SearchVis

Visualising Search Result Sets
Using a Force-Based Method
to Form Clusters of Similar Documents

Susanne Mayr

SearchVis

Visualising Search Result Sets
Using a Force-Based Method
to Form Clusters of Similar Documents

Master's Thesis

at

Graz University of Technology

submitted by

Susanne Mayr

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

October 1997

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

SearchVis

Visualisierung von Suchergebnissen
unter Verwendung eines Ansatzes mit Federkräften
um Gruppen ähnlicher Dokumente zu bilden

Diplomarbeit
an der
Technischen Universität Graz

vorgelegt von

Susanne Mayr

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

Oktober 1997

Diese Arbeit ist in englischer Sprache verfaßt.

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

Abstract

As human knowledge increases, so the volume of electronically available information grows. Finding specific information becomes more difficult and ever more matches are returned in response to a search query. Since quantity is seldom quality, numerous approaches to make sense of search result sets have been proposed.

This thesis describes an approach called SearchVis to visualise search result sets, which is based on an approach by Matthew Chalmers described in his 1996 paper *'A Linear Iterative Layout Algorithm for Visualising High-Dimensional Data'*. The visualisation concentrates on the similarities between the documents retrieved. An animated, force-based technique produces clusters of similar documents. Through this technique similar documents are attracted and non-similar documents repelled.

SearchVis allows the user to adjust the visual discrimination of the clusters using different parameters. It was tested with a varity of test data sets for a wide range of parameter settings. In order to reach as wide an audience as possible, SearchVis was written in Java.

Kurzfassung

Aufgrund des ständig wachsenden menschlichen Wissens, steigt auch die Menge der elektronisch verfübaren Informationen. Die Suche nach einem bestimmten Thema wird dadurch immer schwieriger. Wenn man eine Suchabfrage startet erhält man immer mehr "relevante" Dokumente als Antwort. Nun ist aber Quantität selten mit Qualität gleichzusetzen, daher wurden bereits viele verschiedene Ansätze vorgeschlagen, welche die Suchergebnisse in einer verständlicheren Form darzustellen versuchen.

In dieser Diplomarbeit wird ein neuer Ansatz, namens SearchVis, vorgestellt. Dieser Ansatz basiert auf einer Methode die von Matthew Chalmers in einem Artikel 1996 namens *'A Linear Iterative Layout Algorithm for Visualising High-Dimensional Data'* beschrieben wurde. Hier werden die Ähnlichkeiten zwischen den einzelnen Ergebnis Dokumenten graphisch dargestellt. Dies geschieht mittels eines iterativen Ansatzes mit Federkräften; d.h. ähnliche Dokumente ziehen sich an und unähnliche Dokumente stoßen einander ab. Auf diese Art werden Cluster, das sind Gruppen, von ähnlichen Dokumenten gebildet. Die Cluster-Bildung wird animiert, indem jeder Iterationsschritt angezeigt wird.

Um die Cluster Diskriminierung einzustellen, kann der Anwender verschiedene Parameter einstellen. SearchVis wurde mit einer Vielzahl von verschiedenen Test Daten über einem großen Parameterbereich getestet. Damit dieser Ansatz von möglichst vielen Benutzern verwendet werden kann, wurde er in Java geschrieben. Dies garantiert Übertragbarkeit und Plattformunabhängigkeit.

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.

Acknowledgements

I would like to thank my supervisor Univ.Ass. Dr. Keith Andrews for his help and patience during my whole thesis and for his advice in writing and correcting this work. Special mention and thanks go to Dieter Freismuth, Georg Meszaros, Bernhard Zwantschko and Klaus Schmaranz of the Dino group for their help and understanding. Finally, thanks to everyone at the Institute for Information Processing and Computer Supported New Media (IICM), working there was really agreeable and fascinating.

Last but not least, my most heartfelt thanks to my family, my parents, my brother, my sister and my grand-parents for their support and encouragement during the whole time of my studies as without them this thesis would not have been possible. I would like to dedicate this work to them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays a vast amount of information is available electronically. This information is stored in textual as well as multimedia databases. Searching for a specific topic, nearly every relevant information can be found, even if this search is performed on-line on the Internet. Nethertheless quantity is not quality, and therefore much of the retrieved information may not be of interest. Because of this problem the retrieval of demanded information becomes more difficult and complex.

To allow a distinction of relevant and non-relevant information new ways of visualising information bases of search result sets have been developed. These visualisations try to avoid the problem of getting "lost in Hyperspace" and to help the user to retrieve the desired information in an effectively and easily way.

This thesis describes an application called SearchVis, which is based on an approach by Matthew Chalmers in his 1996 paper *'A Linear Iterative Layout Algorithm for Visualising High-Dimensional Data'* [Cha96]. It visualises search result sets using a force-based method to form clusters of similar documents. This system places the documents at an initial position on the screen and defines forces according to the document similarities and the document distances. The forces cause the documents to move towards their final position.

Chapter 2 introduces the main principles of information retrieval (IR). IR tries to find an search result set according to the users' query using mechanisms such as indexing and the vector space model. These mechanisms help to index documents and to find document clusters. The retrieval effectiveness and document similarities can also be determined.

Chapter 3 focuses on iterative placement models. Starting from an initial place, these models try to reach a minimal energy state by moving iteratively to the final position. Three main models are presented: force directed placement, simulated annealing and a linear iterative algorithm.

Chapter 4 presents and discusses several systems which are directed to the visualisation of information spaces and search result sets.

Chapter 5 describes the main basic concepts of the Internet. Based on this information an multimedia information system of the next generation, Hyperwave, is presented. This powerful server offers solutions to many problems of the first generation. Finally Dino (distributed interactive networking objects) a framework for distributed applications on the web is introduced. Dino offers the possibility to create a distributed application on the web. It offers easy ways to communicate with multimedia information systems like Hyperwave and other Dino applications. As Dino is written in Java it offers the users on the Internet to work and talk together independently of the used platform.

Chapter 6 explains the concept of SearchVis. In this chapter, all steps which lead to the application are described. This includes the developement, the actual algorithm and the differences to the underlying linear iterative approach. Another part of this chapter concerns the available options, which help to obtain a good result.

Chapter 7 describes how an application can be added to the Dino system. Examples of the necessary code parts are shown and explained.

Chapter 8 is an evaluation of different parameters. Those can be set in the general and the expert option panels.

Chapter 9 outlines some ideas for future work and research of SearchVis, which may be implemented. The highest priority should be given to the implemention of a search interface embedded in Dino and to the provision and integration of real document similarities. As soon as these are integrated, a usability evaluation can provide feedback for further improvements.

As no real search result sets and document similarities are available, testfiles are used. The format of those files are explained in Appendix A.

Chapter 2

Information Retrieval - IR

Information retrieval is a special term which is used in computer science for the search, that means identifies and retrieves, of information in an information base in response to an information query. In this chapter we will discuss the historical background of IR and about some of the most well-known IR methods such as the inverted index and the vector space model.

2.1 The History of IR

In the beginning, human knowledge was rather small and passed on by speech. But the knowledge increased over time and other possibilities where needed. One great step further was writing. Now everything could be written down and transmitted to distant places. Another advantage of writing was that the information could also be stored.

Quite soon large libraries such as the famous one in Alexandria existed. Now a new problem appeared, how to find the right information in those masses of books? One great help was the librarian, who mostly knew which books he could offer. But as the libraries increased, this was not enough any more. New methods had to be found. The Sumerians were the first to be known because of their literature catalogue, which can be called the first index. Nowadays the methods of indexing are already quite widespread. In libraries different indexes are used to catalogue articles or books. The user can search by looking for the author, the title or certain keywords. Indexes exist not only in libraries, nearly every book has its own index.

In these days the traditional library index has been replaced by computers. This changed the way of storing and getting information. Computers do not only offer search by title or keywords, but may generate an index of all, or at least nearly all, terms within a document. This index is the so called full text index.

Information retrieval is a subfield of computer science that deals with the retrieval and the automatic storage of documents. The general definition of IR could be as follows: *the identification and retrieval of particular matching documents from an information base in response to an information request (query)* [And94, page 1]. In most cases the document will be a text, but it could also be of any other type, such as an image or video clips.

An IR system matches the query with the terms in the documents and returns the retrieved

Title	Author	Publ.	Year
title 1	author 1	new pub.	1984
title 1	author 2	new pub.	1984
title 2	author 3	pub. x.	1985
title 3	author 1	pub. x.	1987

Relational DBMS

.T
title of the book or article
.A
first author
second author
.P
new publishing co.
.Y
1984

IR system

Figure 2.1: Representation of data in DBMS and IR systems [And94, page 1a].

set of relevant documents. The response is based on statistical methods, which means it may miss some very relevant documents and have some non-relevant instead, depending on the retrieval technique and the terms used.
The documents of an IR system are sometimes stored in a database management system.

2.2 DBMS contra IR

DBMS are characterised by structured, homogeneous records with a specified set of fields also called attributes. The contents of these fields are unambiguous. To retrieve some information an exact match between query and record fields is needed, this is a Boolean retrieval.

IR systems are quite different, they consist of large unstructured documents. Each document may have some formal attributes as title, author or keywords. The attributes are indexed. In an IR system the text can also be indexed, this is the so called full text index. The match in IR systems is a partial one, between the query and the contents of the index, a full text retrieval. As the index in IR systems is much bigger, a substantial effort is needed to generate it and to match the query. We could say, that DBMS is a subfield of IR systems, that has only attributes and the attribute index.

2.3 Indexing

What possible approaches do we have to find out if a certain term is present in a document?

- Sequential Scanning: is a direct comparison between the query terms and the documents. That means we scan each document to find out whether the specific terms are present or not. This not only sounds like a great deal of work - it is. Therefore this

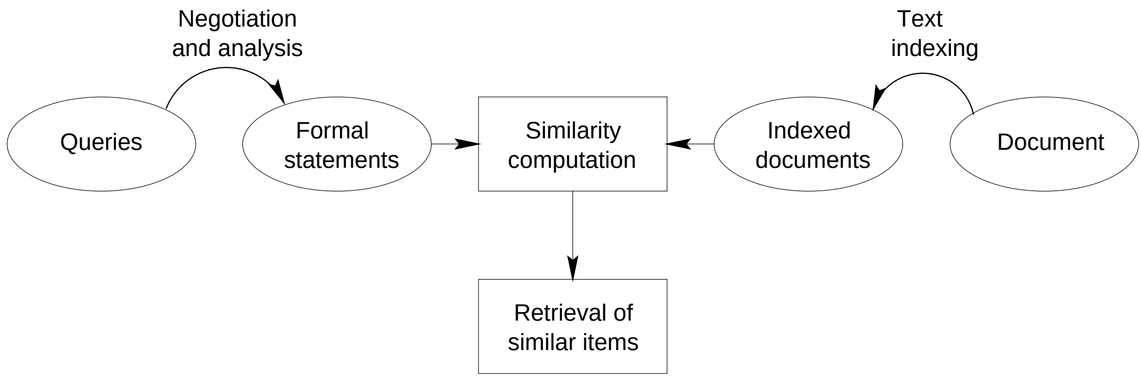

Figure 2.2: Information retrieval using the inverted index [And94, page 2a]

approach is only useful with small document collections, or even better with only one document.

- Inverted Index: here three steps are passed through. A content analysis to generate a indexed representation of the document. This step is only done once, then the index is stored for later use. The next step is a query formulation to transform the query to a formal query statement (done at every search). And finally the comparison, where the formal query is compared with the terms in the index. Here some additional effort is necessary to build the index, but than the result is discovered in a quite short time. An index can be described as a collection of terms pointing to the documents where they are included. There exist two kinds of indexes, depending on what to index:
  - attribute index: all the attributes of the IR system such as title, authors, keywords are indexed
  - full text index: every word in every document of the IR system gets indexed. This index is very big, therefore it is reasonable to compress it.

2.3.1 Inverted Index

First we construct a normal index. This index is called *document term matrix*. There each row represents a document and each column signifies whether a particular term is present (= one) or not (= zero), a row can also be called a term vector, as it consists of terms. This can be seen as an equivalent to the index at the end of a book. In this case it takes a rather long time to find out if a term is present in a document, as the system has to go through all term vectors. It would be more reasonable to use document vectors therefore the index is inverted. With this inverted index documents corresponding to query terms are easily found. Inverted Indexes are also quite practical using boolean queries. The merging operations can be performed in linear time, as the documents in the document vector appear in increasing order. All boolean queries contain a merge step as first step, then:

- VECTOR-Query (Ti, Tj, Tk) : output records in decreasing order of number of entries in merged list

- AND-Query (Ti and Tj) : output only the duplicated records
- OR-Query (Ti or Tj) : output all distinct records
- NOT-Query (Ti and not Tj): output all not duplicate records

Up to know the inverted index is quite useful and practical, the response is rather fast and exact queries are possible. But we cannot rank the retrieved documents according to their relevance, as we have no relevance information. The formulation of useful queries is often difficult and requires exact spelling of the search terms. To improve these disadvantages we can extend the inverted index using term operations.

Term Operations

Our actual inverted index contains document vectors where the terms have no special meaning. We ignore also further information, that we could use for measuring the relevance of the document concerning this term. This lack of information can be easily undone. There exist several operations, which improve the relevance of the terms. Several possibilities exist. Some words can be excluded or the cases can be ignored. Even weights for terms can be used. Some of the frequent used operations are explained in the following:

Case Folding

One problem which might occur when you search for one term, is that the search fails because the query is case sensitive. There exists a really simple method to avoid this - named case folding. All letters of a term are converted to their lowercase equivalents. Naturally this has be done for the terms in the document term matrix and also for those in the search query. This operation does not only prevent case problems, it also reduces the size of the index considerably. If a case sensitive search is wished this can be performed later, using a post-retrieval scan with exact pattern matching.

Stop List

A lot of words like “it”, “the”, “a”, which are of no special interest, exist very often in every English document. Now the questions arises if they should occur in the document term matrix. One possibility to avoid them is to collect them in a so called stop list. This stop list may reduce the size of the inverted index up to 40 percent. As the index is normally quite huge it is mostly compressed. In this case the benefit of the stop list is less clear, as the stop words are those words which have a high frequency and therefore a very high compression rate. Here the amount of saved space due to stop lists gets rather small.

Furthermore, omitting words can cause problems, if users are interested in finding exactly those words. Just think of some literature, where a stop word might be the main term (study of Shakespeare’s use of the verb “to do”). Another problem is, that a lot of words have more than one meaning. Therefore it might happen that one important word is also removed by the stop list. So on principle a stop list is a great thing, but it is quite tricky to build.

Stemming

Stemming stands for stripping common suffixes such as “ion”, “ed” or “ing” from the terms. Using this operation several terms might be replaced by only one term. The final term remains in a root form, negotiating tense and plurality. If stemming was used on our index, then stemming has to be used on the query to achieve a result. You should be careful, although this process may reduce the matrix up to 50 percent, is not useful to use it on all attributes. Just guess what mess would come out if you would stem the author field.

So, stemming is a quite powerful process, that can help a lot. In some way it is also dangerous to use stemming as the search result can get a bit ambiguous. If you want to use stemming, make sure, that the algorithm has a detailed knowledge about the language of the recent document.

Thesaural Substitution (Synonyms)

Everybody tries to write in a good style and therefore nobody does want to use one word all the time. One possibility to avoid this is to use some synonym. For example instead of *exciting* there can also be used *thrilling* or *gripping*. To build a rather small index, these synonyms are annoying. To save some space these synonyms should be replaced by one significant term.

The same replacing algorithm has to be used on the query - just imagine what would happen if the terms would be represented by *exciting* in the index and by *thrilling* in the query. Nobody would find anything!

Term Weights

The terms in the index are normally represented by zero or one indicating the presence. To gain some further information, weights for the terms could be used. One possibility for assigning the weights:

- 10 Points: the term is present in the document title
- 5 Points: the term is a keyword
- 1 Point : for each occurrence in the text

The summation of all these points is the so called term weight. In the index the term weight is represented by:

- a relative term weight $= \frac{\text{term weight}}{\text{sum of all term weights in the document}}$ or
- a normalised document vector.

Now the ranking of the document depending on their similarity to the specific term can easily be obtained. Term weights offer the possibility to vary the weight depending on the position. Another fine thing is, that the user can weight the search terms. It is also important to mention one negative effect: query matching gets more complicated.

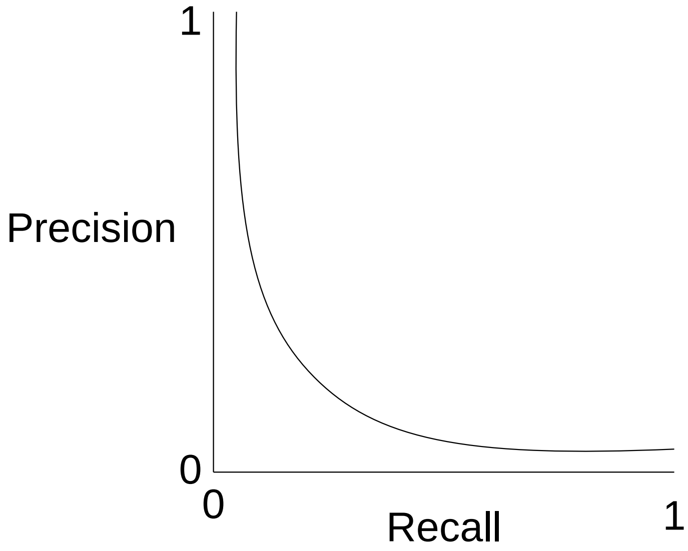

Figure 2.3: The Precision and Recall graph. [FBY92, page 11]

2.3.2 Measuring Retrieval Effectiveness

As we do not retrieve only relevant documents we need some kind of measurement to estimate retrieval effectiveness. There exist many different ways to do so, but the most common measures are precision and recall. Precision, P, can be described as the proportion of retrieved documents which are relevant [And94, page 7]:

$$P = \frac{\text{number of all relevant documents which were retrieved}}{\text{number of all retrieved documents}}$$

Recall, R, can be characterised as the proportion of all relevant documents which were retrieved [And94, page 7].

$$R = \frac{\text{number of all relevant documents which were relevant}}{\text{number of all relevant documents}}$$

Both precision and recall take a value between zero and one. When we have a narrow search we get a high precision and a low recall. Using a broad search it is the other way around. So we see, that precision and recall are inversely related. That is, when precision goes up, recall typically goes down and visa-verse.

