** is chosen dates may be entered with an exact time (e.g. 95/12/31 12:03:56). Otherwise it is only possible to enter the day (e.g. 95/12/31). The **Documents Attributes** item opens another sub-menu to select the attributes which are part of the extended query interface (see Figure A.3).

- The *Help* sub-menu provides help and some information about HarSearch.

Display Section: the display section allows to specify how many found documents are presented at the same time by clicking on the appropriate radio button. The users may select 25, 50, and 100 or they can enter an arbitrary number in the field on the right side of the display section. To enter the value the radio button beside the input field must be clicked first.

Buttons: two buttons are part of every search interface. One is the *Search* button. Pressing this button initiates a search. The *Clear* button clears the interface which means that all entries in the different input fields are deleted.

Query History: the query history is an important tool to rebuild and adapt queries. Every issued query is represented by a line in the query history scroll box. The query history shows the query number, the found documents with the particular query, the query type, and a short description of the query (see Figure A.2).

The query description is made up of the entry in the main query field (i.e. the first input field from the top) and the entries in the different attribute fields. The attributes are indicated by a prefix and separated by a semi colon. The prefixes are:

- A: for author.
- TC: for the creation time.
- TM: for the modification time.
- N: for name.
- DA: for the DocAuthor.
- DD: for the DocDate.

- UA: for UserAtts.

In addition, dates are displayed in combination with the symbols >, <, and − to show the time interval *after*, *before*, and *between*.

If the query history scroll box has the focus the user can browse through the queries using the cursor keys or the mouse to scroll the list up and down. The selection of one query resets the search interface to the state of the particular query. All check-boxes and the input fields are set. The users may change the selection of the check-boxes or the entries in the input fields and then they can search again.

The query history can be opened and closed dynamically by clicking on the *Query History* check box. The query entries are inserted in any case.

Progress Indicator: the progress indicator shows the progress during a search task or any other task which takes a longer time. If the tasks take too long they can be interrupted by pressing the *stop button*. The progress indicator also shows the current state of HarSearch which is usually the ready state (see Figure A.2).

User Display: the user is displayed at the bottom of every search interface. Since the HarSearch has its own database connection the user identified in Harmony is not necessarily the user as who the HarSearch is identified. This could be the case if HarSearch runs on a different computer and has not been started directly by Harmony . If the user displayed by HarSearch is not the same as the user identified in Harmony another identification procedure in Harmony solves this problem.

Short Cuts: to accelerate the user interaction several keyboard short cuts are supported. In general these short cuts are displayed beside the menu items or are indicated by the labels of the check boxes. A particular check box may be toggled using the *Meta* key in combination with the underlined letter of the text label (e.g. to toggle the local server selection type *M-l*). To exit HarSearch *Ctrl-x* may be used. For help press *F1*.

A.1.2 The Simple Query Interface

The simple query interface is made for simple queries and users unfamiliar with Harmony and Hyperwave. Figure A.1 shows the simple query interface during a search for the search term "Hyperwave" in the local server. The search term has to be contained in the titles or the content of the documents.

The check boxes at the top of the simple query interface are used to select where the system should search for documents. The text labels are self-explanatory. The *local server* is the local Hyperwave server and the *selected collection* is the currently selected collection in Harmony.

Under these check-boxes is the query field. The user may enter single search terms separated by a blank or a more complex query with logical operators and parenthesis. (e.g. "Graz AND Uni* AND (informatics OR electronics)") Valid

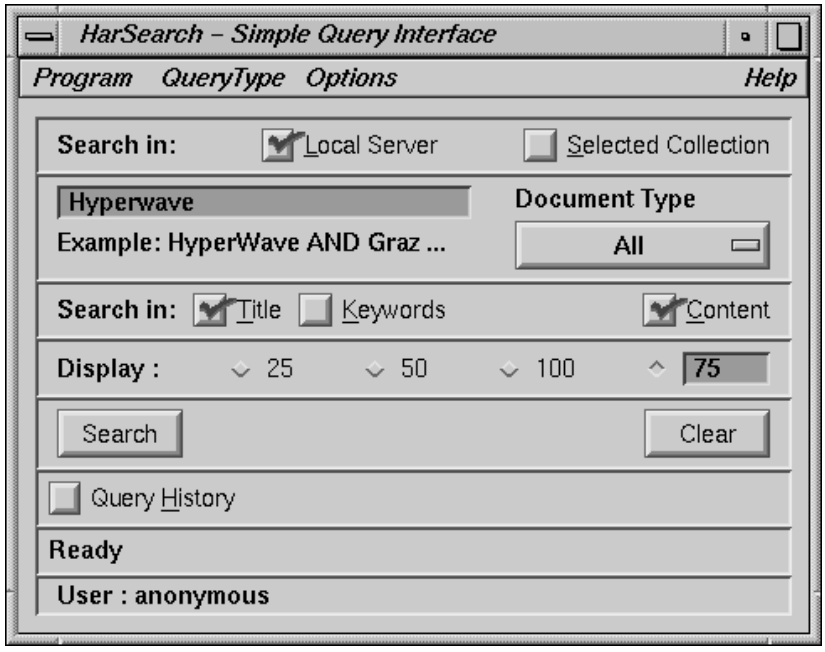


Figure A.1: The simple query interface.

logical operators are AND, OR, and ANDNOT. Instead of the operators the symbols &&, || , and &! may be used. The query is case insensitive. The operator "*" may be used to represent any suffix of a query term.

On the right side of the query field the users can select a specific document type if they only want to find text documents, images, movies and so on. If *All* is selected, every document which satisfies the query is presented. The check-boxes under the query field define whether the system searches in the titles, the keywords, or the contents of the documents. Documents whose titles or keywords match the query are ranked with a 100% relevance score in the result window. Lower relevance score results from a search in the content. The other elements of the simple query interface are described in Section A.1.1.

A.1.3 The Extended Query Interface

The extended query interface extends the simple query interface and the Harmony search interface and can be used for a more specific search. It allows to search for various document attributes.

The top section is similar to the simple query interface. The only difference is the *Active Collections* check box which causes the system to search for documents in the active collections of Harmony (see Figure A.2).