Since one often wishes to compare IR performance in terms of both precision and recall, methods for evaluating them simultaneously have been developed. One of these combined measures has been found by van Rijsbergen (1979). Here the evaluation measure E is defined in [FBY92, page 11] as:

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

In this formula P and R represent precision and recall, b is a measure of the relative importance, to a user, of precision and recall.

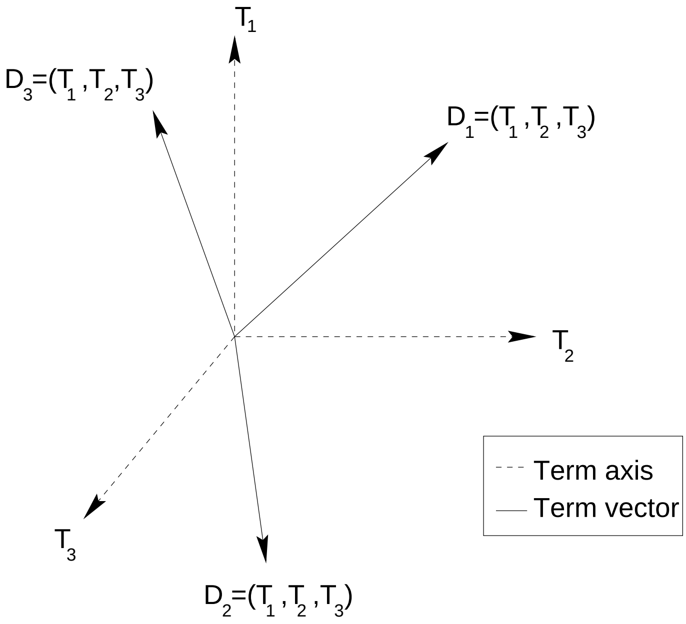

Figure 2.4: The vector space model [And94, page 8a]

2.4 Vector Space Model

For this model we also have to construct a document term matrix. This time the matrix is not inverted, the term vectors are used. That means each document and also each query is represented by a term vector. A vector consists again of the values zero or one to signify absence or presence of a term. Mostly normalised term weights are used to signify the importance of this term (see Section 2.3.1). The documents and the query vectors can be envisioned as a n-dimensional vector space, where n corresponds to the number of unique terms in the data set.

The vector space model makes some stipulations. It calls for uncorrelated and orthogonal term vectors. Another demand is that the term vectors have to form a basis for the vector space.

2.4.1 Vector Matching - Similarity Measures

In this model it is quite easy to find out the similarity between two documents or a document and the query. The most popular approach for this kind of matching is based on the cosine correlation used to measure the cosine of the angle between vectors.

$$
\begin{aligned}
xy &= |x||y| \cos a \\
sim(x,y) &= \cos a = \frac{xy}{(|x||y|)}
\end{aligned}
$$

When the vectors are normalised, the formula gets even more simple as, $|x| = 1$, $|y| = 1$:

$$sim(x,y) = xy$$

This vector matching provides some other applications:

- Ranked output: the documents retrieved in response to a query can be easily ranked using the received similarities. The similarities are sorted in descending order and then the belonging documents are already in the right order.
- Clustering: as vector matching provides also the possibility to compare documents, clustering can be done quite easy.

2.4.2 Relevance Feedback

One problem in using IR systems is, that mostly not only relevant but also quite a lot of non-relevant documents are retrieved. Sometimes the user is not aware of that fact, sometimes only one relevant document was needed. Many situations demand a high recall. This is a problem since relevance judgements are subjective and unreliable. That is, different judges will assign different relevance values to a document retrieved in response to a given query. Nevertheless there are some possibilities to receive better results. One is to conduct the searches iteratively:

1. form tentative query
2. evaluate relevance of the retrieved documents
3. improve query, go back to step 2

In this way the query moves away from non-relevant items towards relevant one.

How to evaluate the relevance and improve the query? In this case a relevance feedback algorithm such as *Ide dec-hi* can be used. First the user assesses the relevance of some retrieved items, than a matrix for relevant documents R is build by adding n_1 relevant vectors. Finally the top non-relevant document is assigned to the vector S.

$$\text{Ide dec-hi} \qquad Q_1 = Q_0 + \sum_{i=1}^{n_1} R_i - S$$

where Q_0 is the vector for the initial query [FBY92, page 255].

2.4.3 Vector Space Model and Inverted Index

The vector space model and the inverted index have some parts of their construction in common. Both models start by building a document term matrix. In the following the inverted index inverts this matrix to use the document vector. The vector space model uses the term vector. The inverting step can still be performed to reach a compatible state with the inverted index or visa-verse.

This fact offers the possibility to use the vector space model atop the inverted index. Now all facilities of the vector space model can be used in the inverted index. This is especially useful because of the simple methods for similarity evaluation, relevance feedback and clustering.

Chapter 3

Iterative Placement Models

In this chapter different iterative placement models will be presented. Here the objects are moved iteratively until the system reaches am minimal energy state, or any other defined situation occurs.

3.1 Force Directed Placement

Force directed placement is a well-known technique for drawing general, undirected graphs. It is based on the spring model. The idea for this approach was taken from mechanical physics. An observation of a system of masses connected via springs. To use this approach for graph drawing, the vertices can be thought to be steel rings (masses) and the edges can be regarded as springs. Those springs represent repulsive and attractive forces among the steel rings.

To start this system, the rings are positioned at an initial position. Afterwards the rings are moved iteratively according to the forces until the system reaches a minimal energy state. A stable state is reached when the resulting forces are equal to zero. Now the steel rings are drawn as points or small circles and the springs as a straight line connecting two rings.
Some generally used aesthetical criteria accepted by the force directed placement are:

- ensuring that edge lengths are uniform
- minimising the edge crossing
- revealing symmetry
- distributing vertices only

3.1.1 Magnetic Spring Model

To be able to draw directed graphs and trees an extension to the normal force directed placement has to be done. This can be done using magnetic fields. The magnetic spring model introduces another aesthetic criteria by constraining *links* to specified orientations.

The approach of the magnetic spring model [SM95] replaces vertices and links with rings and magnetic springs. A magnetic spring can be defined as a magnetised object which has characteristics from springs and magnetic compasses. Links can be divided into two classes: magnetic links and non-magnetic links. Non-magnetic links can be replaced using a common spring.

An additional part of a magnetic spring model are the magnetic fields. They can be separated into two different general types the standard magnetic fields and the compound magnetic field. Standard magnetic fields can be parallel fields, polar fields and concentric fields. The compound magnetic fields are composed from standard magnetic fields. Two different types can be distinguished, orthogonal fields consisting of two orthogonal parallel fields and polar-concentric field consisting of a polar and a concentric field having the same origin.
Taking into account all these components of the magnetic spring model three types of forces can be considered:

1. attractive and repulsive forces caused by the springs between neighbours
2. repulsive forces between every pair of non-neighbouring vertices
3. rotative forces on magnetic links caused by the magnetic fields

Composing these three forces the general force on each vertex can be calculated. According to this force a new position can be found.

3.1.2 Discussion

Unfortunately this model easily gets stuck in local minima as only a state with less energy is accepted.

Some extensions to this approach have been made. The magnetic spring model is one of them. It adds an idea of controlling the orientation of edges by defining magnetic fields. This offers the possibility to draw other classes of graphs as trees, directed graphs and mixed graphs (graphs with several types of links) in a simple unified manner [SM95].

3.2 Simulated Annealing

Simulated annealing is an optimisation method which is derived from statistical mechanics. It differs from standard iterative improvement methods by allowing moves that spoil rather than improve the temporary solution.

The physical model deals with the cooling down of liquids or some steel melts. This means the transformation of an unordered liquid into an ordered solid status. Depending of the velocity of cooling down the energy reaches different smaller values.

- **rapid cooling:** results in an amorphous structure representing a local minima. This state has a quite small energy state, but not the possible minimum.

- **slow cooling:** a totally ordered form called crystal representing the minimum energy state is reached.

This behaviour can be described using the Boltzmann distribution:

$$p(E) \cong \exp \frac{-E}{kT}$$

In this formula $p(E)$ is the probability distribution of the energy values according to the states E. This value is described as a exponential function of the temperature T and the Boltzmann constant k. For each temperature value all energy values E have nonzero probability. Therefore the system can change its state to one with higher energy. At low temperatures the system allows only few movements with low energy states. As the global minimum should be achieved reaching a temperature of zero.

To transform this system into a drawing model the Boltzmann constant k is replaced by the value one, since all temperatures are artificial. To find the probability with which the system moves from one position with the energy E_1 to another with the energy E_2 can be found deriving the Boltzmann distribution:

$$\exp \frac{-(E_2 - E_1)}{T}$$

That means that every energy improvement is accepted and energy increasement may be accepted. This can be used for general complex optimisation problems. The following entities have to be determined to be able to apply a simulated annealing method:

- *set of configurations:* or states of system, including an initial configuration. Often the initial state is not fixed but chosen at random.
- *generation rule for 'new' configurations:* This can be done by defining the neighbours of each configuration and choosing the next configuration randomly.
- *cost function:* as an analogue to the energy. This function has to be minimised over the configuration space.
- *cooling schedule:* Some control value which can be compared with the temperature. This schedule has to include an initial value and rules defining when and how to change the control value.
- *termination condition:* to finish the calculation. This condition can be based on the time and the values of the cost function and/or the control parameter.

If all these entities are defined the simulated annealing method can be started performing the following steps:

1. choose the initial values of the configuration and the control value
2. in this step the following action are repeated until some condition is satisfied:
   (a) a new configuration is chosen according to the generation rule

   (b) the cost function for the two configurations is calculated. Depending on the result the new configuration is set to be the actual configuration or not.

3. decrease the control value according to the cooling schedule

4. examine the termination condition, if it is not satisfied go back to step number two.

3.2.1 Discussion

Simulated annealing can be used to solve complex optimisation problems such as the well-known travelling salesman problem. Nevertheless simulated annealing is not always suitable to those problems. A basic requirement for the cost function has to be taken into account. It should go down in steps of reasonable height. A very steep descent will probably lead to a local rather than a global minima. Another important point concerning the cost function is the time and effort needed to calculate a result. As these calculations are carried out repeatedly and constitute the major computational task. In principle simulated annealing is a quite time consuming method and therefore offering good results.

In general simulated annealing is very successful in finding solutions near the global minima. The global minimum itself is detected seldomly.

3.3 A Linear Iterative Algorithm

This represents the method derived from the paper *'A Linear Iterative Layout Algorithm for Visualising High-Dimensional Data'* [Cha96]

The force based models for N-body problems have a computational complexity of O(N^2) per iteration. Therefore this approach looks for an algorithm with a linear computational complexity and the same benefits as force based models. The documents in this approach are represented using the vector space model (see Section 2.4).

Spring-based forces F_{ij} act between each pair of objects *i* and *j* and cause a separation in the layout. The magnitude of each force is proportional to the magnitude of $d_{ij} - g_{ij}$. Here g_{ij} represents the geometric distance between the two objects in the layout and d_{ij} the high-dimensional distance. This high-dimensional distance can be calculated using the following equation:

$$d_{ij} = (1 - s_{ij}^p)$$

In this formula s_{ij} is the scalar product of the term vectors of *i* and *j*, and *p* is an exponent to increase the discrimination of the distance metric.

Instead of doing all the N^2 pairwise force calculations as in standard force directed placement (see Section 3.1) these calculations are only done between each object and the members of two bounded sets. This causes the approach to have linear complexity. Each document *i* has both sets:

1. *neighbour set V_i:* is stored as a dynamically-maintained list of references to *neighbour* objects. These neighbours are found iteratively during the construction of the random set, they do not have to be the closest objects in the space. This set is of constant length

V_{max}. The entries of this set are stored in order of their distance to the object *i* in the high-dimensional space. The distance of the farest neighbour is saved in a variable called $maxDist_i$.

2. *random set S_i:* this set is constructed anew during each iteration. It consists of a random subset of the information space of a specified constant length S_{max}. No member of the neighbour set are allowed to appear in this set.

To fill the sets the following steps are made. A random object *j* for the set S_i is selected. Then the high-dimensional distance is calculated. If this distance is smaller than $maxDist_i$ this value is inserted in the neighbour set V_i at the appropriate position. This action might remove the most distant neighbour and cause a new $maxDist_i$ value. If the distance is greater, the object is added to the random set S_i. As soon as S_i is filled the forces on *i* are calculated:

$$F_i = \sum_{v \in V_i}^{V_{max}} F_{iv} + \sum_{s \in S_i}^{S_{max}} F_{is}$$

An additional force which is added for some fixed numbers of iterations is a spring force between each object and the plane. This force is used to produce a near planar 2.1 dimensional layout, while unassociated cluster separate without any overlaying as with projections.

When all forces acting on each particle *i* are calculated, an integration step, normally fourth order Runge-Kutta, to update position and velocity can be done. This adds jitter to be able to break out of local minima. For stability, viscous damping proportional to the velocity is added.

The termination of this procedure can be reached after a specific iteration number or automatically when the stress value has little decrease over several measurements. This stress value has been derived from mechanical stress in spring systems. It can be described as the residual sum of squared errors of all interobject distances:

$$\text{Stress} = \frac{\sum_{i<j} (d_{ij} - g_{ij})^2}{\sum_{i<j} g_{ij}^2}$$

As in standard spring models (see Section 3.1) the stress should reach a minimum value. As this evaluation is of high complexity $O(N^2)$ it is not useful to perform it every iteration, because this would increase the overall complexity to $O(N^2)$. This calculation is usually done every $\sqrt{N}$ iteration.

3.3.1 Discussion

Evaluation found out, that the general values for S_{max} and V_{max} can be quite small because of a indirect linkage via neighbours. Therefore the neighbour will indirectly pull another object closer to the specific object. A direct linkage via neighbour sets will cause an extra close pull.

A significant factor for this system is the value *p*. It was found out, that a large value for *p* reduced the stress value significantly. This value led also to subjectively better layouts such as clearer separation of clusters.

Regarding the insertion steps to build up the two sets, it can be said that they have a very high initial value, which falls down fast.

Chapter 4

Information Visualisation

Using IR systems we receive a lot of information. Normally this information is presented to the user as a list containing attributes like author and title. To display further attributes and give the user a better overview, new visualisation possibilities were needed. Some of them will be described in the following sections.

4.1 General Principles of Visualisation Systems

The goal of visualisation systems is to present as much information as possible, as understandably as possible. In other words the user should intuitively recognise relevant documents. A possibility to navigate through the abstract information space to access one document is needed (for example, in lists the ability to scroll).

The vector space model (see Section 2.4) can be regarded as a simple visualisation system. All documents can be considered as points in the corresponding n-dimensional space.

As points do not provide many possibilities to visualise some attributes, other forms can be used. These shapes are often metaphors derived from real world objects. Using such metaphors help the user building up a mental model of this information space. Some forms are taken from simple geometry like rectangles, triangles, circles, spheres, cubes, or even more complex like trees and landscapes.

Further improvement of usability can be done using other visual coding principles. Some of these codes are described in the following:

- *Size Coding:* is a possibility to indicate quantitative informations like number of elements in an object or the relevance of a document. It can also be regarded as an indication of the size of a document.

- *Colour Coding:* is also used to show quantitative information; but it can also represent types of objects.

- *Brightness and Saturation Coding:* can be used for the same purposes as size coding

- *Shape Coding:* shapes can be used to show the type of an document, for example a rectangle for text, to provide an easier distinction

- *Proximity Coding:* the distance between two documents in the space can be proportional to the similarity of these documents

A problem is, that it is not possible to display such a high dimensional space directly and the user mostly will have problems to see through it. Therefore a mapping procedure is applied to transfer it into a lower dimensional space (in most cases 2-, or 3-dimensional) which is more familiar to the user. A well-known mapping is called a multidimensional scaling (MDS) [CC92].

4.2 VR-VIBE

VR-VIBE [BSG$^+$] is a statistical method to visualise data. Collections of data (often documents) are analysed in an attempt to group objects together according to some measure of *semantic closeness*. This system is an extension of a previous two-dimensional system called VIBE [OKS$^+$93] into three dimensions. Users specify keywords that they wish to use to generate the visualization and place these keywords in three dimensional space. Representations of the documents are then displayed in the space according to how relevant each document is to each of the keywords.

Special about VR-VIBE is, that multiple users can apply queries simultaneously and explore these results looking for information of special interest together.

4.2.1 Document Visualisation

The spatial position of each document corresponds to the attraction of this document to the different queries. This attraction represents the relevance. That means the higher the relevance the closer is the document and the other way around.

The query can be arranged by a user within a spatial framework. As soon as a position is associated to a query this query is called *Point of Interest (POI)*. Looking at very few POIs the distribution of the documents is quite simple: two POIs, the document is positioned on a line between these two POIs according to the relative attraction. This approach works fine with up to four queries displaying the result in three dimensions. Then ambiguity occurs.

This ambiguity concerns the relevance of a document. A document having the same high relevance to all POIs is positioned at the same place as a document being weekly attracted to all POIs. Now two documents share the same place, one very relevant one and one quite non-relevant one.

Despite of the ambiguity, the user is allowed to define as many queries, containing as many keywords, as needed. As there exist some methods to solve this problem. One possibility is an movement of the POIs by the user (see Section 4.2.2). To show general relevance information two different visualisation styles can be used:

1. The POIs are placed in a two dimensional plane. Now the documents are positioned according to their relative attraction. To get some relevance information the third dimension is used. Documents with a higher relevance score are placed high above the POI plane, as shown in Figure 4.1.