The real extension is the *Document Attributes* section of the extended query interface. Several Hyperwave document attributes may be used for searching. These

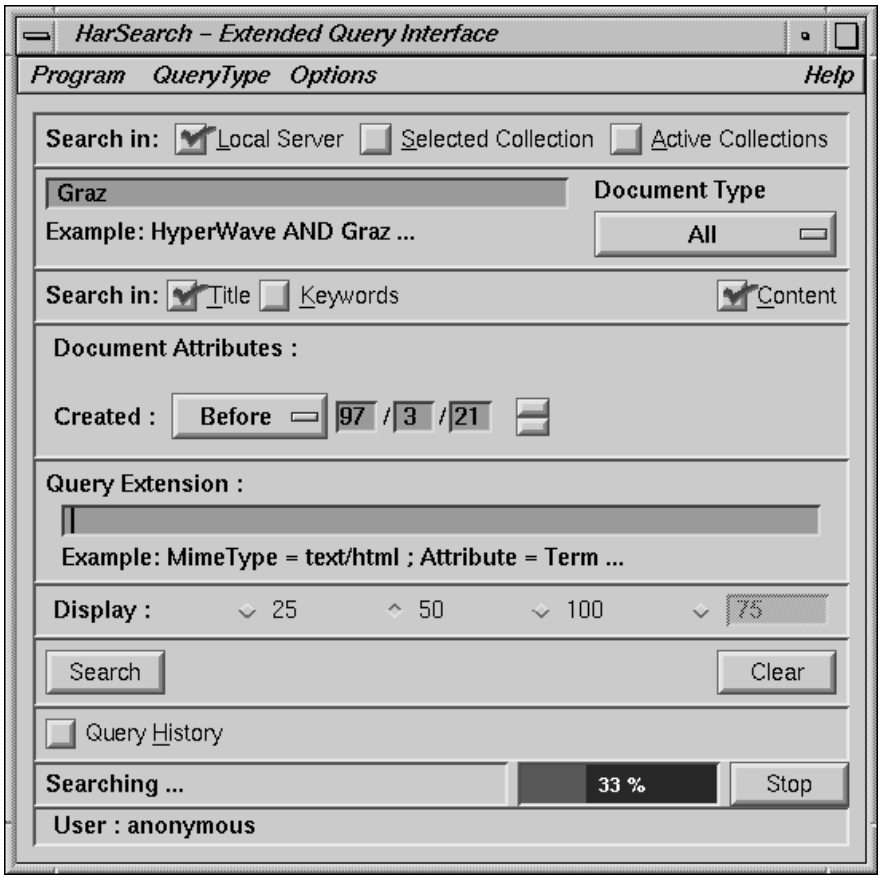


Figure A.2: The extended query interface.

attributes are:

- Author (indexed)
- TimeCreated (indexed)
- TimeModified (indexed)
- Name (indexed)
- DocAuthor
- DocDate
- UserAtts

If the attributes are indexed they can be used directly to retrieve documents. This means that it is not necessary to enter a query if, for example, an author is given. The other attributes can only be used to filter a search result set by a particular attribute. The attribute field may also contain a more complex specification with logical operators like in the query field. To use a particular attribute it has to

be selected in the *Document Attributes* sub menu in the *Options* menu. Selected attributes are marked with a tick (see Figure A.3).

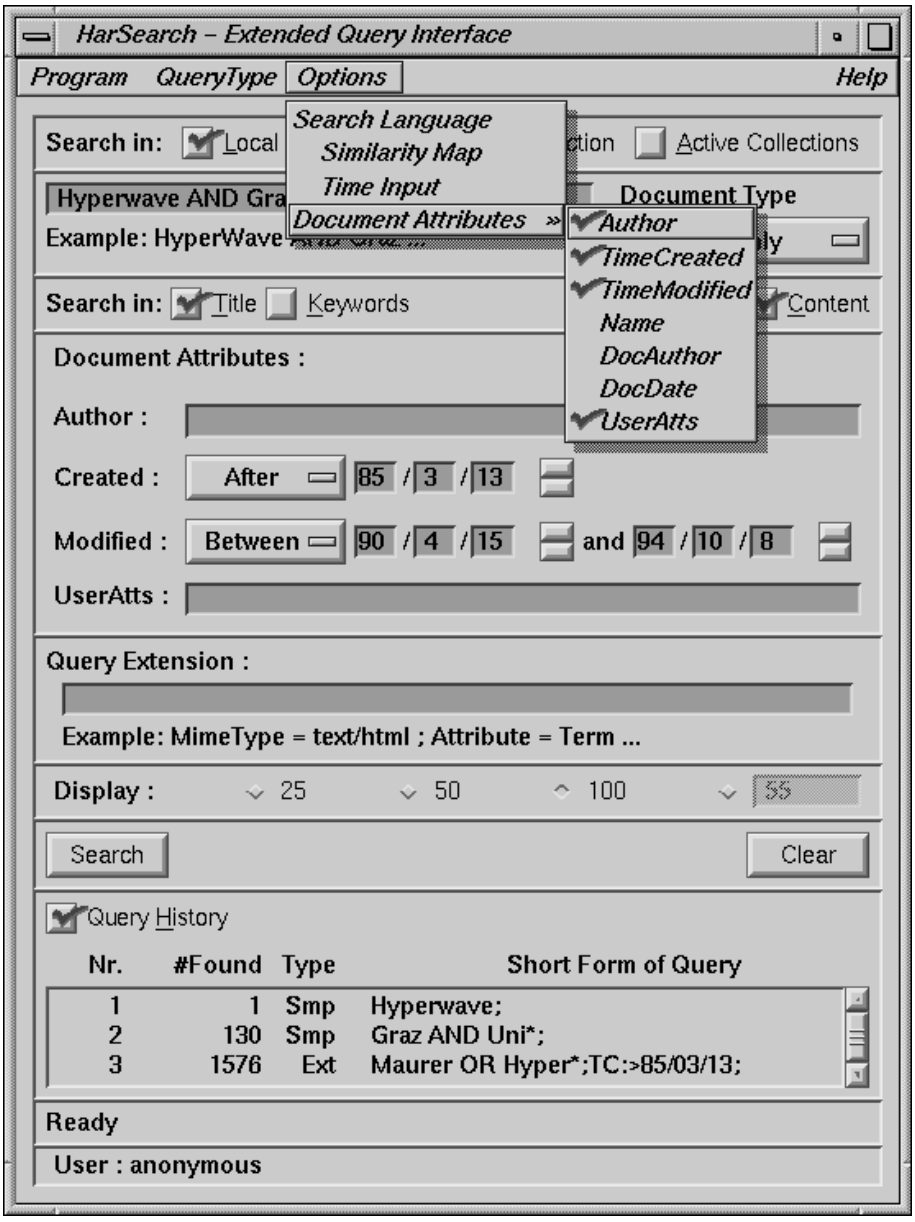


Figure A.3: Several document attributes may be selected in the attribute menu of the extended query interface.

Dates are entered using date editors. The time interval can be selected via the "selection button". The time interval may be *after* a certain time up to now, *before* a certain time, and *between* two specified dates. If an accurate time input is required the *Time Input* option in the *Options* menu must be selected. The date and the time may be entered directly in the appropriate field or may be adjusted with the two small buttons beside the date fields.

The Query Extension may be used to filter the search result by additional document attributes which are not supported by the *Document Attributes* section. Any attribute may be entered and complex specifications are possible. Several attributes must be separated by a semicolon. (e.g. `attribute1= term1 AND term1 ; attribute2= term3 OR term4 AND term5 ...`)

The other elements of the extended query interface are common to all query interfaces and described in A.1.1.

A.1.4 The Power Query Interface

The main parts of the power query interface are two input fields. The field at the top is the query field. The user may enter either a key query or a full text query in this field. The query syntax is described in Section 8.2. The type of the query must be specified via the radio buttons under the query field. The second field may be used to input a plain Hyperwave object query.

The check boxes at the top are again to specify where the system has to search for documents (see A.1.2). Figure A.4 shows a screen shot of the power query interface. The other elements are described in Section A.1.1.

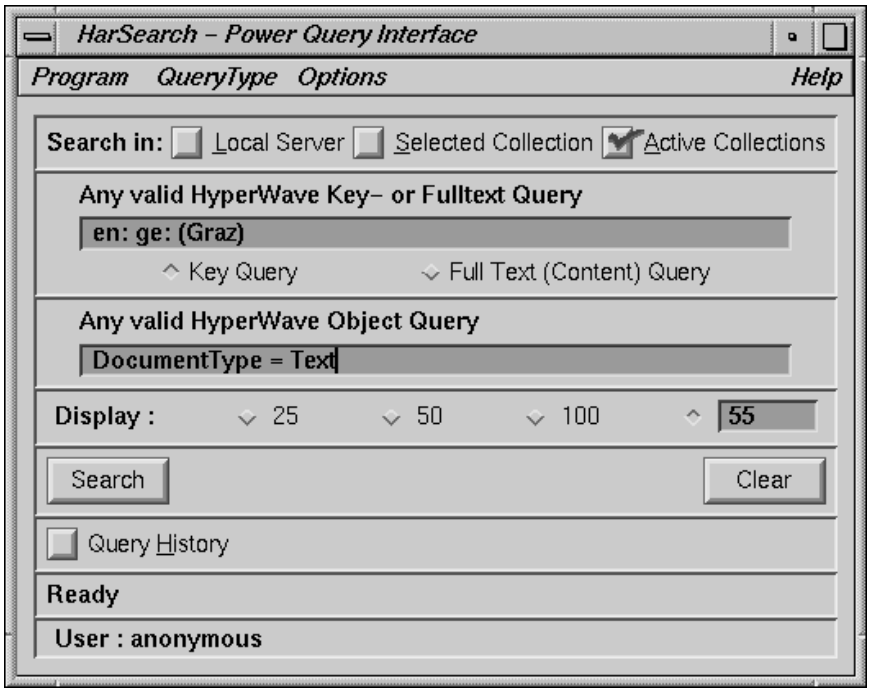


Figure A.4: The power query interface.

A.2 The Result Window

Every search result is presented in the result window. The number of documents which are displayed at the same time is specified in the *Display* section of the query interfaces and changes dynamically. The document icons are placed in an area which can be scrolled via the horizontal and vertical scrollbars (see Figure A.5).

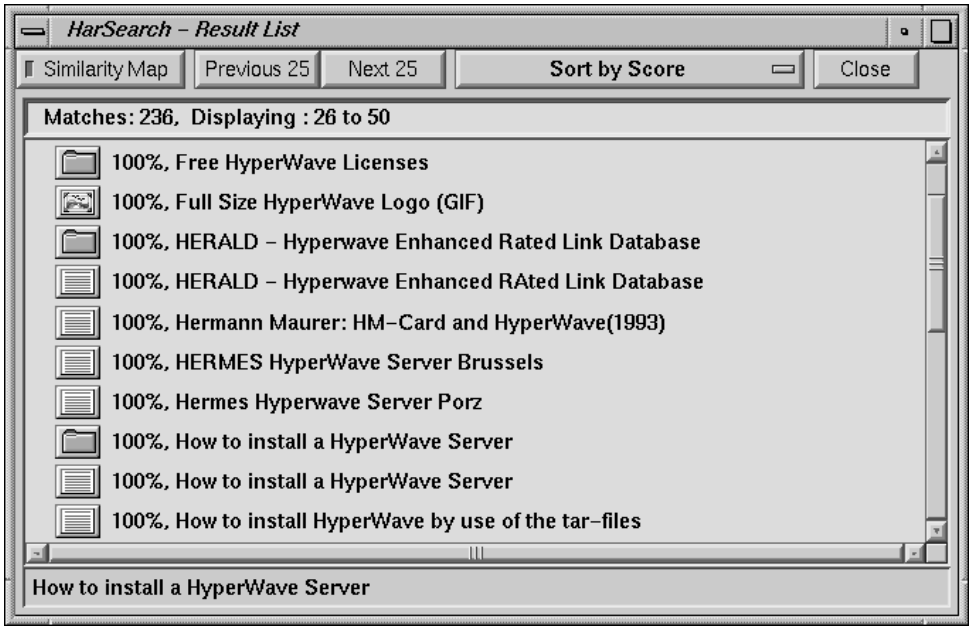


Figure A.5: The result window displaying the result of the search for the word hyperwave.

If the number of found documents is greater than the specified number in the interface the user can view all documents using the *Next* and *Previous* button. At the beginning only the first documents are presented. The sort order can be defined via the *sort order selection button* in the right corner of the result window. The possible sort orders appear after a mouse click with the left mouse button on the selection button. If the user moves the mouse pointer on a particular document the title of that document is displayed at the bottom of the result window.

A single-click selects a document and a frame is drawn around this document. It is also shown in Harmony and the other visualisations of HarSearch. This mechanism is called *location feedback*. A single-click with the right mouse button causes Harmony to display the document attributes. To view a particular document the user has to double-click on that document. This causes Harmony to show the document with an appropriate viewer. The document icons show the document type. Already viewed documents are marked with a tick. Furthermore, the user may open the HarSearch Similarity Map by pressing the *Similarity Map* button. Pressing the *Close* button closes the result window.

A.3 The HarSearch Similarity Map

The *HarSearch Similarity Map* visualises the similarity relations between retrieved documents in form of a tree. It can be opened via the options menu of the search interface or by clicking the *Similarity Map* button in the *Result Window*.