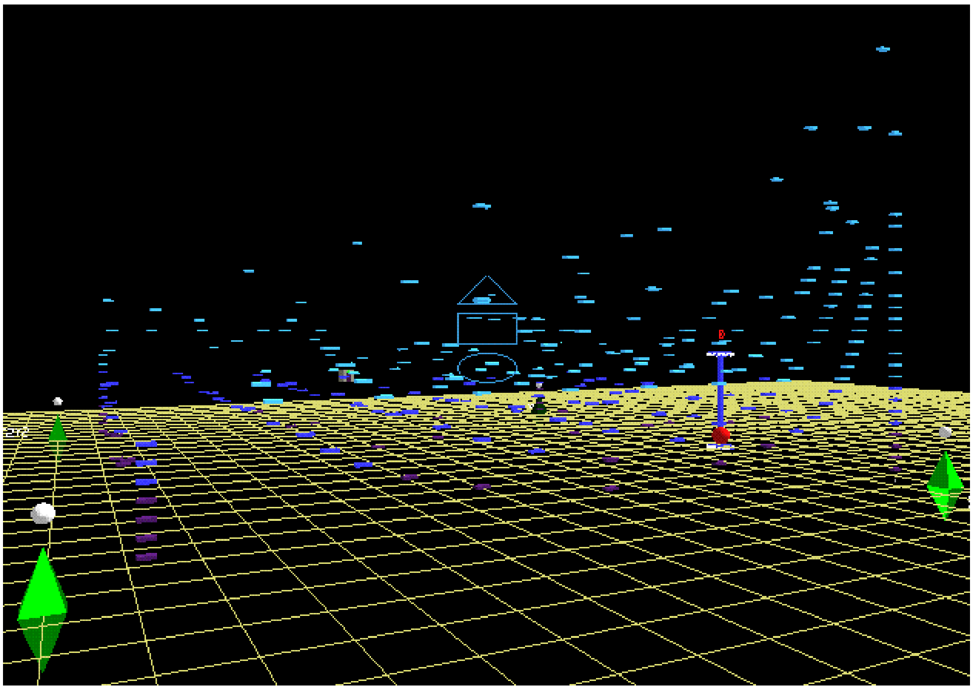

Figure 4.1: The POIs are placed in a two dimensional plane, the third dimension visualised the relevance to the query[Sno].

2. The POIs are positioned freely in the three dimensional space. And the documents are positioned according to their relevance. In this case the overall relevance is visualised using *size- or shade coding*. The more relevant a document is, the larger or brighter it is displayed. This effect is shown in Figure 4.2.

Up to now, only the relevance of the documents has been discussed. VR-VIBE also specifies the overall significance of each POI using shade coding.

4.2.2 User Interaction

The VR-VIBE system runs on a multi-user platform. That means multiple users work simultaneously. Each user can navigate free through the information space. Special document actions can be done:

- *Selection:* To get some additional information like the title, one or more objects can be selected. This causes the concerned icon to change colour to make this selection visible for all users, as shown in Figure 4.3.
- *Marking:* If one user thinks a document to be of special interest for one or all other users he can mark this object. To mark a document a label with some information provided by the user is attached.
- *Viewing:* This allows the user to view the whole document invoking an appropriate browser (e.g. a World Wide Web browser, have a look at Figure 4.4).

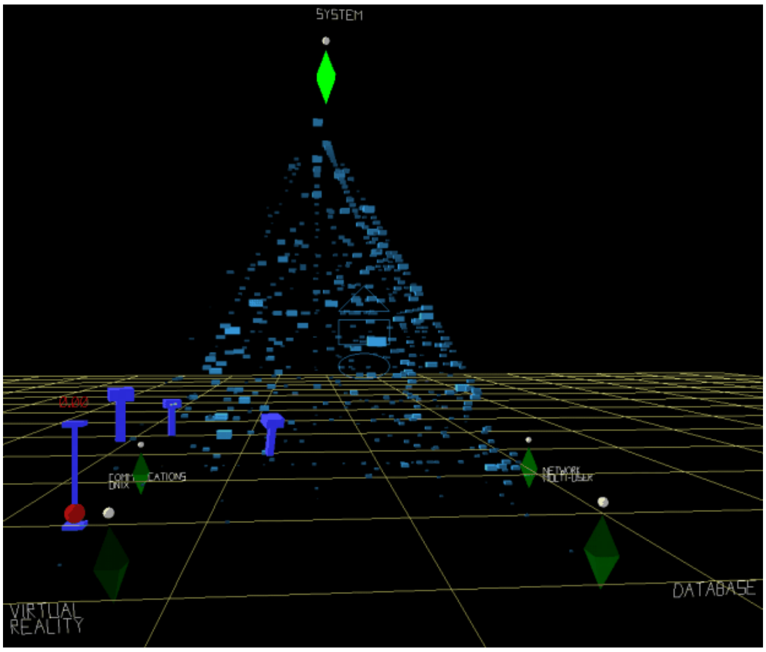

Figure 4.2: Here the overall relevance of the documents to the query is visualised using size- or shade coding. The query terms are system, The POIs are placed in the three dimensional space. In this information space several users can be seen represented by a figure looking like a T [Sno].

- *Filtering:* Using a provided relevance slider a relevance threshold can be defined. Only documents with a relevance above this threshold will be displayed.

The user has many other possibilities as to communicate directly with other users via an live audio channel. He can also work on POIs; they can be added, removed or dragged interactively.

Finally, the user can also use traditional search queries to find some documents in the information space. The result of such a query is indicated by highlighting the matching documents. Therefore typical VR-VIBE queries and traditional queries can be used together.

4.2.3 Discussion

VR-VIBE offers the possibility to manipulate the information space dynamically. Nevertheless, it still depends a lot on choosing the right query. The three dimensional representation does little to solve this problem, but offers the user the opportunity to browse and to compare and contrast the result of different queries, or different query positions.

On the one hand the multi-user platform offers some great possibilities, but on the other also some problems. The subjectivity of every user is one of them. The marking of documents is for sure a subjective task. Other users might not want their information space to be cluttered up with other people's comments. The presentation of those text labels in the three dimensional space causes additional problems. The relevance filtering is another subjective issue since every user would assign different relevances.

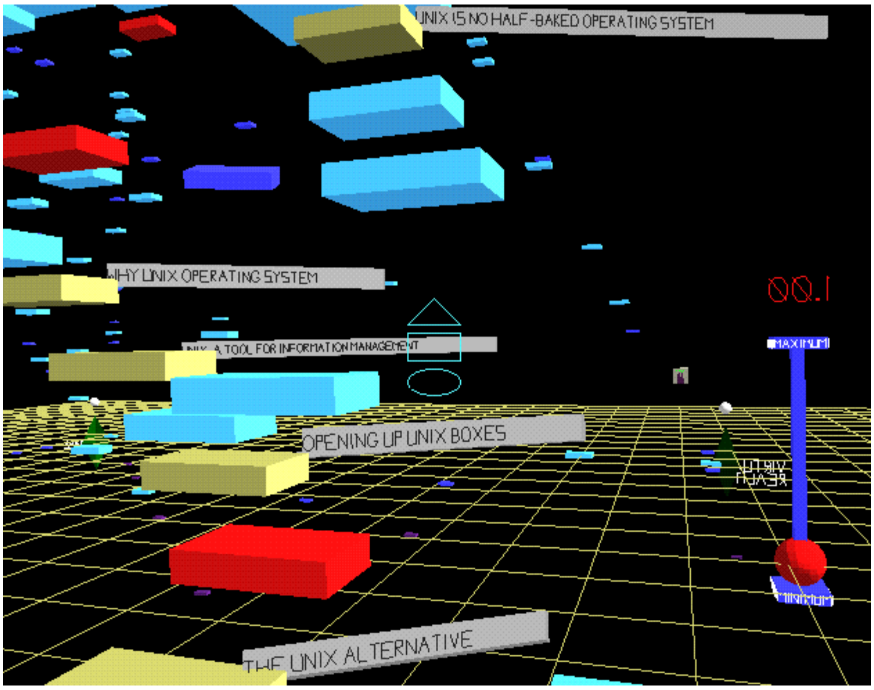

Figure 4.3: A selected group showing the titles of the documents. The icons of this group are marked using a different colour [Sno].

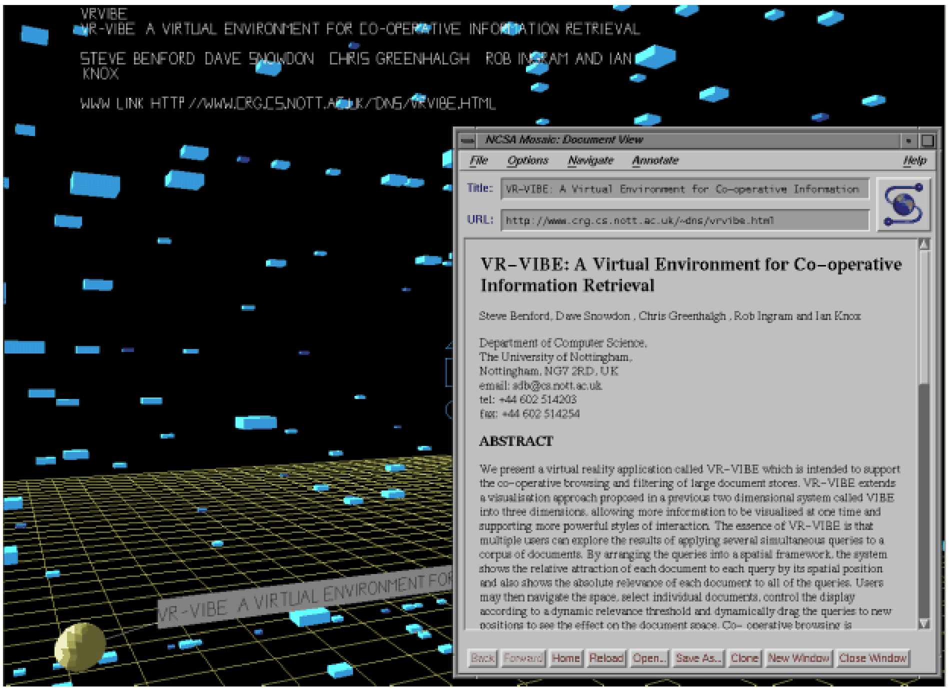

Figure 4.4: A NCSA Mosaic browser is used to inspect a document (in this case a web page) [Sno].

4.3 LyberWorld

The LyberWorld [IPSb, HKW94, Hem93] search visualisation system has been developed at the *German National Research Center for Computer Science (GMD)*. This system is a prototype IR user interface to visualise an abstract information space using fulltext. It offers the possibility to browse in a natural way by applying metaphors of spatial navigation. Two different visualisation tools help the user to find interesting information.

4.3.1 LyberTrees

For the building of LyberTrees some additional information is needed. To help the user to create a new query which narrows down the search result set keywords are made available. By clicking at a document they shown. Naturally those keywords are not filtered out during the computation, but when the document is added to the IR system. They are stored in a database.

The search result set achieved by a search query is often quite inadequate. Out of this results an initial LyberTree is build. This tree is a cone tree, containing all documents found. The documents are represented using labels. In this document cone tree the more relevant documents are presented frontal to the user and the most relevant is indicated using a different colour. To view the less relevant information the user may rotate this cone tree and have a look at all titles.

If some interesting information is found a new cone subtree will be shown by selecting this document. This new tree is a term cone tree, showing the specified keywords for this document. A particular document can be opened using the LyberRoom, a three-dimensional visualisation of a room. There the documents are projected onto the wall. To find similar documents a document may be selected. This causes its keywords to be shown in a term cone tree. Selecting a word starts a new search query and the retrieved documents form again a cone tree, which is added. Naturally the document and keyword subtrees can contain information, which is already provided in an upper tree. To indicate this a special colour is used (orange).

To present a large set of documents the radius of the cone tree is enlarged. This works fine up to a certain number of documents, than a new representation would be more useful. Therefore the items may also be unfolded as a spiral tree. To get a good overview of the cone tree tree zooming is implemented, as shown in Figure 4.5.

To get more relevance information about the document an other tool of LyberWorld is used, the Relevance Sphere.

4.3.2 Relevance Sphere

To get a better relevance feedback and to do document clustering another tool, the Relevance Sphere can be used.

Here the result document set is displayed in the sphere. The query terms represented by small spheres are positioned on the surface. The spatial position of the documents depends on the relevance and similarity. To get an overall relevance, the distance to the surface of the

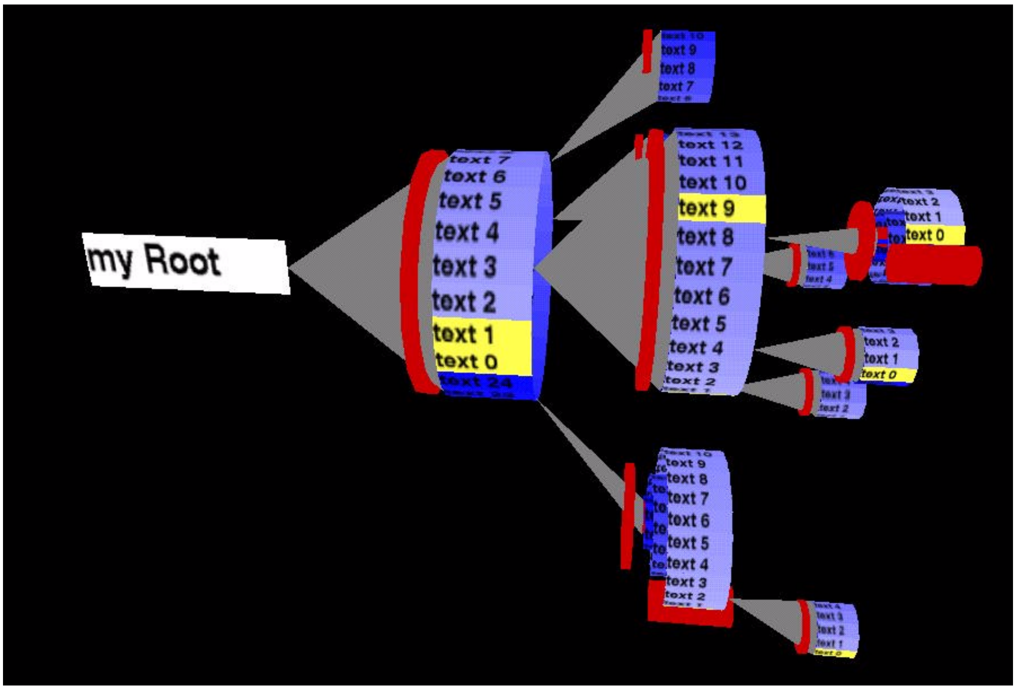

Figure 4.5: A LyberTree showing a complex structure. The document subtree extracted from text 8 is represented as a spiral tree [IPSa].

sphere is of special interest. The smaller the distance, the more relevant is a document. If a document is positioned near a special search term, than this document is especially relevant for this term (Figure 4.6).

To overcome the problem of ambiguous placement, the user can change the position of the query terms on the surface. This causes interactive clustering of documents.

4.3.3 Discussion

LyberTree can be described as a visual history tool. The user can navigate forward and also backwards following the tree. When an area has already been studied this will be recognised immediately. By structuring the information as a tree the cognitive load on the user is decreased.

The Relevance Sphere is quite practical to get an insight into relevances and similarities. Nevertheless some problems appear using this sphere. It may be difficult to judge the overall relevance of individual documents as it depends on the position of the search terms. An experienced user can reach the appropriate result by moving the search terms and assigning suitable weights.

In LyberWorld all tools work together, therefore a switch between Relevance Spheres and LyberTrees can be done easily.

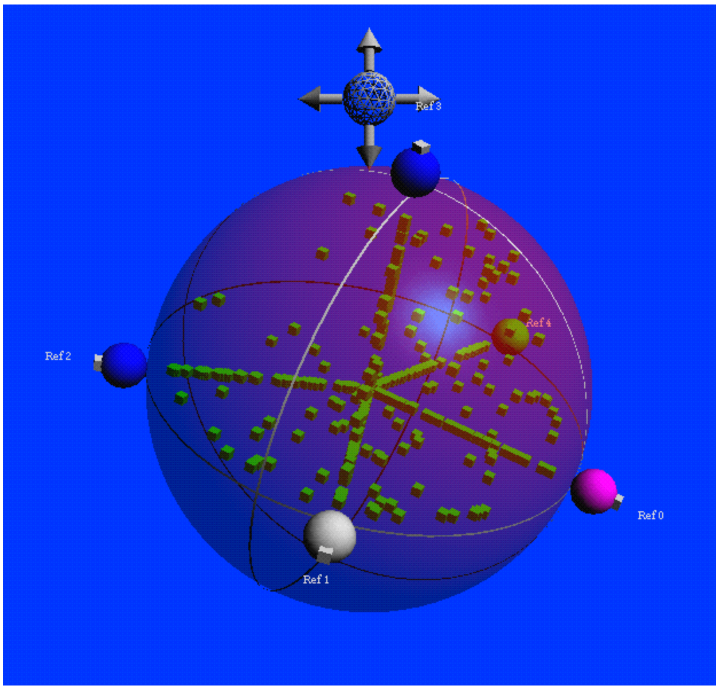

Figure 4.6: A Relevance Sphere with 500 documents and 5 search terms [IPSa].

4.4 InfoCrystal

InfoCrystal [Spo93] addresses the problem of how to visualise abstract information, such as a document space, without explicit spatial properties. As the amount of available information keeps growing, it is necessary to have tools that enable to explore, manipulate and relate large information spaces to the users interests in an interactive way. InfoCrystal visualises all the possible relationships among N concepts.

4.4.1 Visualising Search Results using a Venn Diagram

To visualise all the relationships on a two dimensional screen, the common approach of Venn diagrams is used. To describe this approach, we take three sets. At the beginning, the Venn diagram is exploded into its disjoint subsets. Those subsets are then represented by interior icons whose shapes reflect the number of criteria satisfied by their contents. Finally the subset icons are surrounded by a border area. This border area contains also icons, the criterion icons, that represent the original sets (Figure 4.7). Looking at the final visual representation, the user should intuitively notice the relation of the interior icons and the criterion icons.