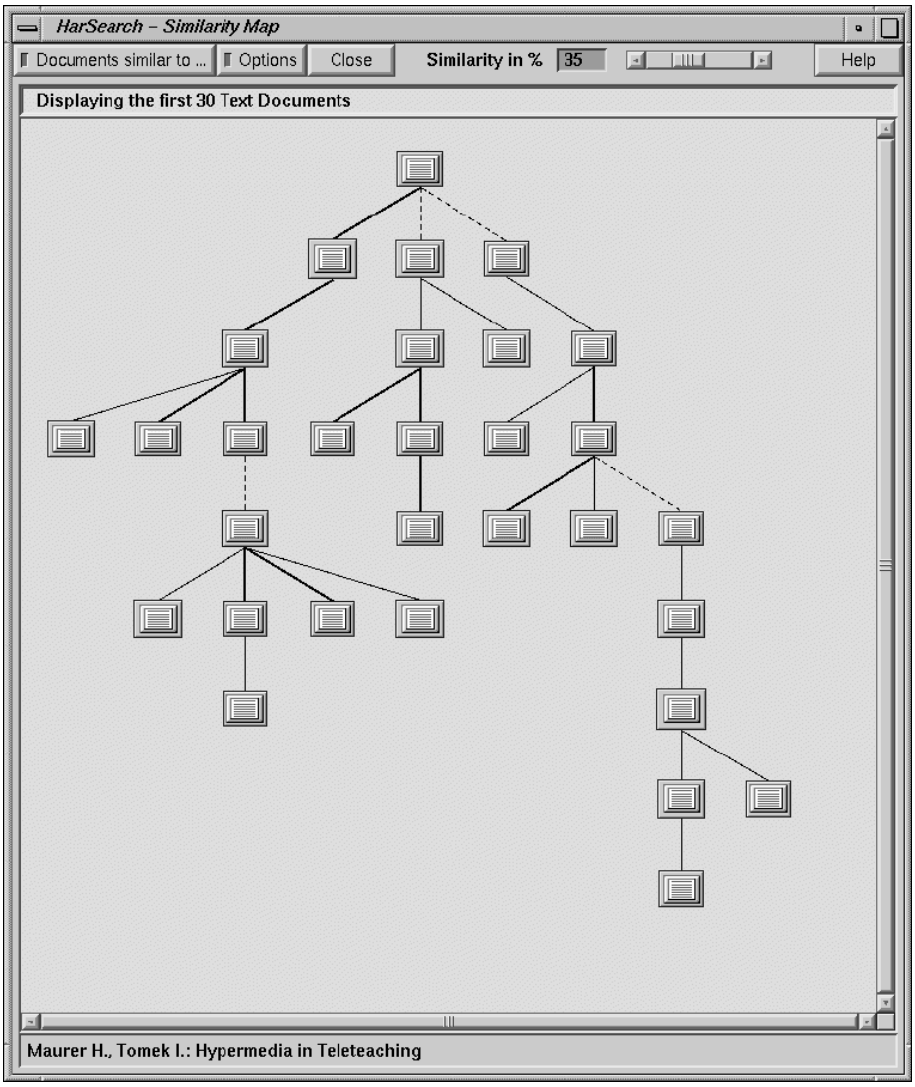


Figure A.6: One possible visualisation of 30 documents in the HarSearch Similarity Map.

In the Similarity Map each document is represented by an icon and connected with a line to the document it is most similar to. The line is called an *edge*. Figure A.6 shows an example how the visualisation of 30 text documents may look like in the Similarity Map.

Thick edges represent high similarity, thin edges medium similarity and dotted edges show a lower similarity relation between two particular documents. The overall relevance is indicated by the icon size of the documents. Bigger icons display higher

relevance. As in the result window the documents are placed in a scrollable area. The user may scroll the visible area by using the scroll bars at the bottom and on the right side of the scrollable area.

Moving the mouse over a document immediately shows the title of the document at the bottom of the Similarity Map window. A single-click with the left mouse button selects the particular document and a double-click opens an appropriate viewer. As in the result window a single-click with the right mouse buttons shows the document attributes. As the result window the Similarity Map provides *location feedback*.

Below the button row the *status line* is situated showing how many documents are in the Similarity Map. The Similarity Map only displays the documents of a search result up to a user defined number. It does not show all documents as in the result window. Therefore, it may happen that the Similarity Map cannot show a document which has been selected in the result list. If no number is specified the Similarity Map shows as many documents as selected in the search interface. The Similarity Map displays only text documents since the similarity can only be determined between text documents.

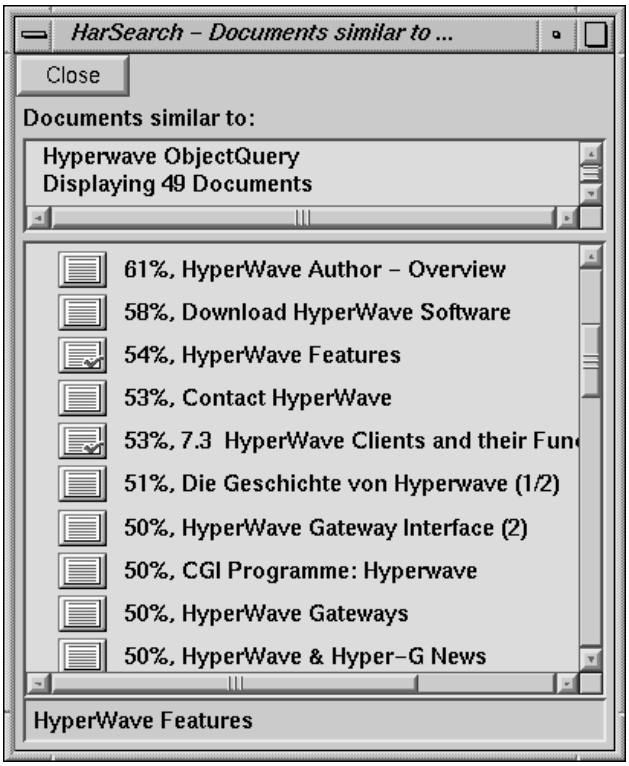


Figure A.7: A list of all similarities to a previously selected document.

The *similarity slider* in the right top corner allows to set a similarity threshold. Only similarities greater or equal this similarity threshold are presented. This can be used to find cluster of similar documents by simply increasing the similarity threshold. The Similarity Map removes the edges dynamically. Furthermore, the user

may define that documents which are no longer connected to any other document are removed.

The *Documents similar to ...* button is only enabled if a document is selected. If it is pressed a window is opened which presents the exact similarity values for all other documents in the Similarity Map to the currently selected document (see Figure A.7). The documents are presented in a list showing the similarity in percent and the document title. The list is ordered by descending similarity values.

The *location feedback* mechanism is also provided by the similarity window. While the window is open it shows the similarities to the document which was selected when then button was pressed. This allows the user to select documents in that window and to see where these documents are located in the Similarity Map. To get the similarities to another document the window must be closed and opened again.

The HarSearch Similarity Map offers various layout styles and has numerous options which can be set in the options dialog (see Figure A.8). To open the dialog an *Options* button can be found in the button row of the Similarity Map.

In the middle section of the dialog the user may specify the style of the document icons and of the connecting edges. In each case four different styles are available. The icon styles may be:

- Simple; documents are presented by small icons as the icons in the result window but without title.
- Scaled; a frame is drawn around the simple document icons according to the overall relevance. Therefore higher overall relevance results in larger icons.
- Coloured; instead of a scaled frame a coloured frame may be used to show overall relevance. In this case the frames around the document icons have equal size.
- Scaled&Coloured; this icon style is the combination of scaled and coloured icons.

Equally the edge style can be specified as:

- Simple; to use only thin lines of one colour to represent similarity relations. The colour of these "simple" lines may be set by XDefaults (see Section A.4).
- Width Coded; as in the example in Figure A.6 the edges are drawn thick, thin, and dotted if the width coded edge style is selected. The used colour is the one of the simple lines.
- Colour Coded; also the edge colour may be used to present different levels of similarity.
- Width&Colour Coded; to combine the last two styles.

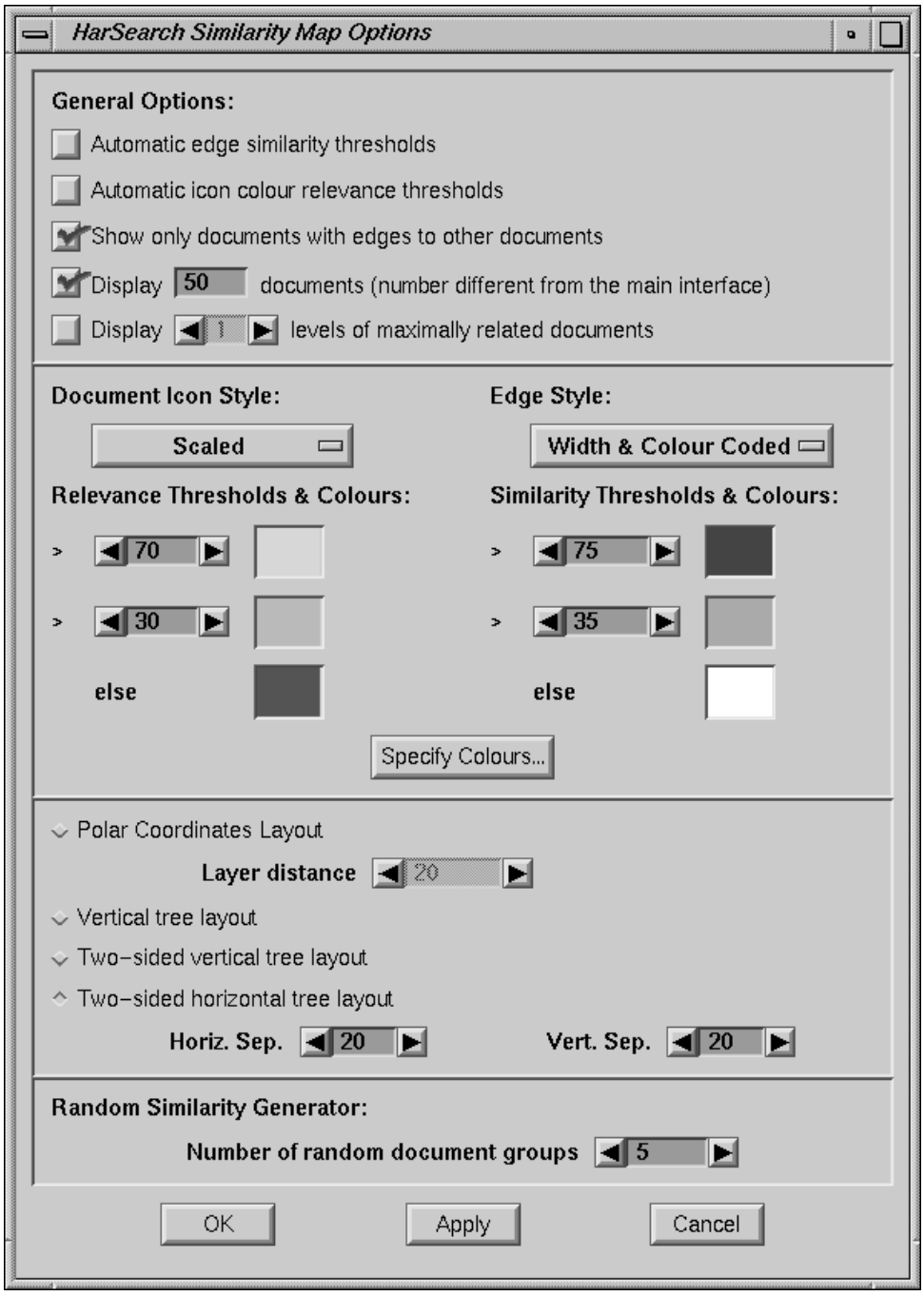


Figure A.8: The options dialog for the HarSearch Similarity Map.

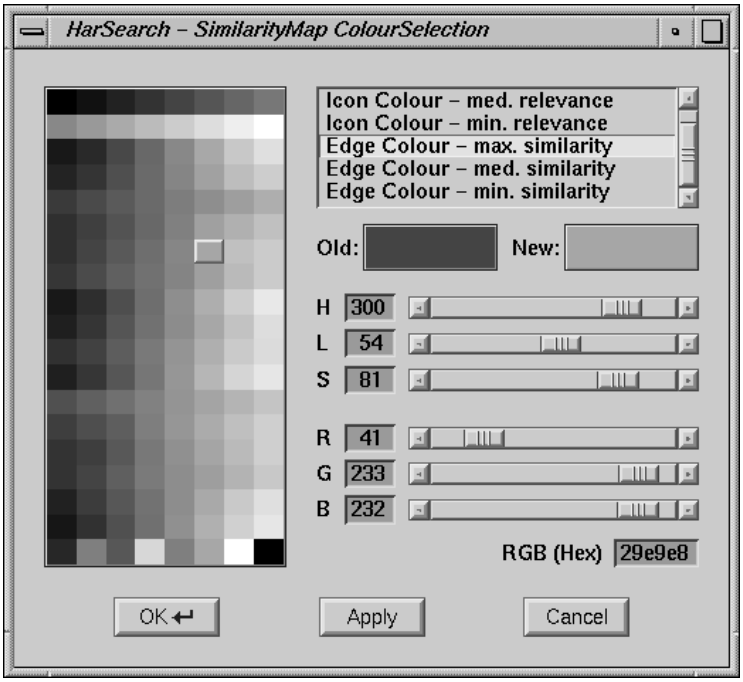
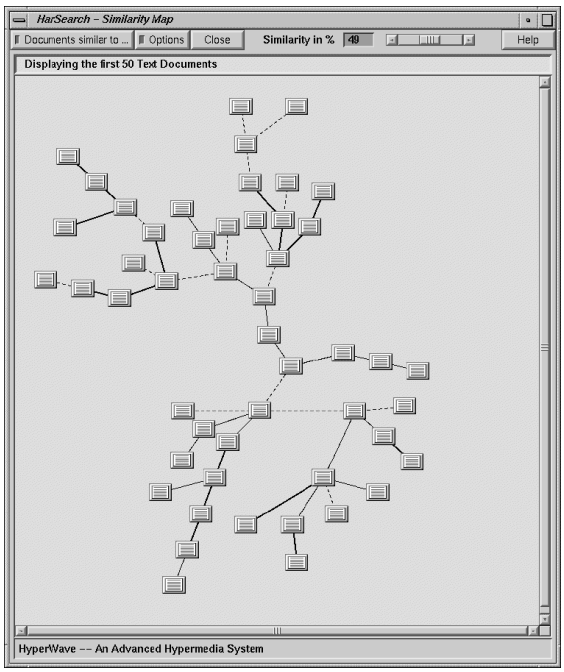