The following visual coding principles are used to establish this relation:

- *Shape Coding:* to indicate the number of criteria that the contents associated with an interior icon satisfy; for example: one $\Rightarrow$ circle, two $\Rightarrow$ rectangle, three $\Rightarrow$ triangle, four $\Rightarrow$ square, and so on.

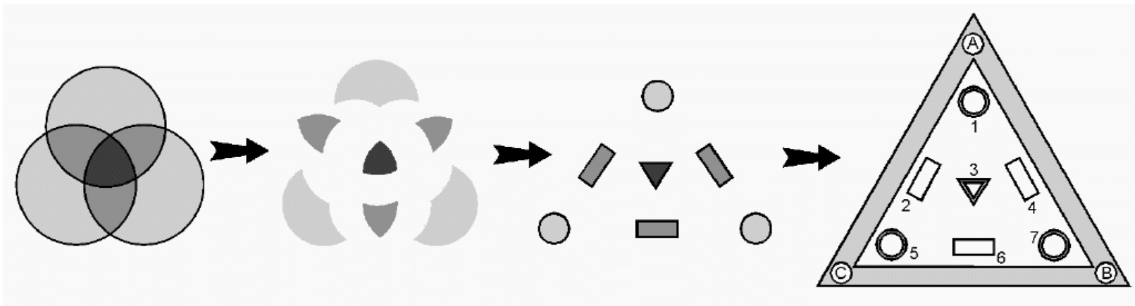

Figure 4.7: The transformation of a Venn Diagram into the interior and criteria icons.

- *Proximity Coding:* The closer an interior icon is located to a criterion icon, the more likely it is that the icon's contents are related to it.
- *Rank Coding:* Icons with the same shape are grouped in *invisible* concentric circles. The rank of an icon is equal to the number of criteria satisfied. The smaller the radius is, the greater is the rank.
- *Colour or Texture Coding:* is used to indicate which particular criteria are satisfied by the icon's content.
- *Orientation Coding:* The icons are positioned so that their sides face the criteria they satisfy.
- *Size or Brightness Coding:* is used to visualise quantitative information like the number of elements represented by an icon.

As a interior icon can contain more than one document, the number of those documents is written into the corresponding icon. This can be seen in Figure 4.8

To get more information about the rank of a document, a second layout style exists. It is called *bull's-eye layout*. It causes the relationships with a higher relevance to be placed closer to the center. To determine the placement of the interior icons a polar transformation is used. Thus, an interior icon is placed closer to those criterion icons that it satisfies than to those it does not.

4.4.2 Visual Querys

InfoCrystal has the desirable property that each of its interior icons represents a distinct Boolean relationship among the input criteria. Therefore they can be used to specify boolean queries (see Section 2.3.1) by selecting a subset of interior icons. That means the InfoCrystal can be combined and organised in a hierarchical structure to create complex Boolean queries. The output of one InfoCrystal (the query designed out of the selected subset of interior icons) is used as input for the InfoCrystal one level up in the query hierarchy. Special about this tree is, that the parents do not only inherit the data elements associated with their children's nodes. Instead there is an intermediary step, where the relationships among the children's nodes are represented by an InfoCrystal. The user can ask *what if* questions by changing

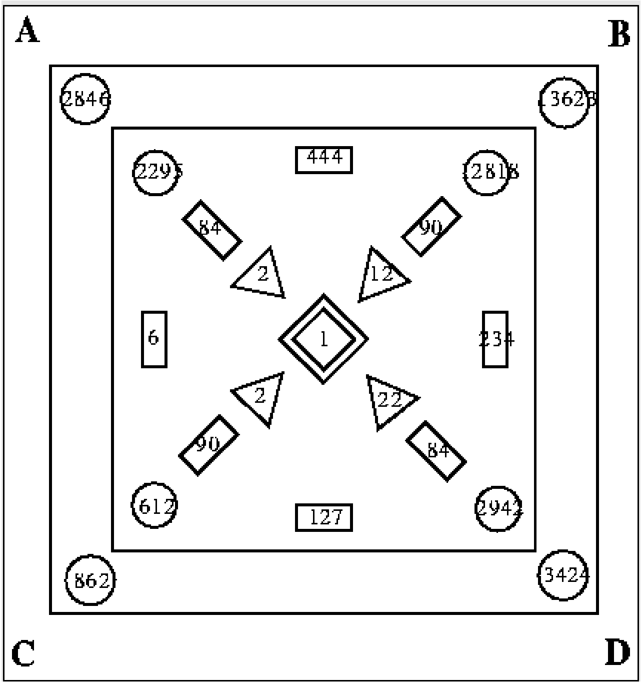

Figure 4.8: The result of he separation of four sets. The numbers inside the interior icons specify the number of documents which are represented by this icon.

which interior icons are selected in one InfoCrystal and observing how the contents of the dependent icons higher up in the hierarchy change dynamically.

In some situations the user will assign relevance weights to their search interests. This can be done by interacting with a slider and choosing a value between minus one and one. Negative weights indicate that the user is *not* interested in documents that contain this concept. The weight is equivalent to the logical NOT. By interacting with another slider, the threshold slider, users can select only the interior icons whose relevance score is above the threshold.

4.4.3 Concluding Remarks

InfoCrystal is designed to visualise the similarity and the possible multiple relationships between the contents of an information space and N search criteria. Further, the user is allowed to specify Boolean (see Section 2.3.1) as well as vector-space queries (see Section 2.4) graphically. Keep attention, complex queries might need too much screen space to be effective, even if a hierarchical structure is used.

4.5 TileBars

The evaluation of overall relevance to rank a search result list is often problematic or even wrong. The longer a document is, the more unlikely the calculated relevance will agree with

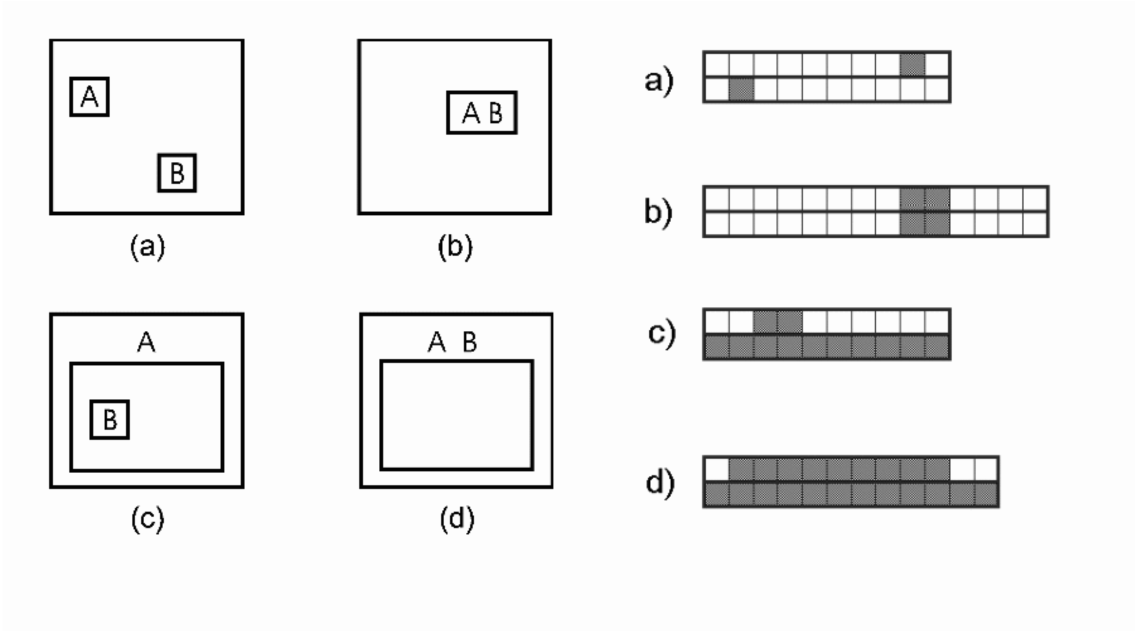

Figure 4.9: The distribution of two terms in a document, created with TileBars [Ber].

the relevance a user would assign.

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

4.5.1 Document Structure and Representation

Lets have a closer look to the problem of relevance evaluation. For rather small documents, the frequency of a query term in the document compared to the number of terms in the document can be regarded as an equivalent to the relevance. Looking at large documents, the situation changes. The query terms might appear quite often, but as the document is quite large, the term will get a small relevance value. If the document consists of sections, it would be more reasonable to have a look at those section and subsection, as those might deal with completely different topics. A query term might occur frequently in one section and hardly in another. Inside one document, looking at two terms, the following term distributions can occur: the terms are disjoint, that means handled in different sections; both terms are discussed in the same section; one term occurs in the whole text the other term only in one section; both terms occur globally.

To subdivide a document into subtopic, an algorithm called Text Tiling is used. This algorithm determines the internal structure of a text document. It does so by analysing the term repetition pattern withing the text to detect the boundaries of the subtopics.

To represent the location of term sets inside a document the TileBars display simultaneously shows:

1. the *relative length* of the document

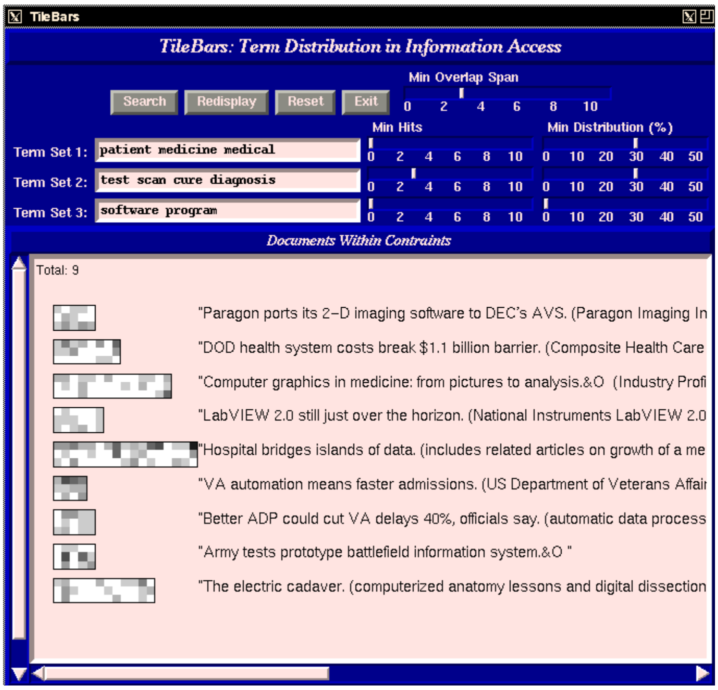

Figure 4.10: A TileBar Application [Ber]

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

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

The relative length of the document is indicated by the length of the corresponding TileBars representation. To be able to compare the size of different documents, the TileBars are left aligned.

To present the term distribution each document icon within TileBars is subdivided into many non-overlapping squares, or TextTiles. Each square corresponds to one subtopic. These TextTiles offer also the possibility to show the frequency of a term. The more often a query term occurs in a particular section, the darker the TextTile gets. To combine the different query terms in one term set a logical OR is taken. Since several term sets can be made at the same time, those can be combined with an implicit AND.

4.5.2 Discussion

TileBars have a different approach to visualise search results. The document itself is not displayed in a graphical way, but a member of a list together with the graphical representation of the relevance. The user can browse all the documents by scrolling the list and estimate the personal relevance of each document before viewing it. Not only interesting documents, but also the relevant topic can be accessed directly.

As TileBars are represented as a ranked list, the common disadvantages appear. There is only a small part of the whole search result visible. No general overview for finding related documents is supported.

4.6 Envision

Envision [NFH+96] provides powerful information visualisation by displaying search results as a matrix of icons. With the aid of these icons a unique variety of document characteristics, but also ranked query-document similarity can be represented. The layout semantics is under user control. To make all the users' wishes available, an extensive range of user tasks is served by this system. To create a real user-centered product interviews and user task analysis have been made. The following principles were discovered to be important:

- 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 one another's work to form on ongoing conversation in print.

4.6.1 Design of Envision

The window containing the search results display can be divided into three main parts:

1. the *Query Window:* to create a new query or to edit an existing one
2. the *Graphic View Window:* contains the graphical representation of the search result, the icon matrix
3. the *Item Summary Window:* provides bibliographic information in form of a list, corresponding to some marked icons in the Graphic View Window.

The query window represents the search interface of Envision. It provides the possibility to search for authors, words in title and content words. Each search facility has its own input-field. By starting the search, the contents of these fields are connected to a query. Another feature of this search interface is the possibility to edit and resend old queries. Therefore a Query History is provided.

The actual visualisation is done in the graphic view window. This window consists of a matrix of icons in the main part and some adjustment facilities and a legend in the upper part. The user can change all the semantics to get his preferred representation. That means the user can not only change the icon attributes as size, colour or shape but also the semantics of the axis. Therefore every possible combination of document characteristics can be made. That means an axis can represent estimated relevance, author names, publication years, index terms or document type. According to the axis the icons are placed at their position in the matrix. Up to now, the icon design may indicate:

- icon size: may be uniform or may vary to indicate estimated relevance.
- icon colour: may be uniform or may vary to indicate relevance or document type.

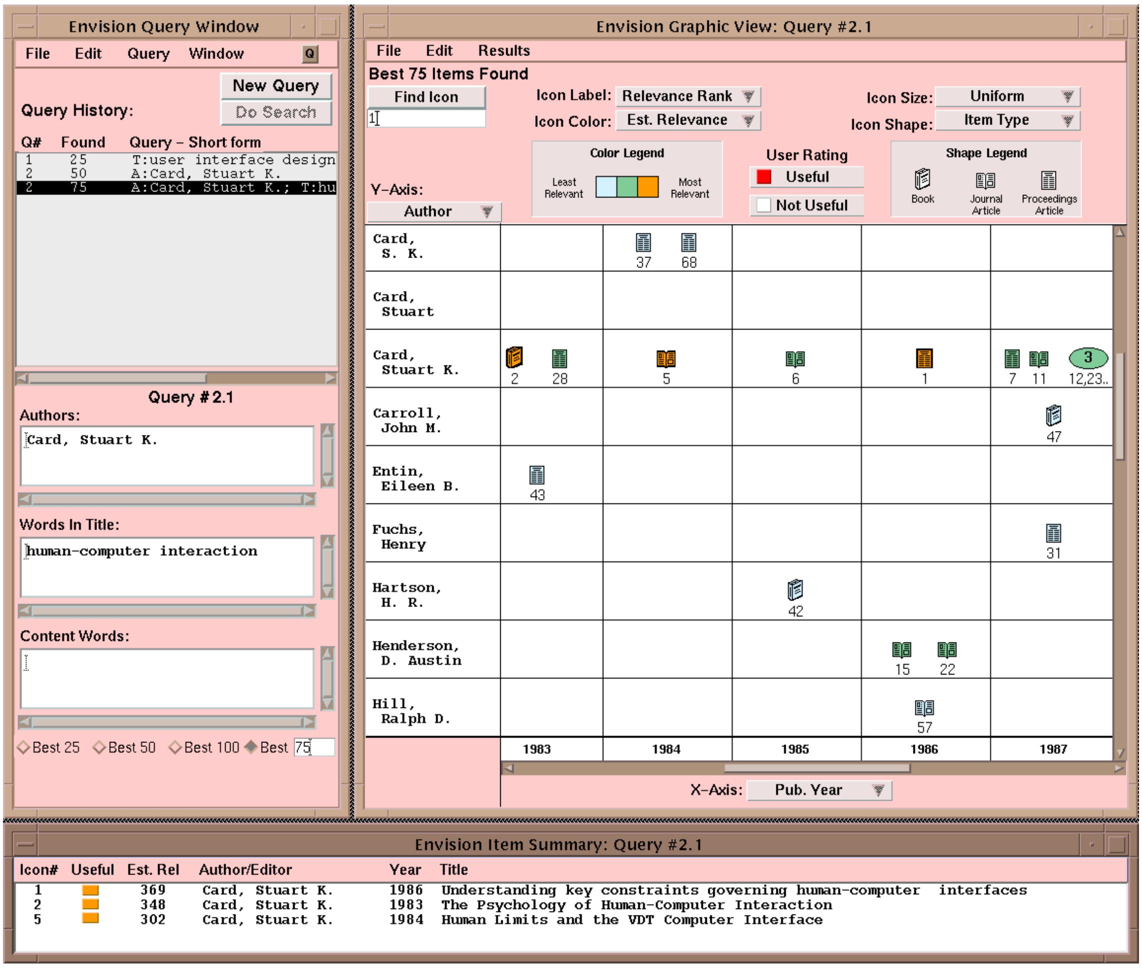

Figure 4.11: Envision, showing all three windows, the Query Window, the Graphic View Window and the Item Summary Window.

- icon shape: may be uniform or may vary to indicate document type.
- label associated with each icon: may signify either the relevance rank of the document or a unique Envision document identifier.

The scales of the axis depend on the chosen document attribute. As the size of a cell is limited, some requirements for the layout were necessary. The most relevant document should be the first icon in the cell. The layout should be uncluttered, making effective use of the cell. If icons would overlap, they are combined to an elliptical cluster icon. The number of documents represented is shown in the cluster icon. A label shows the ranked identifiers of the two most important documents. The colour of a cluster corresponds to the colour of its highest ranked document.

Marked icons are indicated by a surrounding box, selected icons by using bold icon labels. The document corresponding to a selected icon is shown in the item summary window. A double-click on an icon in the graphic view window or a line in the item summary window will start XMosaic to access that document.

4.6.2 Discussion

Envision is a very flexible visualisation system, which can show a lot of different document characteristics. The displayed result in the graphic view window is clearly arranged and self-explaining. Envision is specialised for the use with digital libraries.

One problem of this system is a naming problem. The so called cluster icons may mislead the user, since clusters in information visualisation are normally a set of similar documents. Here the documents in the cluster icon may share only the publication date and the author. Relationships between documents are not visualised, as the axis can only represent document attributes. Therefore the results of the full text retrieval (content words) cannot be seen.