Figure A.9: Similarity Map colour specification.

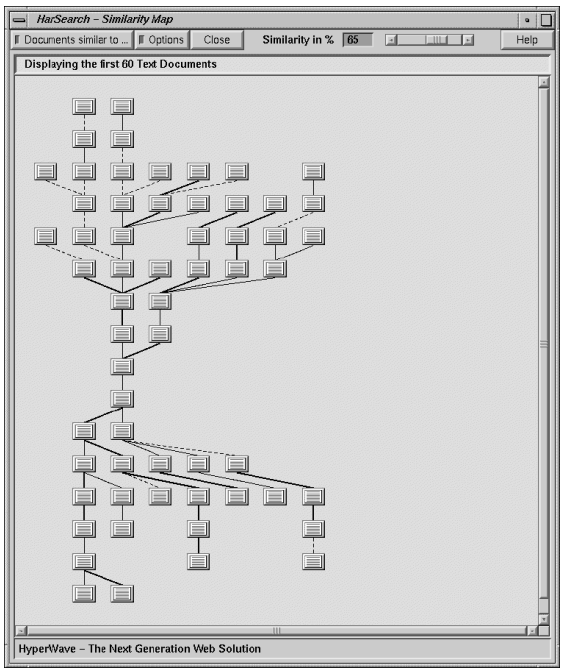
The thresholds which are set below the selection buttons for the icon and edge styles allow the user to specify ranges when to use a specified colour or edge style. For example, if the similarity thresholds for the edges are set to 70% and 35% then edges which represent a similarity >70% are drawn as thick lines, edges which represent a similarity between 70% and 36% are drawn as thin lines, and the edges below 36% are drawn as dotted lines. The colours for the icons and edges are used accordingly. If the user does not want to set the thresholds it is possible to use automatic thresholds. The automatic thresholds are used if the appropriate check-boxes in the *General Options* sections of the dialog are selected. HarSearch tries to find an automatic threshold so that always a third of the edges or icons is displayed in one colour. The user can specify the colours in the separate colour dialog (see Figure A.9).

The other check-boxes in the *General Options* section may be selected to show only documents with edges to other documents, to display a different number of documents in the Similarity Map as specified in the search interface, and to show levels of maximally related documents around the currently selected document.

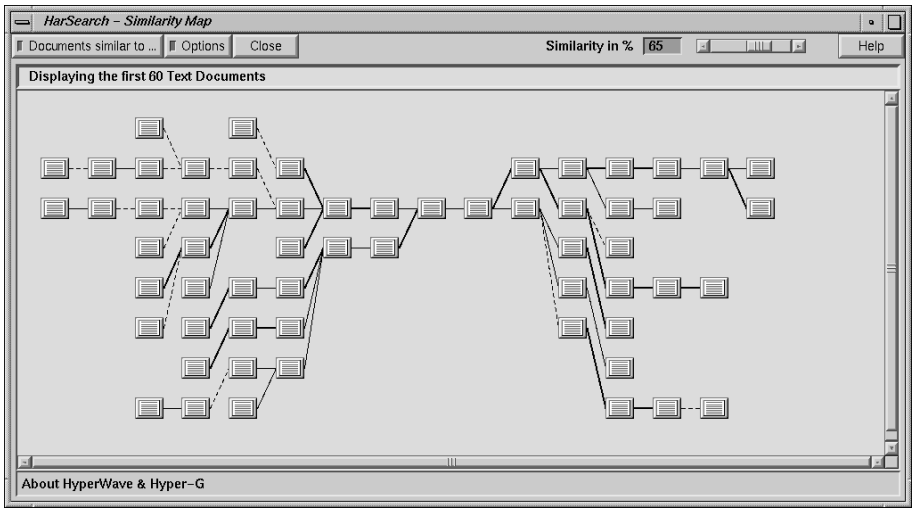
The layout of the similarity tree can be changed in the bottom section of the options dialog. The user can choose between a radial (polar coordinate) tree layout, a vertical tree layout, a two-sided vertical tree layout, and a two-sided horizontal tree layout. Furthermore, it is possible to adjust the radial, vertical, and horizontal separations between the tree levels. The vertical tree layout the layout used in Figure A.6. Examples for the other three layouts are shown in Figure A.10.



(a) Polar coordinate layout.



(b) Two-sided vertical tree layout.



(c) Two-sided horizontal tree layout.

Figure A.10: Different Similarity Map layout styles.

A.4 XDefaults

Since HarSearch has been developed for the X Window System several options may be set via XDefaults.

The general colour of the background and of flat objects can be set with:

```
HarSearch*background:      #CCCCCC
HarSearch*flat:            #CCCCCC
```

The flat objects are objects of the interviews class library which has been used to implement the interface (see Chapter 9).

The fonts used in the interface in general and for the button labels are also set via XDefaults:

```
HarSearch.font:              FONT-SPEC
HarSearch.Button.font:       FONT-SPEC
```

The standard X Windows font specification has to be used. For example, a medium, Helvetica font is defined as follows:

```
-*-helvetica-medium-r-*-*-*-140-*-*-*-*-iso8859-*
```

To keep it simple and avoid long lines the string for the font specification is replaced by `FONT-SPEC`.

To avoid flickering the double-buffering has to be switched on:

```
HarSearch*double_buffered:   on
```

The search interface starts with one of the three possible interfaces. The initial interface is set by:

```
HarSearch.querytype:         Simple
```

The option `Simple` can be replaced by `Extended` or `Power` which means that the extended query interface or the power query interface should be used instead of the simple query interface.

The states of the different check-boxes are set by the following XDefaults. The value `on` means *open* in the case of the query history:

```
HarSearch.searchinlocal:     on
HarSearch.searchinselected:  off
HarSearch.searchinactive:    off
HarSearch.searchtitle:       on
HarSearch.searchkeywords:    on
HarSearch.searchcontent:     off
HarSearch.queryhistory:      on
```

The attribute fields in the extended query interface may be selected via pull-down menu or set via XDefaults. A particular attribute field is part of the extended query interface if the appropriate default value is set to `on`:

```
HarSearch.author:            on
HarSearch.timecreated:       on
HarSearch.timemodified:      off
HarSearch.docname:           off
HarSearch.docauthor:         off
HarSearch.docdate:           off
HarSearch.useratts:          off
```

To set the appearance of the progress indicator the next default values are used:

```
HarSearch*Progress.background:            blue
HarSearch*Progress.foreground:            red

HarSearch*ProgressLabel.foreground:       white
HarSearch*ProgressLabel.font:             FONT-SPEC
```

The size of scrollbars and sliders can be set with:

```
HarSearch*sliderSize:                     15
HarSearch*moverSize:                      15
```

The colour and the size of the scrollable area of the result window may be specified with the following XDefaults:

```
HarSearch*ResultList.background:          #033b68
HarSearch*ResultList.foreground:          cyan
HarSearch*ResultList.ScrollAreaWidth:     500
HarSearch*ResultList.ScrollAreaHeight:    300
```

The foreground colour is used for the document labels, if no special label colour is defined, and for the display of the document count above the scrollable area. If the users want to set their own document label colour they have to use the XDefault `HarSearch*DocumentLabels.foreground`. Furthermore, it is possible to set the colour of the frame around the currently selected document and the colour of the background of the currently viewed document:

```
HarSearch*DocumentLabels.foreground:      white
HarSearch*currentdocbordercolor:          yellow
HarSearch*currentdoccolor:                FireBrick
```

By default the Similarity Map is not opened immediately after a search. If the user wants to use it from the start the following default value has to be set to `on`:

```
HarSearch.SimilarityMap:  on
```

The general foreground and background colour as well as the size of the scrollable area is set with:

```
HarSearch*SimilarityMap.foreground:          cyan
HarSearch*SimilarityMap.background:         #033b68
HarSearch*SimilarityMap.ScrollAreaWidth:     700
HarSearch*SimilarityMap.ScrollAreaHeight:    250
```

The numerous options of the Similarity Map may also be set to `on` or `off`:

```
HarSearch*SimilarityMap.AutomaticSimilarityThresholds:      off
HarSearch*SimilarityMap.AutomaticRelevanceThresholds:       off
HarSearch*SimilarityMap.ShowOnlyDocumentsWithEdges:         on
HarSearch*SimilarityMap.DisplayDifferentDocumentNumber:     on
HarSearch*SimilarityMap.OnlyMaximallyRelatedDocuments:      off
```

The number of document to display and the levels of maximally related documents are set with:

```
HarSearch*SimilarityMap.DocumentNumber: 50
HarSearch*SimilarityMap.DocumentLevels:  1
```

The default tree layout style may be:

`TreeVertical`, `TwoSidedHorizontal`, `TwoSidedVertical`, `PolarCoordinate`:

To set a tree layout the following default has to be set:

```
HarSearch*SimilarityMap.LayoutStyle: polarcoordinate
```

The possible default values for the icon style are:

`simple`,`scaled`,`coloured`,`ScaledAndColoured`:

The style can be set with:

```
HarSearch*SimilarityMap.DocumentIconStyle: Scaled
```

Using `simple`, `widthcoded`, `colourcoded`, `WidthAndColourCoded` the edge style can be set:

```
HarSearch*SimilarityMap.EdgeStyle: widthcoded
```

Also the thresholds can be defined:

```
HarSearch*SimilarityMap.maxRelevanceThreshold:    70
HarSearch*SimilarityMap.medRelevanceThreshold:    30
HarSearch*SimilarityMap.maxSimilarityThreshold:   85
HarSearch*SimilarityMap.medSimilarityThreshold:   45
```

The separations between the tree levels can be set as follows:

```
HarSearch*SimilarityMap.HorizontalSeparation:     30
HarSearch*SimilarityMap.VerticalSeparation:       40
HarSearch*SimilarityMap.LayerDistance:            20
```

These are the default colours used in the Similarity Map:

```
HarSearch*SimilarityMap.maxRelevanceColour:       #ff0000
HarSearch*SimilarityMap.medRelevanceColour:       #00ff00
HarSearch*SimilarityMap.minRelevanceColour:       #0000ff
HarSearch*SimilarityMap.maxSimilarityColour:      #ffff00
HarSearch*SimilarityMap.medSimilarityColour:      #00ffff
HarSearch*SimilarityMap.minSimilarityColour:      #ff00ff
```

If no edge colour coding is specified the `normalEdgeColour` is used to draw the edges:

```
HarSearch*SimilarityMap.normalEdgeColour:         white
```

The general options of the window that displays the similarities to a certain document are set with:

```
HarSearch*SimilaritiesToDocument.background:          #033b68
HarSearch*SimilaritiesToDocument.foreground:          cyan

HarSearch*SimilaritiesToDocument.ScrollAreaWidth:     300
HarSearch*SimilaritiesToDocument.ScrollAreaHeight:    150
```

To start HarSearch directly via the Harmony pull-down menu the following two lines have to be inserted into the XDefaults file for Harmony:

```
Harmony.Session.Collection.Tools.search.commandline:
              harsearch -apihost $localhost -apiport $apiport
Harmony.Session.Collection.Tools.search.menuentry: HarSearch
```

Harmony displays its API port at the start-up if the next variable is set to on:

```
Harmony.Session.displayapi: on
```

Appendix B

The Test File Format

The test file to simulate search results with document similarities has the following format:

```
<number of documents>

<global object id 1>;[<relevance score of object 1>]
<global object id 2>;[<relevance score of object 2>]
[...]
<global object id n>;<relevance score of object n>

<similarity values>
```

The relevance score is optional. If it is not specified it is generated randomly. The number of global object ids (GoIDs) must be equal to the specified document number. The similarity values have to be in the form of the lower triangular similarity matrix. Each line represents the similarities of one particular document to the documents defined before. For example, the first line with similarity values contains only the similarity between document 2 and document 1. The second line contains the similarities of document 3 to document 1 and document 2 and so on. The single values must be separated by semicolons. HarSearch also allows comment lines in the test file which have to begin with either '!' or '#'. A test file for 30 Hyperwave objects may therefore look as follows:

```
!number of objects
30
# GoIDs of the objects and score
0x811b9908 0x000ee197;1.0
0x811b9908 0x00067bcc;1.0
0x811b9908 0x000680ba;0.92
0x811b9908 0x000680c9;0.87
[...]
0x811b9908 0x0014c18e;0.2
0x811b9908 0x003728d1;0.17
```

```
0x811b9908 0x0033cd50;0.05

! Similarities between 30 documents
0.38;
0.18;0.13;
0.15;0.11;0.27;
0.8;0.19;0.12;0.46;
0.49;0.67;0.44;0.2;0.25;
0.43;0.49;0.83;0.37;0.26;0.26;
0.38;0.55;0.11;0.67;0.14;0.31;0.45;
[...]
```

Bibliography

[ALV] Altavista, search engine. `http://altavista.digital.com`.