Another problem of Envision is that one document may belong to multiple categories on both axis. That happens because a single document may correspond to several index terms and several authors. Some further work to present these multi-sets without clutter and to realise further document characteristics will be done.

4.7 HarSearch Similarity Map

The HarSearch Similarity Map [Rod97] is a tool implemented for the Hyper-G (former version of Hyperwave, see section) client Harmony. This search visualisation system especially concentrates on similarity display and cluster evaluation. The relations between similar documents in the search result are shown. HarSearch Similarity Map uses trees to show the documents and their similarities.

4.7.1 The Concept of HarSearch Similarity Map

HarSearch Similarity Map shows the relation of each document to the document in the search result set, which is most similar. The similarities between documents are provided by a *similarity triangle*.

To find the relation which should be present in the resulting tree a complete graph $G = (V, E)$ is build. In this graph the vertices represent the documents of the search result. The edges can be interpreted as weighted relations, that means similarities, between the specific vertices.

As only the most similar document shall be connected, an integration step has to be done. An acyclic subset $T \subseteq E$ connecting all vertices and having the maximum weight will be build.

$$w(T) = \sum (u, v) \in T w(u, v)$$

The resulting subgraph is the maximum spanning tree that represents the HarSearch Similarity Map. The maximum spanning tree is an equivalent to the well-known minimum spanning tree, it can be grown using Prim's Algorithm .

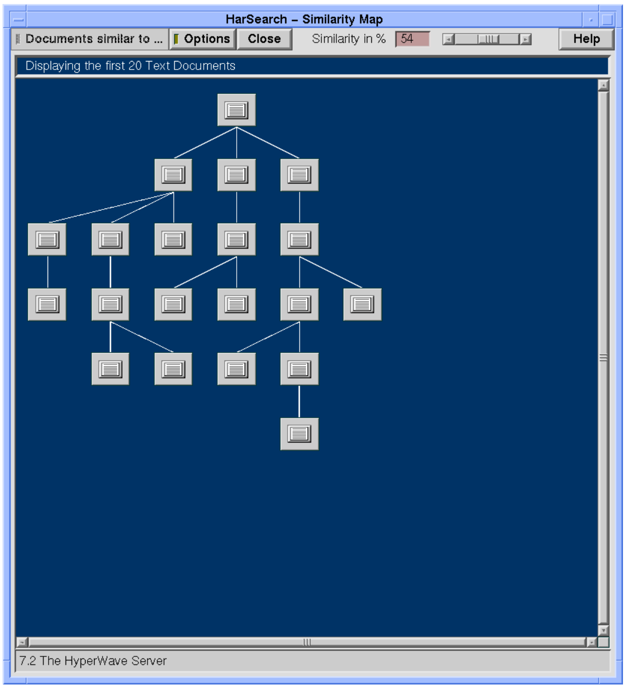

Figure 4.12: A one-sided vertical HarSearch Layout.

4.7.1.1 Layouts in HarSearch Similarity Map

To visualise this maximum similarity spanning tree, HarSearch Similarity Map provides several tree layouts. Some of them are quite common for computer users, but as they have some disadvantages, other layouts have also been realised.

- *One-sided tree layout (vertical or horizontal):* for example the vertical layout: in this tree, the root is placed at the top of the tree and all its children are positioned in the level below. The children of the children are again placed one level deeper, ... Have a look ar Figure 4.12.
- *Two-sided tree layout (vertical or horizontal):* in a vertical two-sided tree the root is not placed at the top of the tree, but in the center. The children are split into two groups. One group is placed above the root, the other below the root. The children of the lower group are positioned one level deeper, the ones of the upper group are positioned one level higher. This can be seen in Figure 4.13.
- *Polar coordinates layout:* in this layout a one-sided tree is transformed into its polar coordinates. Now a radial level drawing can be done, this is shown in Figure 4.14.

4.7.2 Clustering in HarSearch Similarity Map

Clustering can be done quite easily using HarSearch Similarity Map. This system knows all the similarities between the document. In the maximum similarity spanning tree, all documents are connected with their most similar neighbour(s) as described in the last section.

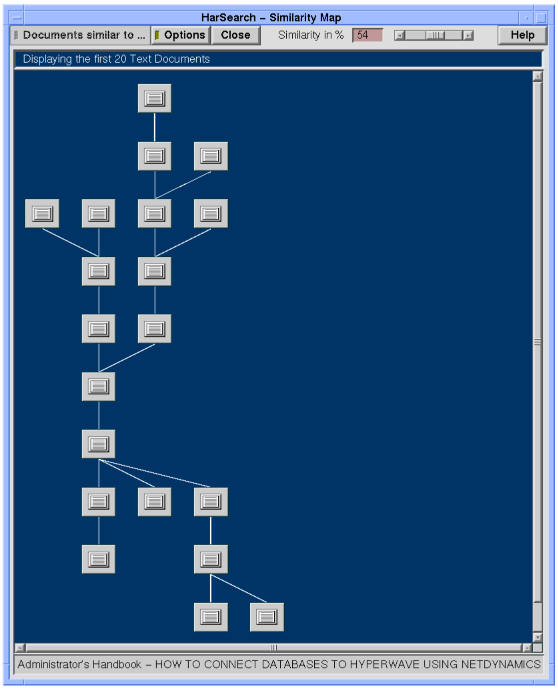

Figure 4.13: A two-sided vertical HarSearch Layout.

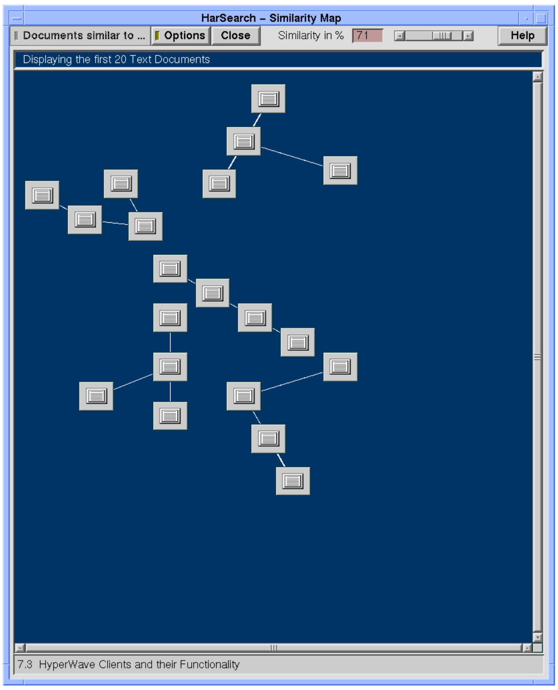

Figure 4.14: A HarSearch polar coordinate layout with clustering.

To generate clusters a *similarity slider* is used. This slider can take any value in between zero and one hundred. The actual slider value is taken as a threshold. Every edge in the tree representing a smaller similarity is not shown any more. Using this approach, the maximum similarity spanning tree disintegrates into several smaller trees. Those subtrees are the desired cluster.

4.7.3 Discussion

HarSearch Similarity Map offers several layout possibilities to reduce the cognitive load on the user as documents are connected according to their similarity. A path for exploration through the search result set is suggested.

Unfortunately HarSearch Similarity Map could not be tested with real similarity values. As this system is a client of Hyperwave, these values have to be computed via the server's full text index. However this facility is not implemented up to now. Therefore some questions remain without an answer like possible consequences from real similarity values and also the usability of the similarity map.

4.8 Bead

Bead is a prototype for the graphically based exploration of information [CC92]. The general information base of this system is a bibliography. The aim of Bead is to retrieve information more graphically by using everyday phenomena as spatial position and colour.

Each document is represented as a particle in three dimensional space in Bead. To find out relations between the documents forces according to the similarities are generated. Depending on the similarity and the distance of two particular document a force is created. This force is either an attractive one, to pull similar documents closer together or a repulsive one to move dissimilar documents apart. Using this forces patter in the document space as cluster of similar documents can be found.

4.8.1 Model

Each document in Bead is represented as a vector of n terms. The value of one term in the vector can be zero to indicate absence or one to indicate presence. Another possibility is a value between zero and one, representing a normalised weight(see Section 2.4). Because of this representation of documents similarity measurement can be done easily using the cosine correlation (see Section 2.4.1).

The generated similarities are used as input for the underlaying physical model. Here the documents are regarded as particles with a specific mass and the similarities are used to define some springs between these particles. If two particles are close, but not similar the spring force pushes them apart, else they draw them towards each other. The main goal of these springs is to reach a stable state with minimum energy (see Section 3.1). This state shows structures and pattern in the modelled data.

Bead is a part of the well-known N-body problem. That means for all N particles (N-1) pairwise interaction have to be done. The complexity of this approach is proportional to N^2. To get a lower order of complexity approximate solutions have been developed. In Bead a three dimensional tree is used to subdivide the particle space. Each node in the tree is called a metaparticle. It consists of a particle position, a mass and a term vector. Document distances to metaparticles are used to recursively descent the tree in an effort to insert a new document near to similar documents. The maximum number of documents for each leaf node is set to ten. As soon as this limit is reached the leaf is split. In attempt to minimise the unbalanced force on each particle a iterative fourth-order Runge-Kutta algorithm is applied.

To measure the progress of force using approaches usually the maximum unbalanced force on any particle in the system is taken. Another possible evaluation can be done analogue to the mechanical stress of the spring system. In this case the geometric and the document distance is compared. The ideal value for this stress value is zero.

As the development of the Bead system is still ongoing, a new approach for reducing the complexity has been realised. This algorithm reaches a linear complexity by using two lists. A neighbour list, which is assigned to each particle, containing the recent neighbours and a random list to calculate the forces to some random particles who are not neighbours (see Section 3.3).

4.8.2 Layouts

The outcome of Bead is a three dimensional point cloud, containing all documents in the information space. Now this result has to be presented in a form which allows the user to build an intuitive mental model. The first approach of a visualisation was the grid viewer. Later a landscape metaphor was realised.

4.8.2.1 Grid Viewer

This viewer shows the three dimensional point cloud using a two dimensional visualisation. The grid viewer shows three orthogonal plots (in XY, XZ and YZ) as well as an animated perspective view. Each document is represented by its id number.

Grid viewer offers some features as the possibility to make the distribution of a chosen set of query words displayable. This is done by assigning a colour to each particle according to its document distance from the query.

To get a better overview the user can zoom in and zoom out on a chosen particle. As all particles are visualised a *sphere of interest* with an adjustable radius is defined. The center of this sphere is a chosen particle. All particles which are positioned outside this sphere have their colour slightly reduced in intensity.

4.8.2.2 Landscape Metaphor

The landscape metaphor [Cha93] accepts a loss of exactitude of relative distances in three dimensions to increase the accessibility and the familiarity by using a 2.1 dimensional landscape representation. This decreases the cognitive load on the user.

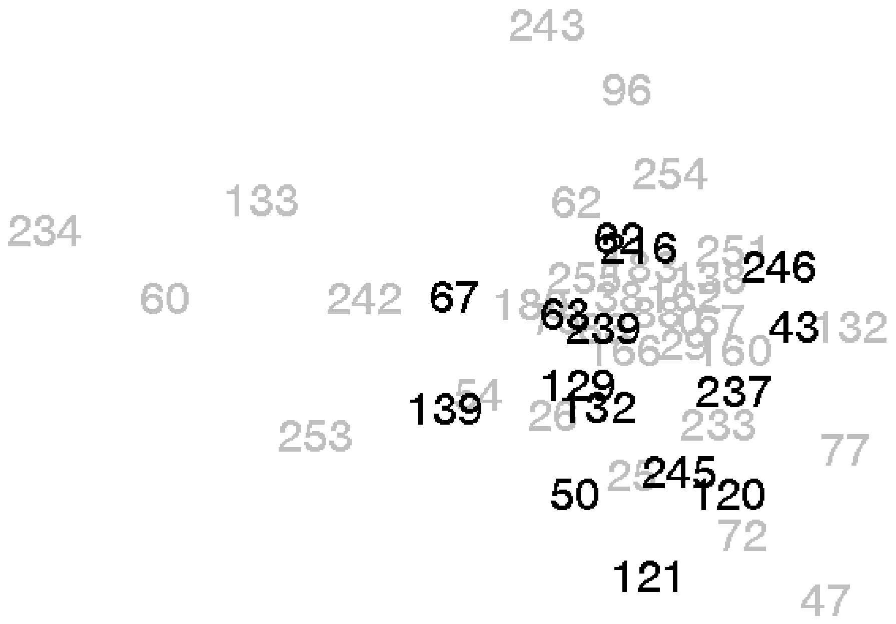

Figure 4.15: A point cloud visualisation [Blu96]

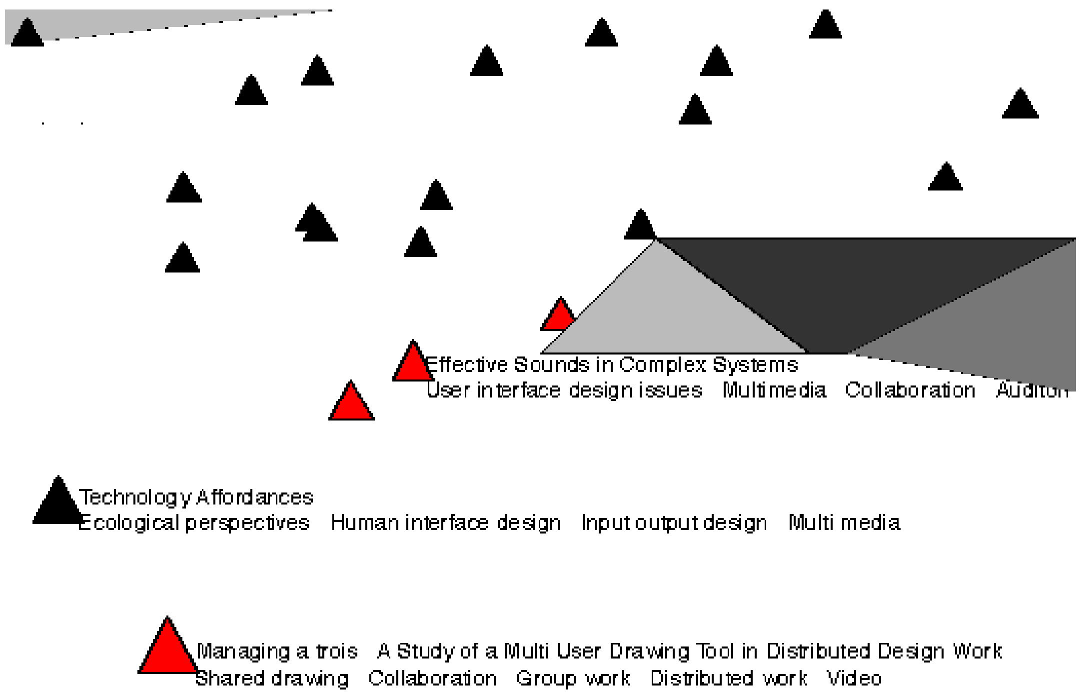

Figure 4.16: A zoom in a Bead landscape [Blu96]

To introduce some useful characteristics, features such as landmarks and borders were generated. The document position show still local consistency of themes and topics. Individual documents are represented as coloured markers within the landscape.

The modelling process can lead to peaks and valleys. They serve as natural reference points (landmarks) to ease the orientation and navigation. The peaks and valleys represent areas where the *fit* of documents is rough. This can be interpreted in two ways, either the system just could not find a good layout, or no planar layout exists for these documents.

Landscape offers several activities, the user can move around freely to find some interesting documents. The possibility to zoom in and out is provided and again queries are indicated by changing the colour of the relevant documents.

4.8.3 Concluding Remarks

Bead offers a means for providing a *naturalistic* model of document relationships. The interactive visualisation and navigation of the space allows to browse and explore the corpus. The results of searches for keywords are represented by different colours.

The two different layouts offer some different possibilities. Grid viewers are more exact, but as this surrounding is not a used one, the user has some problems to orientate and remember certain formations. The landscape metaphor gives up the exactness to gain a perspective visualisation which can be understood intuitively by the user.

Further work on Bead is still done do improve the model and the visualisation layouts.

Chapter 5

Internet - Hyperwave - Dino

This chapter focuses on the main concepts of the Internet. Furthermore some hypermedia document managing systems are presented. A new application called distributed interactive networking objects (DINO) a basis for communicating with hypermedia document managing systems on the internet will be discussed.

5.1 Internet

The development of the Internet started in the 1969 when the *US Department of Defense Advanced Research Projects Agency (ARPA)* created an experimental network called ARPANET. This net originally connected four universities and enabled scientists to share information and resources across long distances. ARPANET continued to expand, connecting many more sites throughout the 1970s the 1980s. Naturally the ARPANET was used mainly for military purposes. An important development that grew out of ARPANET was the TCP/IP (TCP... Transmission Control Protocol; IP... Internet Protocol). This decision caused a standard for other networks, and generated the use of the term *Internet* as a network of networks all using or interacting with the TCP/IP protocols.

Other significant events included the introduction of the desktop computer, the development of networking tools such as telnet, a File Transfer Protocol (FTP), gopher and WWW, and the release of graphical browsers.

In the following some of the main features of the Internet will be described in more details.

5.1.1 TCP/IP

The TCP/IP (Transmission Control Protocol/Internet Protocol) stands for a collection of protocols an rules that manage the transport of data from one machine to another across the network. IP is the part, which is responsible for the actual, physical travel through the net; TCP works upon IP, giving the necessary information and data parts to IP. Compared with the ISO/OSI seven layer model, we could say that IP corresponds to layer 3 and TCP corresponds to layer 4 and 5.

5.1.1.1 TCP

The TCP builds a virtual connection between the machines.

At the sender TCP breaks the message into datagrams. Each datagram gets a header containing at least 20 bytes. The average size of such a datagram is 500 bytes. In the header *port numbers* of the sender and the receiver, to keep track of different conversations, can be found. Also a *sequence number* indicating which part of the whole message is represented by this datagram. This number is used to make sure that the receiver gets the datagrams in the right order. At the end of the header there is another important part of the TCP packet, the checksum. The checksum is used to ensure the correctness of the datagram.