[And96] Keith Andrews. *Browsing, Building, and Beholding Cyberspace: New Approaches to the Navigation, Construction, and Visualisation of Hypermedia on the Internet.* PhD thesis, Graz University of Technology, September 1996. `http://www.iicm.edu/keith-phd`.

[Blu96] Oliver Blumert. Seminarvortrag: Information visualisation I, 1996. `http://www-cui.darmstadt.gmd.de/~blumert/`.

[Boc74] Hans Hermann Bock. *Automatische Klassifikation.* Vandenhoek & Ruprecht, Göttingen, 1974.

[Bot93] Rodrigo A. Botafogo. Cluster analysis for hypertext systems. *Proceedings of SIGIR'93*, pages 116–124, June 1993.

[BSG+95] Steve Benford, Dave Snowdon, Chris Greenhalgh, Rob Ingram, Ian Knox, and Chris Brown. VR–VIBE: A virtual environment for co-operative information retrieval. *EUROGRAPHICS '95*, 14(3), 1995.

[CC92] Matthew Chalmers and Paul Chitson. Bead: Explorations in information visualization. *Proceedings of SIGIR'92*, pages 330–337, June 1992.

[CC93] Matthew Chalmers and Paul Chitson. *Using a Landscape Methaphor to Represent a corpus of Documents*, pages 377–390. Proc. European Conference on Spatial Information Theory, COSIT '93. Springer Verlag, Elba Island, Italy, September 1993. Andrew U. Frank and Irene Campari.

[CKP93] Douglass R. Cutting, David R. Karger, and Jan O. Pedersen. Constant interaction-time Scatter/Gather browsing of very large document collections. *Proceedings of SIGIR'93*, pages 126–134, June 1993.

[CKPT92] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. *Proceedings of SIGIR'92*, pages 318–329, June 1992.

[CLR92] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Intoduction to ALGORITHMS.* The MIT Press, 1992.

[CT94] Isabel F. Cruz and Roberto Tamassia. *How to Visualize a Graph: Specification and Algorithms.* Tufts University Brown University, 1994.

[DH96] Wolfgang Dalitz and Gernot Heyer. *HYPERWAVE; The new generation Internet Information System based on Hyper-G Technology.* dpunkt, Heidelberg, 1996.

[EEL] Electronic libraries, intelligent network agents, and information catalogues. `http://fox.cs.vt.edu/talks/KY95/`.

[ES90] Peter Eades and Kozo Sugiyama. How to draw a directed graph. *Journal of Information Processing*, 13(4), 1990.

[FBY92] William B. Frakes and Ricardo Baeza-Yates. *Information Retrieval; Data Structures and Algorithms.* Prentice Hall, 1992.

[HBT] HotBot, search engine. `http://www.hotbot.com`.

[Hea95] Marti A. Hearst. TileBars: Visualisation of term distributed information in full text information access. *Proceedings of CHI'95*, pages 59–66, May 1995. `http://www.acm.org/sigchi/chi95/Electronic/documnts/papers/mah_bdy.htm`.

[Hem93] Matthias Hemmje. LyberWorld– Eine 3D–basierte Benutzerschnittstelle für die computerunterstützte Informationssuche in Dokumentmengen. *Der GMD–Spiegel 1'93*, January 1993.

[HKW94] Matthias Hemmje, Clemens Kunkel, and Alexander Willet. LyberWorld– a visualization user interface supporting fulltext retrieval. *Proceedings of SIGIR'94*, pages 249–259, July 1994.

[IFS] InfoSeek, search engine. `http://guide-p.infoseek.com`.

[Kü93] Thomas Kürten. *INVITING; InterViews Tutorial Introduction Guide for beginners.* Technische Universität München, 1993.

[LCI+92] Mark A. Linton, Paul R. Calder, John A. Interrante, Steven Tang, and John M. Vlissides. *InterViews Reference Manual 3.1.* The Board of Trustees of the Leland Stanford Junior University, 1992.

[LYW] The LyberWorld Homepage. `http://www-cui.darmstadt.gmd.de:80/visit/Activities/Lyberworld/`.

[Mau96] Hermann Maurer. *Hyper-G now Hyperwave; The Next Generation Web Solution.* Addison Wesley, 1996.

[NFH+96] Lucy T. Novell, Robert K. France, Deborah Hix, Lenwood S. Heath, and Edward A. Fox. Visualization Search Results: Some alternatives to query-document similarity. *Proceedings of SIGIR'96*, August 1996. `http://ei.cs.vt.edu/papers/ENVreport/final.html`.

[OKS+93] K. A. Olsen, R. R. Korfhage, K. M. Sochats, M. B. Spring, and J.G. Williams. Visualisation of a document collection:The VIBE System. *Information Processing and Management*, 29(1):69–81, 1993.

[SBC97] Ben Shneiderman, Don Byrd, and W. Bruce Croft. Clarify search: A user-interface framework for text searches. *D-Lib Magazine*, January 1997. `http://www.dlib.org/dlib/january97/01contents.html`.

[SCB] Scatter/gather browsing communicates the topic structure of a very large text collection. `http://www.soe.berkeley.edu/~schank/parc/pp_txt.htm`.

[Spo93] Anselm Spoerri. InfoCrystal: A visual tool for information retrieval. *Proceedings of Visualisation '93*, pages 150–157, October 1993.

[STT81] Kozo Sugiyama, Shojiro Tagawa, and Mitsuhiko Toda. Methods for visual understanding of hierarchical system structures. *Man and Cybernetics SMC-11*, 2:109–125, 1981.

[UCB] UC Berkeley digital library project. `http://elip.cs.berkeley.edu/tilebars.html`.

[VRV] VR-VIBE screen shots. `http://www.crg.cs.nott.ac.uk/crg/Research/pits/vrvibe/`.

[WMB94] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes, Compressing and Indexing Documents and Images*. Van Nostrand Reinhold, New York, NY 10003, 1994.