The receiver TCP checks the arriving datagrams according their sequence numbers and their checksums. If no error occurred the the original data gets reconstructed.
Each of the TCP packets, including the Internet address of the receiver, is sent to IP.

5.1.1.2 IP

The main service of IP is to find a route for the data and to get it to the other end. IP receives the TCP packets and builds its own package containing some TCP packets and an IP header. The main things in this header are the source and destination Internet addresses, the protocol number and the checksum. The protocol number tells IP at the other end to send the datagrams to TCP. The IP checksum serves to verify that the header did not get damaged during transmission.

IP Addresses

Every machine connected to the internet needs its own unique address to be able to communicate with others. Such an IP address is a 32-bit number normally written as 4 decimal 8-bit numbers with a point in between (for example 129.27.2.3). A network administrator who wants to connect some computers to the Internet does not get one number for each, but an address range for the whole subnet. Depending on the size of the subnet, this range can be one of five different classes (Table 5.1).

These addresses are managed by the Internet Assigned Number Authority (IANA). They assign port numbers and delegate blocks of addresses to Internet Network Information Center (InterNIC). InterNICs handle all the requests for new numbers.

To navigate through the Internet, complex machines are needed to store the information about the IP Number and the physical place of the subnet it is assigned to. Such machines are called *routers* because they also try to route datagrams in the right direction. As there are many possible ways to go. The router tries to choose the fastest and most secure one. The circumstances can be very dynamic, therefore packets of the same transfer can take different ways (depending on network traffic,...).

Domain Names

Internet addresses are difficult to remember, that is why each computer is also assigned an Internet name. These names are usually structured hierarchically. Every part of the name

class	*8-byte*	*8-byte*	*8-byte*	*8-byte*	*number of PCs*	*comments*
A	fixed (1 - 127)	free	free	free	16.387.064	
B	fixed (128 - 191)	fixed (1 - 255)	free	free	64.516	
C	fixed (192 - 223)	fixed (1 - 255)	fixed (1 - 255)	free	254	
D	fixed (224 - 239)	fixed (1 - 255)	fixed (1 - 255)	free	254	special use
E	fixed (240 - 255)	fixed (1 - 255)	fixed (1 - 255)	free	254	special use

Table 5.1: The different types of network classes.

tells in more detail where the machine is located. For example the computer with the name: fiicmal03.tu-graz.ac.at, the analysis is started on the right side:

- *at* means the domain Austria
- *ac* means domain academic
- *tu-graz* means domain Graz, University of Technology
- *fiicmal03* is the name of a local DEC Alpha Workstation at the IICM

Another advantage of the names is, that if a machine or service is moved, it will probably keep its name, but get a different address.

Domain Name Service

There is some service needed to handle the relation between the IP addresses and the names. This can be done by a domain name service. If somebody wants to send some data over the Internet specifying the destination by name, the local domain name server is the first to react. He checks, whether the target belongs to the same domain. In this case, the address has to be found in the lookup tables of this server. As next step, the cache is consulted, looking for recently used names. As final step, if no address has be found, the server of the top level domain is asked for the address of the next lower domain, and so on down the name.

5.1.2 Networking Tools

In the Internet we have a lot of applications, which help us to perform special activities. As these tools use the net they can be summarised as networking tools. They provide possibilities to connect to other computers, which may be situated in another net. Another tool offers an easy way to transfer files.

5.1.2.1 Telnet

Telnet is a protocol, that enables one computer to connect to another. This process is called remote login. The remote computer, the machine to which we want to connect, may be at any place, in the next room, or even in another country. Once connected, the remote computer is emulated on the local computer. When the user types in some commandos, they are executed on the remote computer, but the display is set to the local one.

Telnet can provide access to many resources around the world such as library catalogues, databases, and other Internet tools and applications.

5.1.2.2 FTP

FTP is a part of the TCP/IP protocol suit. It is a protocol which enables the transfer of files between networked computers. When an FTP connection is opened, the files can be transferred in both directions. To access files that can be transferred an FTP client program is used. This is an interface that allows to locate the file(s) to be transferred and initiate the transfer process. Anonymous FTP allows a user to access a wealth of public available information including shareware, freeware, upgrades and documents.

5.1.2.3 E-MAIL

Electronic mail, or e-mail, allows information to be sent and received through the Internet. It is the most widely used Internet resource. A client program is used to read mail, replay to mail and send new messages. Those messages can contain text as well as other file types which are encoded and decoded by MIME.

The connection type of e-mail is opposite to the type of telnet or ftp. Here no point-to-point connection is provided. This means, that a message is stored and forwarded. Therefore no connection is kept until the message arrived.

5.2 Hypertext and Hypermedia

Hypertext can be described as non-linear text. Normal text is linear, the pages can only be read in a sequential order. That is fine for books or magazines, but the Internet provides more possibilities. Therefore hypertext offers a way to jump from a part in a document to another part or even to another document. This jumps are done by following links between chunks of text. The term hypertext (hyper for the branches and text for the normal text) was coined by Ted Nelson in 1965.

For more general purposes the term hypermedia was build representing the combination of hypertext and multimedia. In this case not only text can be accessed non-sequential, but also other documents as images, audio clips, video clips, ... can be used.

A new terminology to describe this model was found:

- document or node: is an individual chunk of information which can be an

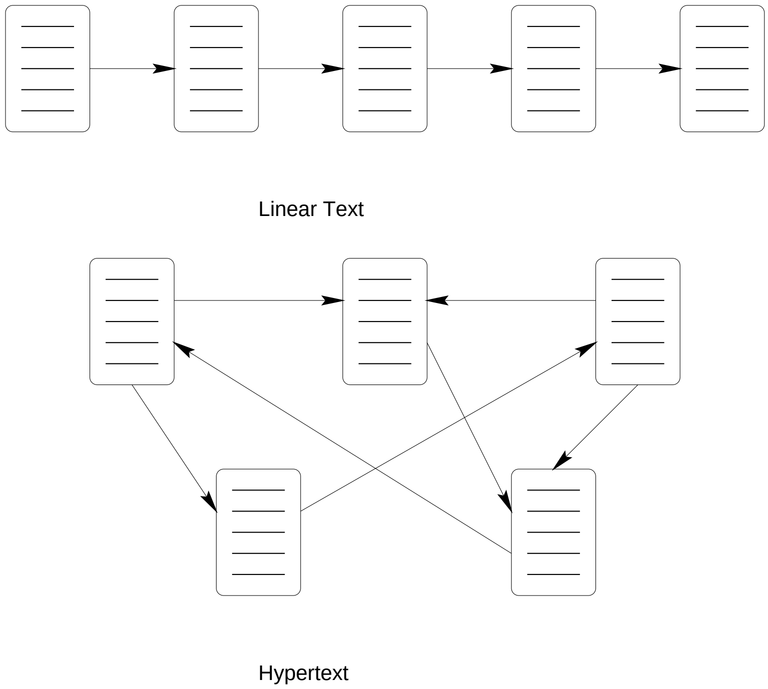

Figure 5.1: Linear Text and Hypertext [And94]

- anchor: is the starting point or the destination of a
- link or hyperlink: the connection between two anchors
- Hyperdocument: is a distinct set of nodes and links which constitute a logical entry
- Web or Hyperweb: is a distinct set of hyperlinks

Links

When we see some highlighted part in the text, this might be the anchor of an link. Anchors may consist of the whole node, or just a small passage inside a node as a character sequence, some button, ... To follow the link only one mouse click is needed. Prior to link activation, some attributes of the link and destination document should be shown. As it is not always easy to see where the links leads to. To help the user avoid known documents some check-marks may exist.
There exist two methods to store links,

1. embedded in documents: This is the most common used method, because it is easy to implement, scales well in networked environment and can be edited in place. On the contrary only unidirectional links are supported. Another disadvantage is that links cannot emanate from read only or multimedia documents. No automatic link maintenance can be provided, therefore this method may lead to dangling links. It is also hard to implement link features as access rights using this kind of link storage.

2. in a separate database: used in some well-known hypermedia systems as Hyperwave which will be discussed in the next section. Here some additional tools are necessary. For example an extra database to save the links, some link editing tool and a notification mechanism. But this efforts cause, that all the problems of embedded links can be handled in an easy way.

Problems in Hypermedia Systems

Navigating in hyperspace can leave to a situation called *lost in hyperspace*. Following links is easy, but to find the way back to the starting point and to keep track of the original orientation can be very difficult. Another problem is that in most cases links only point in one direction, and therefore going back to the previous document is not possible if there is not provided some help. It is also not known how much information exits on a subject, or how much has already been seen. Some solutions to avoid this problems are

- guided tours are used to lead the user on a predefined path through hyperspace. It is a linear walkthrough with a limited number of branches. Another offered possibility is to leave the tour and explore, but the guide waits in case the user returns.
- search facilities return all documents found on a certain subject. Several different ways to look for are offered as attribute search and full text search.
- backtracking allows to return to the previous mode with the aid of some tools like visual cache (that are miniatures on the screen), history lists, history trees,...

Assuming that the links are embedded another problem comes up. As the URLs statically addresses a certain object it may happen, that the URL points to nowhere as the document has been moved or deleted (dangling links). The URL could also point so a completely different object. In case that a document is transferred to another place all documents which contain a link to that particular document have to be updated.

A version control is missing - readers have now problem to find any new information on the net. But what happens if some old document, which has already been visited by the user, changes? The page stays marked as if it was read, although this is not any more true. Another problem is to decide, which information can be deleted as it is not any more up to date.

5.3 Hyperwave

Hyperwave, formerly called Hyper-G, is a large-scale, multi-user, distributed, structured hypermedia information system [Mau96]. It is developed at the Institute for Information Processing and Computer Supported New Media at Graz University of Technology. It can be called a next generation hypermedia system since it combines all advantages of the other systems as gopher, WAIS or WWW (World Wide Web) and supports some additional features which overcome several problems of those.

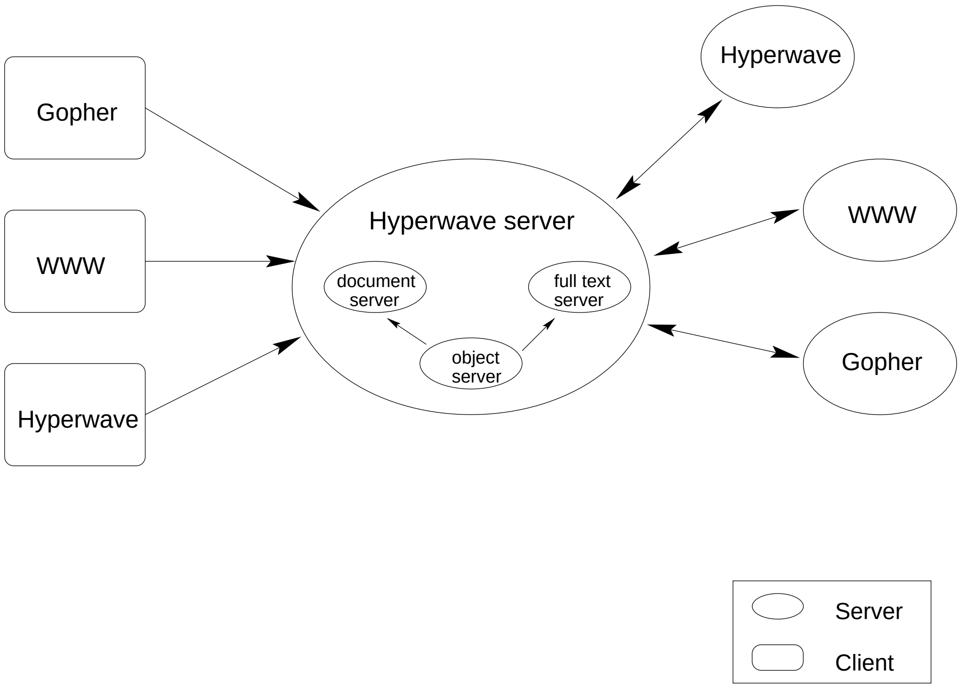

Figure 5.2: The architecture of Hyperwave. [And94, page 81]

5.3.1 The Hyperwave Server

The Hyperwave server architecture provides and manages large scale hypermedia information bases by using a multi-user concept. Hyperwave also offers gateways to other existing information systems like WWW (World Wide Web) and Gopher. That means the server can be accessed by those clients, but it can also access server of those systems. Nevertheless in most cases Hyperwave will be accessed using some Web browser as Netscape or the Microsoft Internet Explorer. In this case Hyperwave reacts like a *standard* Web server and offers some buttons at the top of the page to access the additional functionality.

The Hyperwave server can be divided into four main parts:

1. *Document Database:* here the documents are stored and served to the client. Another task of the document base is to fetch and cache remote documents. Each object has a unique ID independent of the physical location. This ID consists of two parts, the birthplace ID (32-bits) and the object ID (32-bits). When an object moves the birthplace server maintains a link to the new location. Documents and collections can also be given one or more symbolic names.

2. *Link Database:* this server maintains links and attributes. The link database is also the reason some other nice features as the possibility to follow the links backwards (links are bidirectional), local maps and link consistency. As the link is not embedded, real hypermedia documents such as images, audio-clips or video-clips can have links emanating from them. Links may also be inserted interactively.

3. *Structure Information:* the hyperspace can be structured by using some facilities to combine individual documents. One combination is

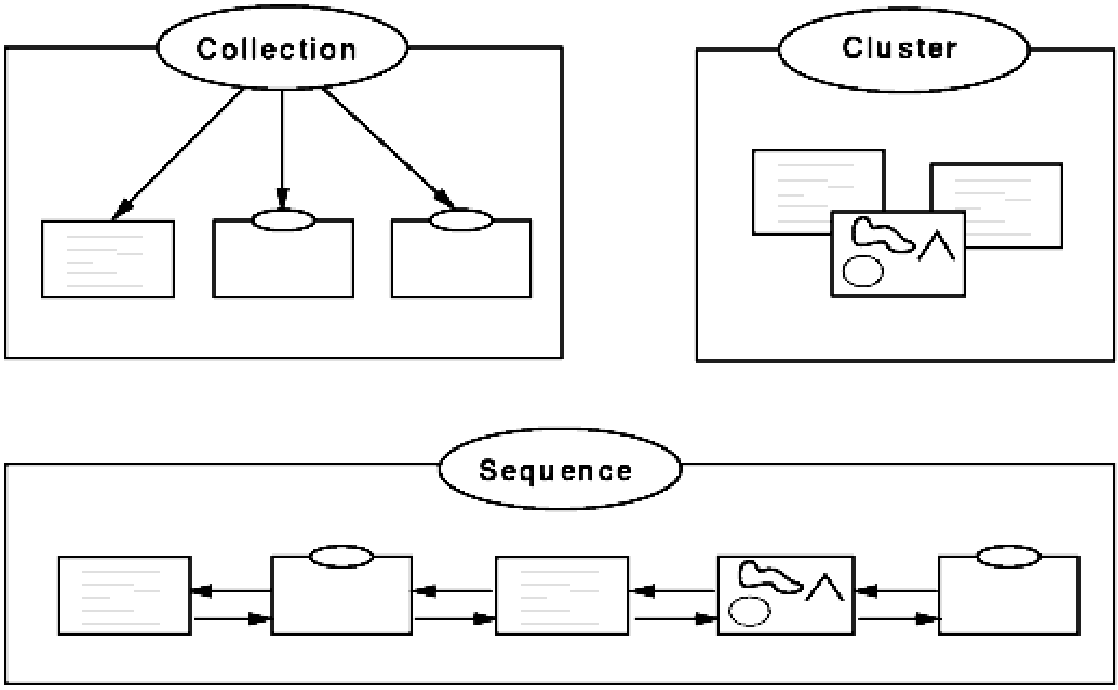

Figure 5.3: The different structuring elements of Hyperwave. [Mau96, page 108]

- a *collection:* that is a set of other collections and documents (also across server boundaries). Another possibility is
- a *cluster:* Two kinds of differents clusters can be distinguished:
  - multimedia cluster: documents of different media are presented simultaneously
  - multilingual cluster: documents in multiple languages; the document with the most suitable language is selected dynamically
- sequence: here the documents are presented in a fixed sequence. The user can navigate using the next and previous buttons. If a document is added or removed, the connections are updated automatically.

4. *Search Index:* Hyperwave server have built in search facilities. Two different kinds to search for information are integrated, which can be used simultaneously.

   - attribute search: here certain meta-informations like author, time the document was last modified or keywords can be looked for.
   - full text search: offers the possibility to search for terms in actual text of text documents.

   Both search facilities are stored in an inverted index (see Section 2.3.1) and maintained incrementally. Therefore it is possible to search through them quickly.

   One special opportunity is, that not the entire local server has to be searched through. The user can specify a certain collection to be the information base. If a native Hyper-

G client is used there exists another possibility, a selected group of collections can be used as information base.

Hyperwave Client/Server Connection

The connection port depends on the client type. As Hyperwave is a web server the default client is Netscape or an IE (Internet Explorer). These clients are connected via port number 80. Nevertheless Hyperwave still offers a native Hyper-G client to connect. In this case the used port number is 418.

The client tries to open a connection to the server by starting a TCP connection on the appropriate port. When a connection is established, the protocol negotiates on the available format. There exist binary and ASCII (American Standard for Information Interchange) forms of the protocol. If both can be handled the binary is the preferred one. The next step tries to find out the compression style and the protocol version. Now everything is ready for the communication. Messages are exchanged (semantics similar to Remote Procedure Calls RPC) between the server and the client. All control information is transported via the client port. Depending on the capability of the client a document may be compressed before it is sent using a dynamic port number. The connection is closed by the user, or by the server if the user does not react in a reasonable time (timeout).

5.4 Dino

Currently in the Internet each user acts independently. There is are lots of possibility for communication or collaboration between several users. Such solutions can be chatrooms, news or conference systems, but they satisfy only particular demands and are often platform dependent and they cannot communicate with each other.

As a solution to this problem Dino (distributed interactive networking objects) has been developed. Dino is a framework for distributed applications on the web. It can live on any platform, as long as a Java Virtual Machine is available, because it is fully written in Java. The center of Dino is a messaging system called OMA (Open Messaging Architecture) which can load applications at runtime. These applications are allowed to send and receive messages. A number of system applications offer some basic functionality. For example OPA (Open Protocol Architecture) enables the use of protocols like HTTP or FTP to connect to a server.

5.4.1 OMA

OMA (Open Messaging Architecture) is the heart of Dino. Here all the applications are held together via a linking interface called SPOM (Service Provider for Open Messaging). Each application is assigned to its own unique SPOM. The only possibility of communication for an application is using the services provided by the SPOM.

Therefore SPOM can be interpreted as a link between an application and the Dino system. Each SPOM is an instance of the SPOM class. In the SPOM class a static table keeps the

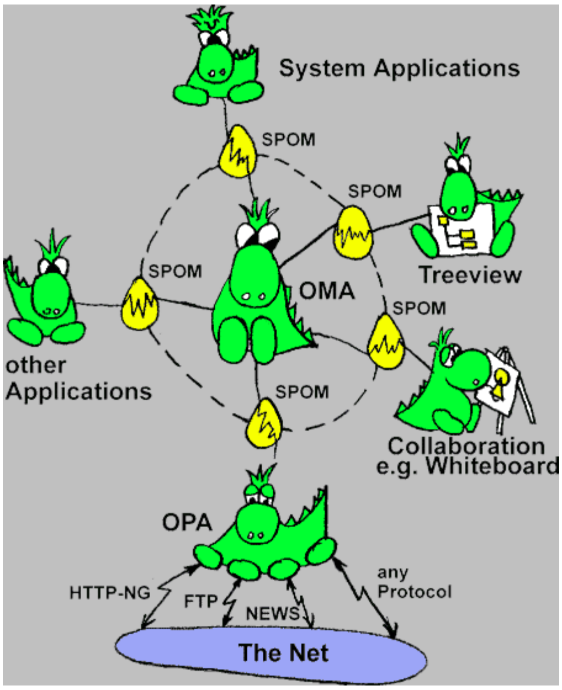

Figure 5.4: The internal organisation of Dino [Zwa97].

information of all existing SPOMs. With the aid of this table, SPOMs can directly send messages to one specific SPOM. An application can only send a message by using the SPOMs send message. The SPOM delivers this message according to the specified receivers:

- *to All:* All other SPOMs get this message. The SPOMs can be imagined as a circle, where each element has a connection to his direct neighbours. Now this message goes along, till one SPOM realises that he gets the message for the second time. Then this SPOM stops the message. Here every SPOM gets the message, even the sending SPOM.

- *to Others:* The same as to All, with one exception, the sending SPOM does not get the message.

- *to ID:* Using the static internal list, the SPOM sends the message to the SPOM with this ID.

- *to Myself:* There exists also the possibility to send the message to nobody else but myself (the sending SPOM).

- *to System:* The message is sent to the system SPOM. This SPOM reacts in the wished way.

One SPOM instance called the System SPOM is special. A System SPOM does not have an application, but is responsible for the other SPOMs. When an application wants to connect, a message is sent to the System SPOM. Now a new SPOM is generated and assigned to the application. All the work to add the SPOM to the OMA and to update the internal table by

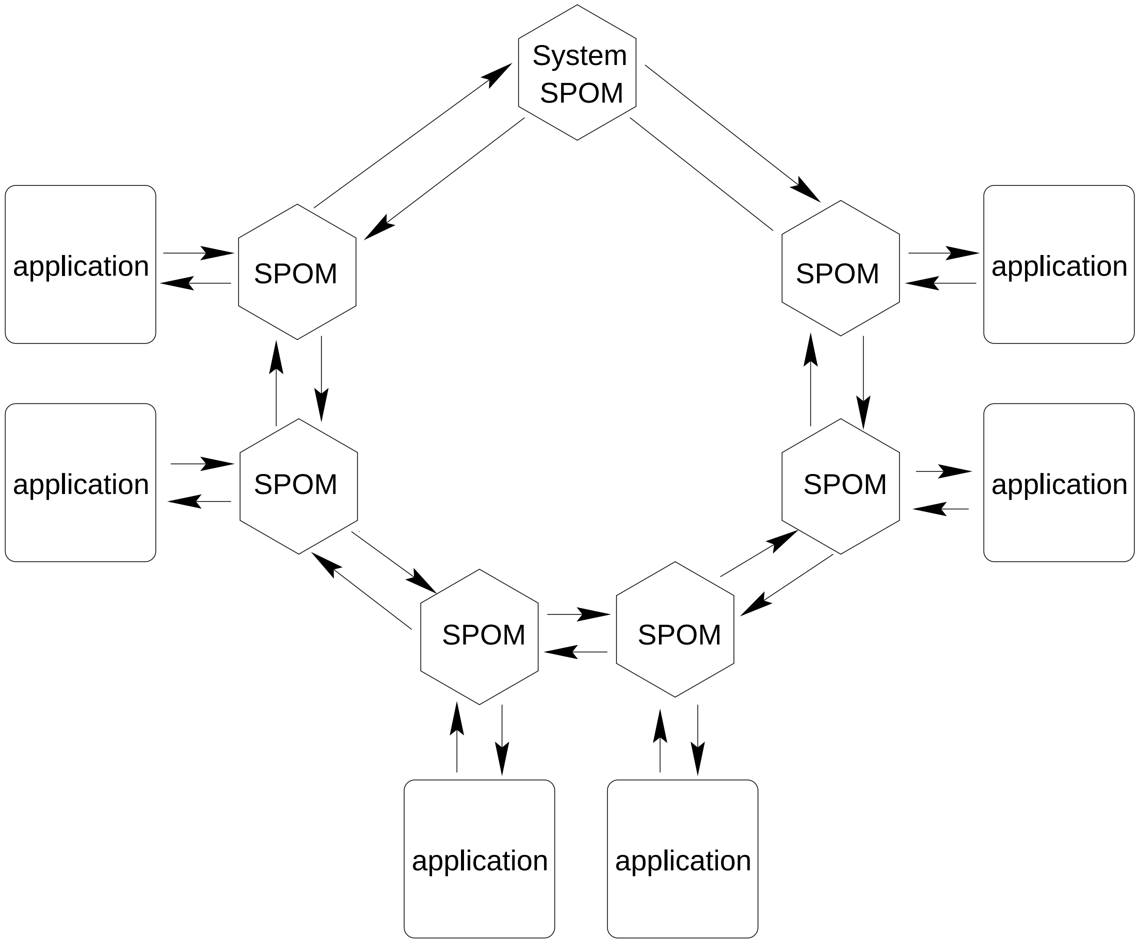

Figure 5.5: The structure of OMA (Open Messaging Architecture) [Zwa97].

registration is done here. If a message is sent to a system application, which does not exist at the moment, the System SPOM can open this application and send the message to the new SPOM.

What happens at a SPOM when a message arrives? Each SPOM has a queue holding the arriving messages. To deliver a message to the application the SPOM calls the applications `gotObject(...)` defined in the interface `DMessageReceiver`. The application can handle the message in any way; react, respond or even throw it away, depending of the type of message. Messages are extensible in any way which allows to create well fitting types for every application.

5.4.2 System Applications

System applications are developed to provide some functionality for distributed networking. They are within the Dino system package and have higher priority than all other applications. In the following some of these applications are described in more detail.

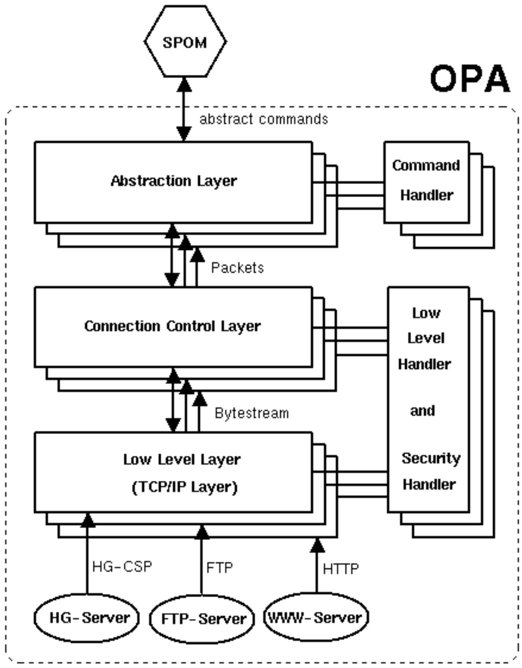

Figure 5.6: The internal layer structure of the OPA (Open Protocol Architecture) [Zwa97].

OPA

The OPA (Open Protocol Architecture) is probably the most important system application. This application is responsible for access to servers around the world, no matter which kind of protocol they use. Other tasks are the access to the local file system and local databases, the connection to other Dinos for synchronous collaboration and secure connections on the transport layer.

OPA on its own cannot speak any protocol. But it knows how to get an appropriate handler if one is available. That means that OPA loads particular handlers at runtime if they are needed. For this reason OPA is extendable to any existing or new protocol, where protocol includes reading file systems and local databases of any format. The protocol handler can be connected to the OPA using one of its three layers.

- the *Abstraction Layer (AL):* this layer understands application level messages, the so called server messages. With the help of an protocol specific Command Handler the messages are translated them into packets with some additional information.
- the *Connection Control Layer (CCC):* converts the packets into a bytestream
- the *Low Level Layer (LLL):* sets up an TCP/IP connection

To communicate with OPA an user application has to derive its message from the DServer-Message. This server messages contain a so called description. It can be constructed from an URL, but it is more than an that. The description defines the spoken protocol, the host-concerned, the contact port and additional information.

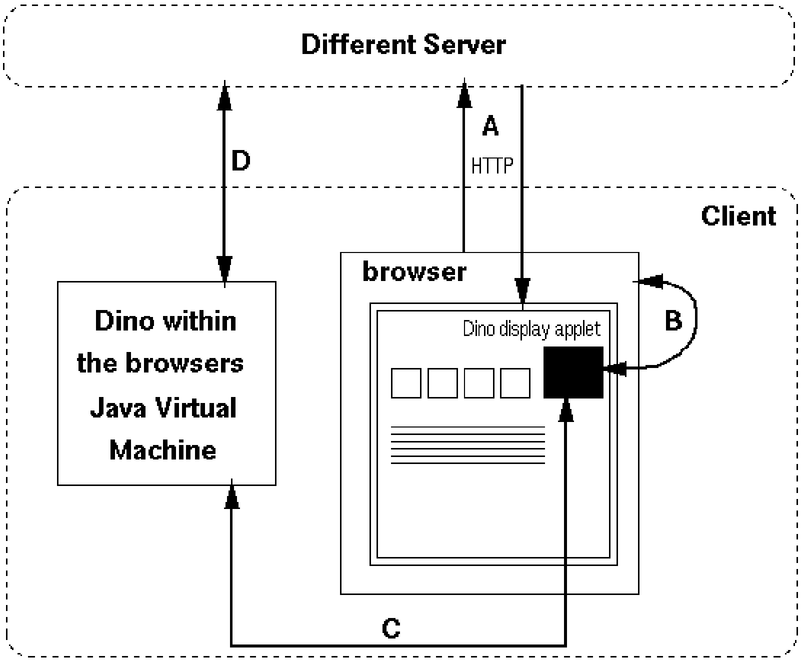

Figure 5.7: The Dino BCM (Browser Control Manager) [Zwa97].

BCM

Most times Dino will work within the Java Virtual Machine of a browser like Netscape Navigator or Microsoft Internet Explorer. In such a case the Dino will be started by an applet. As Applets are instanciated by the Java Virtual Machine of the browser they get an object, the AppletStub. This allows Dino to get all other applets from the same HTML page, to load images and audio clips, to tell the browser to show some string in the status bar and most important to let the browser load and show any particular HTML page from the Web.

Most functions of the AppletStub are only valid if the applet which it belongs to is visible or within a visible HTML page. To ensure to have at least one *valid* applet a *dummy applet* can be constructed. This so called DisplayApplet has to be embedded into every page on the server. Working with a Hyperwave server eases things, as this task will be done for you without the need for editing every HTML page. The display applet registers at the BCM (Browser Control Manager) any time it becomes active and any time it gets deactivated. Thus the BCM can hold a record of all active applets (see connection C in Figure 5.7). If a Dino application for instance wants to make the browser show a specific document, it generates a showDocument message and sends it to the BCM. The BCM will then look into its record for an active display applet which it will use to perform the requested action. To load the document, the browser uses its own connections, in most cases this will be a HTTP connection (A in the figure). The Dino itself can connect any server on the Web with help of the OPA as long as there are no security restrictions. Such connections are indicated with D in Figure 5.7.

Chapter 6

Development of SearchVis

This work concentrates on the building of clusters using force-based methods.

- *A cluster* can be interpreted as a group of similar objects.
- *Force-based methods* are calculated using several iterations. During each iteration a resulting force on each object depending on all other objects is produced.

To show the user how the forces act on the objects the result of each iteration is visualised. In this way an animated visualisation is generated.

This chapter describes the concept of *SearchVis: visualising search result sets using a force based method to form clusters of similar documents*, the idea behind, the development, the actual algorithm and the possibilities for users.

6.1 Motivation

As the information space keeps growing it gets more difficult to gain an overview. When the user searches for some information mostly more feedback is retrieved as expected. In the typical information representation of search engines this search result set is presented as a set of objects sorted according to decreasing relevance. Unfortunately relevance estimation is always subjective. Therefore the information for which the user looks might be represented by the first object and some other related objects at quite different positions. These facts lead to the need of visualisation to help the user:

- *find the most relevant documents* which satisfy the user's information request
- *find groups of documents* which deal with the same subject
- *find similar documents* to one document of interest, which has already been found

To take an example, suppose the user is looking for *mouse*. As this is a very inexact query the search result set may contain information about computer mice, field mice and Mickey Mouse. All these results have the same high overall relevance. Therefore they will appear in some random sequence.

The actual approach tries to give an easy way to find related documents by building clusters (groups) of similar objects. Looking at the above example three clusters would be formed and the user could decide which one is the most interesting. The maximum effort to find this relevant cluster would be to look at one document of each cluster.

6.2 Development

The aim of this project was to generate a graphical representation for search result sets. In a first step it was tried to generate clusters containing similar documents using the force directed placement (see Section 3.1).

For all N documents in the search result set a force acting on this document was calculated. This force consisted of the subforces to the (N-1) other documents in the search result set. To get the force between a pair of documents the geometric distance of these objects and their similarity was used. Unfortunately this approach had several severe problems. The most disturbing, was that this algorithm tends to get stuck in local minima. Therefore, clusters do not separate the right way. Another annoying side-effect of this algorithm is, that it is very time consuming as it has a complexity proportional to N^2.

To get rid of these features a new start was made using an improved force based model according to the method published by Matthew Chalmers in 1996 *'A Linear Iterative Layout Algorithm for Visualising High-Dimensional Data'* [Cha96]. This method can be described as a stochastically-based algorithm of linear complexity. It produces good layouts and has low overhead. Another important fact is, that it is easy to implement. Here the forces are built by calculation of the subforces between the actual object and the members of two sets. The first set consists of the neighbours and is updated during each iteration. In contrast to this set the second one is generated anew for each iteration. Both sets have a fix predefined size. The neighbour set is characterised by an additional variable, containing the maximum distance between the current document and the farest document in the neighbour set.

6.2.1 The Implemented Algorithm

In this approach the high-dimensional distance d_{ij} of the original method is replaced by a value proportional to the similarities. The new calculation of d_{ij} can be described as the following:

$$d_{ij} = (1 - s_{ij}^p)$$

Notice that in this case s_{ij} is not the scalar product of the two term vectors i and j, but the similarity of i and j. The exponent p can be interpreted as an factor to increase the discrimination of the similarity metric. To show the effect of p examine Table 6.1.

If the p value is one, than it does not change anything. Each high dimensional distance is calculated by subtracting the similarity from the value one. It can be easily seen, that this causes regular steps between the similarity. To receive a markedly higher d_{ij} value for similar objects and a smaller value for objects with low similarity, p is used. This causes all similarity values to get smaller, but the distance between them is not any more fixed. Similar objects have a low and non similar ones a large high dimensional distance. Now if

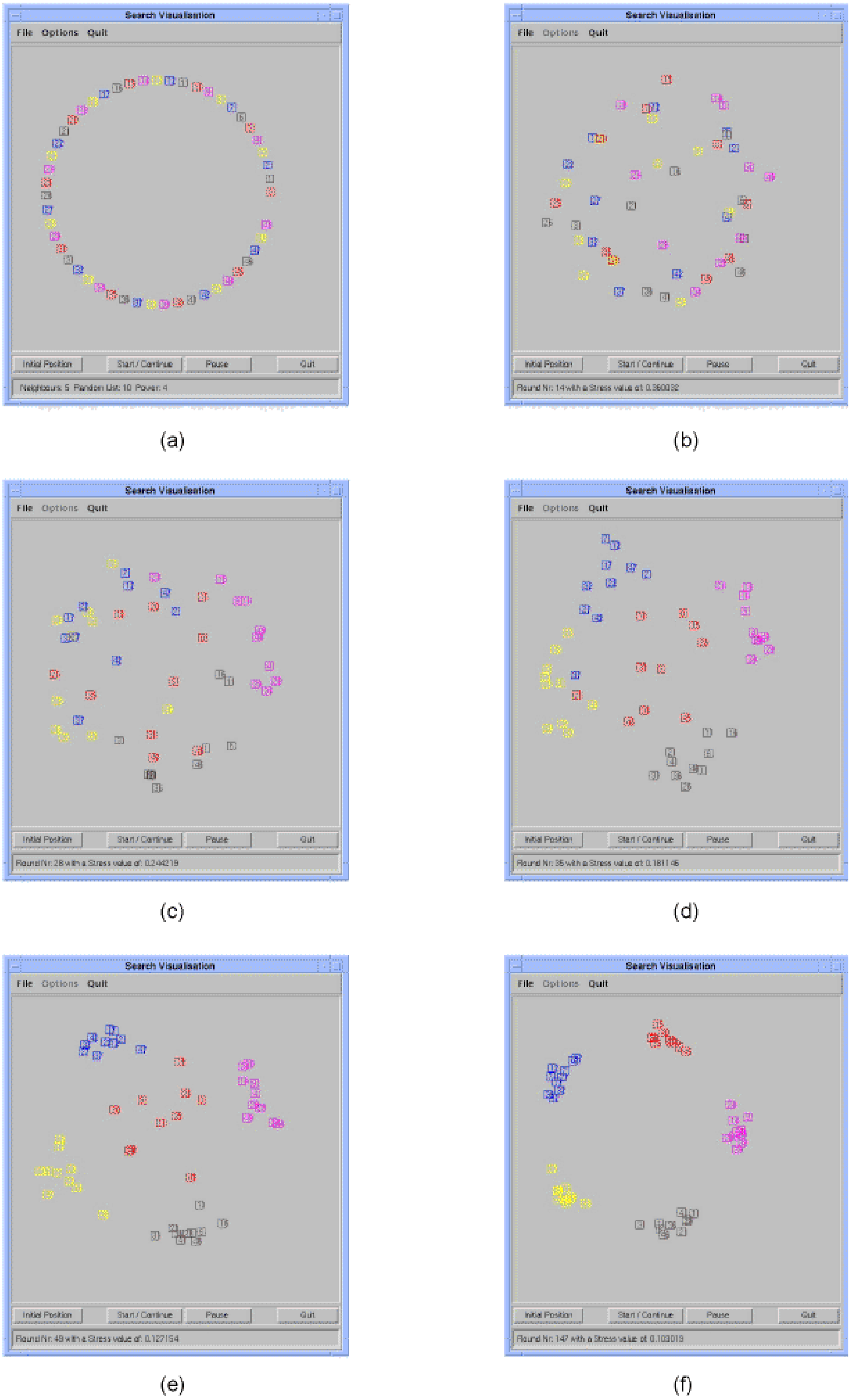

Figure 6.1: The generation of four similarity clusters showing the intermediate steps.

p = 1		*p = 2*		*p = 3*	
s_{ij}	d_{ij}	s_{ij}	d_{ij}	s_{ij}	d_{ij}
0.9	0.1	0.81	0.19	0.729	0.271
0.8	0.2	0.64	0.36	0.512	0.488
0.7	0.3	0.49	0.51	0.343	0.657
0.6	0.4	0.36	0.64	0.216	0.784
0.5	0.5	0.25	0.75	0.125	0.875
0.4	0.6	0.16	0.84	0.064	0.936
0.3	0.7	0.09	0.91	0.027	0.973
0.2	0.8	0.04	0.96	0.008	0.992
0.1	0.9	0.01	0.99	0.001	0.999

Table 6.1: The change of the similarities according to the p value. The distance is the difference to the similarity value in the next row.

the subtraction is implemented only really similar documents have a small high dimensional distance and the distance grows quite quick. Looking at the above example for (`p` = 3) a similarity of 0.7 has already a high dimensional distance of 0.657. All similarities smaller than this have already a high dimensional distance of more than 0.9.

Before the algorithm can be applied all documents have to be placed at their initial positions and the variable `max_distance`, representing the maximum geometric distance in the neighbour list, is assigned a high value. The following steps are made once for each object during each iteration:

1. *Generate the sets:* during the first iteration both sets are null. To fill them random elements of the search result set are taken and their geometric distance to the actual document is calculated. Now two possible states may occur:

   - The distance is smaller than `max_distance`: This object is inserted into the appropriate position of the neighbour set. The elements of this set are stored in order of distance. As soon as this list is full, the variable max_distance is assigned the distance value of the final element. This insertion can cause other elements of the set to be removed since the length of the set is fixed.
   - The distance is greater than `max_distance`: this object is inserted into the random set. Remember that this list is not kept by the system. After one force calculation it is again set to null.

   The above step to choose an element is repeated until the random set is filled. This implies a full neighbour set. An important condition is, that no document is allowed to occur more often than one time in the combination of both sets. In other words it may either be an element of the neighbour set or of the random set.

2. *Calculate the new position:* the new position is retrieved according to the forces caused by the elements of the sets. Because of the interaction with each element a new position for the actual object is found using interpolation.

$$t = d_{ij}$$

$$p_i j(x, y) \quad = \quad ((1 - t) * p_i(x, y) + t * p_i(x, y))$$

To get the final position all these positions are added and divided by the length of the neighbour set (max_neigh) plus the length of the random set (max_rand):

$$p_i(x, y) = \frac{\sum_{j=1}^{max\_neigh+max\_rand} p_i j(x, y)}{max\_neigh + max\_rand}$$

3. *Draw the documents at their new position:* each document is positioned at its new place. As this is done for each iteration the visualisation is *animated*. The user can see how the different forces influence a particular document.

To terminate the iteration two control thresholds are used. One is the number of iterations and the other the stress threshold. The stress value can be described as the residual sum of squared errors of all interobject distances. It is evaluated using the following formula:

$$\text{Stress} = \frac{\sum_{i<j} (d_{ij} - g_{ij})^2}{\sum_{i<j} g_{ij}^2}$$

As soon as the number of iterations exceeds its threshold, the algorithm is stopped no matter what value of stress.

If the stress value falls below the stress threshold, the application is not terminated immediately, since it might be possible to find a better stress value. Iteration continues as long as stress values continue to decrease. Two cases may cause the iteration to stop. The maximum number of iterations can be reached, or the actual stress value can increase again. Both possibilities cause the application to terminate at once.

6.2.2 Differences

In this section the differences between the actual approach and the original approach are looked at. The distinctions start at the document representation. In the original approach a document is represented by a term vector as in the vector space model (see Section 2.4). Here the documents are represented by their relevance and an additional similarity matrix. This causes the different representation of the d_{ij} value.

Another important difference is the integration step to update the position. Chalmers performs this step using a fourth order Runge-Kutta although the simpler Euler Step is often adequate. There the position and velocity for each object is updated. For stability, viscous damping proportional to the velocity is added and excessively high forces are clipped. In the modified model this step to update the position is done completely different. Here no velocity and therefore also no damping is used. To calculate the new position the force between two documents and their geometric distance is used in an interpolation formula. This approach is very simple and reduces the computation effort markedly. A negative effect is that the animation becomes less smooth, nevertheless it is still sufficient.

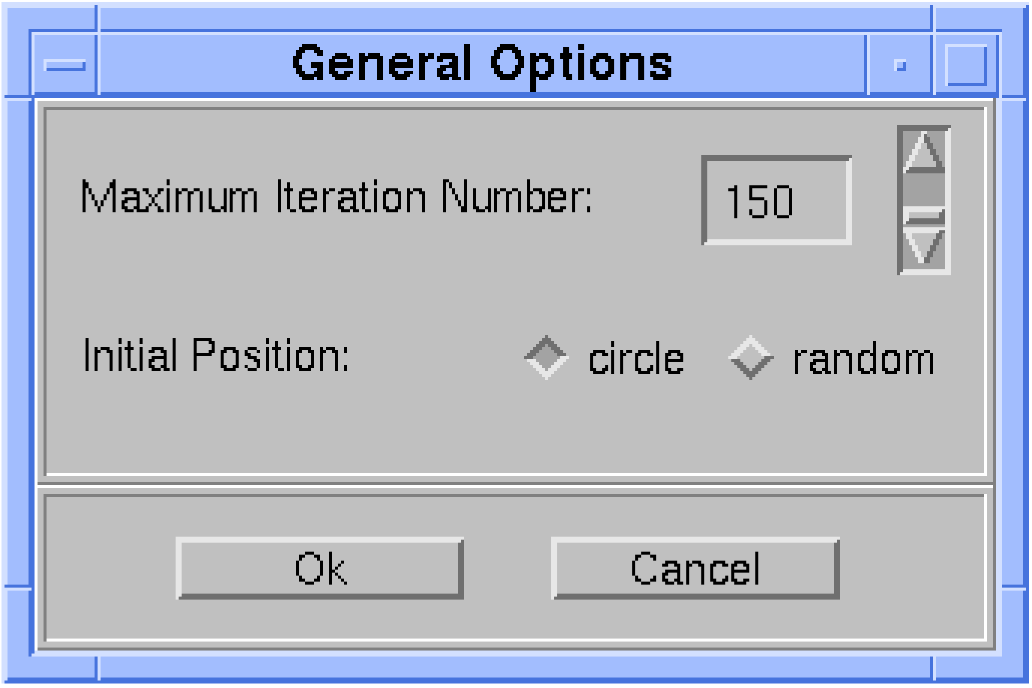

Figure 6.2: The General Options panel.

6.3 User Options

As this application is developed as an aid for the user, several adjustments can be done. The visualisation can be interrupted and continued any time the user wishes, just by pressing the appropriate buttons (break and go on). For some additional information a status line is added. The user cannot only view the different iteration steps, but also control a number of different options. These options include the termination thresholds and the size of the two sets. They are divided into general, expert and similarity options.

6.3.1 General Options

These options help the inexperienced user to adjust some features, which have predefined default values. The user can change:

- the maximum number of iterations: This is one termination threshold. As soon as this iteration number is reached the algorithm stops. It is predefined to be three times the number of documents in the search result set.

- the initial document position: This position can be defined to be random or at the radius of a circle. The general start position is a circle.

If these values are changed the behaviour of the algorithm is not really different. A possible difference might be a longer computation time.

Figure 6.3: The Expert Option Frame.

6.3.2 Expert Options

In this panel an experienced the user has more possibilities. The following values can be changed:

- the size of the neighbour set
- the size of the random set
- the discriminator of d_{ij} the value p
- the termination threshold stress

Changing these values may cause a quite different result. The computation time which depends on the size of the sets an the termination threshold can increase or also decrease. The separation of the cluster also depends on the set sizes. Not to forget the value p which is responsible for good discrimination.

This frame also checks if the new values are reasonable. If any error occurs it is closed and the former values are kept. Here the only values which have to be tested are the sizes of the sets. Their sizes added may have a maximum common length, equal to the number of documents retrieved in the search result set, N.

6.3.3 Similarity Options

Unfortunately up to now no real similarities are provided by the Hyperwave server. Therefore those values have either to be taken from a test file or generated randomly.

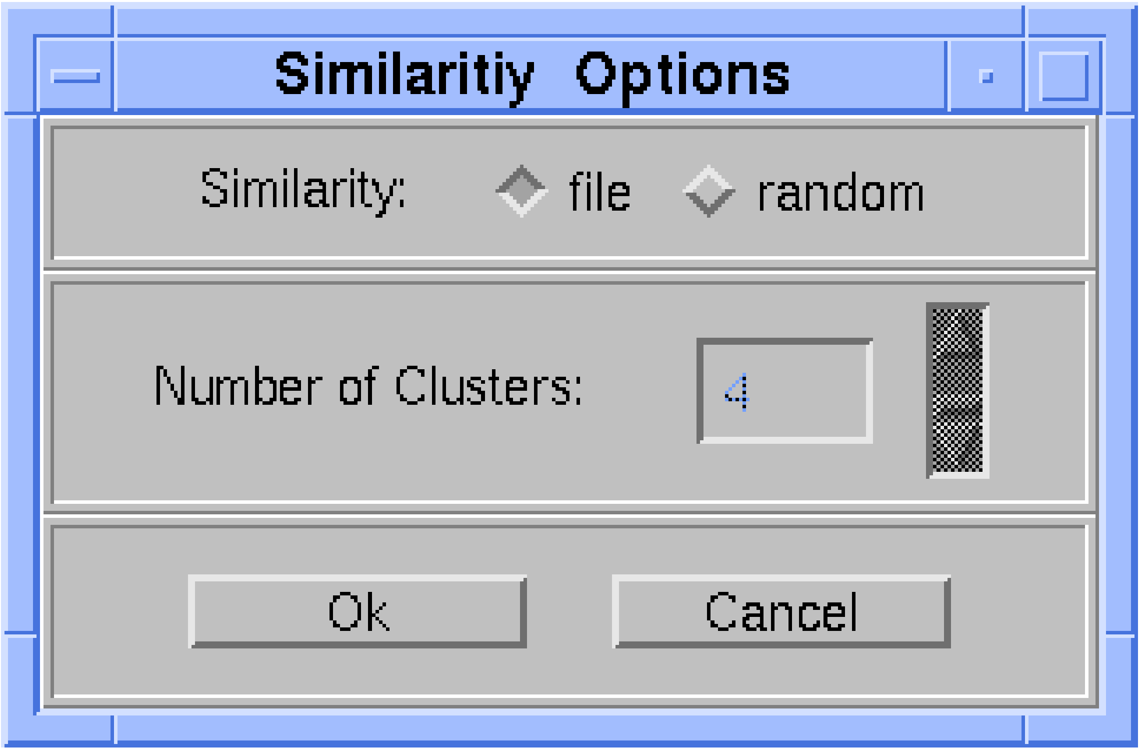

Figure 6.4: The Similarity Option Frame.

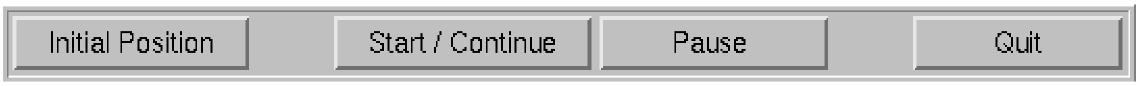

Figure 6.5: The Action Panel.

Random similarity generation allows to define a certain number of clusters in advance. This causes the documents to ge separated into that many groups. As a cluster may also contain zero elements the group number is equal or less to the set number.

Each document is randomly assigned to a cluster. In a next step the similarities are generated. Inside a cluster the objects have a similarity value in [0.75-1]. The similarity to documents outside this cluster has a value somewhere in the interval [0-0.25].

6.3.4 Action Panel

To be user-friendly the main actions are represented by buttons. Those buttons are placed on a panel, which is situated at the at the bottom of the application. The following possibilities are offered:

- *Initial Position:* This button causes the documents to go back to an initial position. This position can be random or at the radius of a circle according to the settings made in the *General Option Frame*. At this state all options can be changed.

- *Start / Continue:* To start or continue an iteration. The option menu item gets disabled. As all settings have to be made before the calculation starts.
- *Pause:* Pressing this button causes the calculation to be stopped. This is especially useful if a certain iteration step is already satisfying. Another possibility is that the user wants to have a closer look at the actual distribution of the documents.
- *Quit:* If the user wants to leave the application, this button has to be pressed. Another possibility for leaving is offered via the menu bar.

These actions can be regarded as the main part of the *SearchVis* application as they offer all possibilities to run the tool. Special settings can be done using the appropriate order in the menu bar.

6.4 Java

This application was written in Java, because this language is designed for programming on the internet. Therefore a Java applet or application can run in Internet browsers like Netscape or Microsoft Internet Explorer. For the user the network will become the computer. It offers the possiblility to take software from some random web page and execute it on any platform.

To describe Java the following catchwords can be used:

- *simple:* to enable most of the programmers to learn Java quickly, it was made similar to the well-known language concept of C. Nevertheless Java is not just a C derivative. A lot of features of C were not implemented to keep the language small.

  One important simplification it the elimination of pointers. Those are not needed in Java since arrays and strings are objects and structures are not supported.

  Another simplification is the automatic garbage collection. This handles the memory management issues.
- *object oriented:* Java is a real object oriented language, as this concept was designed from the ground up. Therefore nearly anything is an object. The only exception are the simple data types.
- *platform independent:* because of this platform independence any Java application is portable. That means that it can be taken to a new operating system and run there without any additional effort.
- *secure:* this is especially important as Java is implemented to run on the Internet. Therefore some mechanisms to protect the user against code that might try to smuggle in a virus or invade in the file system is needed.
- *multithreaded:* This means that Java provides the possibility to execute multiple threads at the same time. Such a thread can be the playing of an audio clip, scrolling in a page or the process of downloading an image.

The above points show the importance of a programming language like Java. Nevertheless there exit still many other features of Java, which have not been mentioned. One aspect, which sounds terrible for a lot of programmers is the speed of Java. Compared with C it is about twenty times slower. But for interactive network-based applications this is sufficient.

6.5 Discussion

This method does not easily get stuck in local minima as the termination conditions are more flexible. It is not automatically stopped if the stress value increases as this value should reach a state under a certain threshold. If this threshold is too low the second termination condition prevents an endless loop.

Unfortunately the document representations of the visualisation result overlap. To change this a repulsive force had been implemented. Nevertheless this idea could not change anything as the algorithm has no complexity of N^2. Therefore the objects which were not considered for the generation of the new position kept disturbing the layout. As this force did not cause the wished result it was eliminated.

Chapter 7

Embedding in Dino

In this chapter is described how an application can be added to the Dino system. The main and the communication parts are discussed. To add an application a SPOM has to be generated and registered. This SPOM realises the connection between the Dino kernel OMA and the application. SPOMs can communicate with other SPOMs. One SPOM is special, it is responsible for the registration and it may open some system applications as OPA or BCM. This SPOM is called system SPOM.

7.1 The HTML page

To start any java applet in the net or an applet viewer, a HTML page referring to the main class is needed. In this case the HTML page generates opens the `dino.system.bcm.Dlauncher` class using an APPLET tag. This is a simple applet which offers to open an application by pressing a button. If the button is pressed the application class `dino.app.searchvisualisation.messages.MakeVis` is started. This class implements the SPOM generation, registration and opens the actual application which visualise the search result set. An example HTML page is shown in Table 7.1.

This example HTML page shows some of the different parameters which can be used. The parameter named set0 - set3 set some system properties as the language and the session key. They can be regarded as static global properties for all applications.

The other having the name prop0 - prop20 define special layout parameter and application properties.

- prop 0: the type of the button to start the application. For example a TextButton or a ImageButton
- prop 1: defines the label of the button
- prop 2: defines a standard image
- prop 3,4: if an image button is used the image defined in prop3 displays the buttons standard state. As soon as this button is pressed the images changes to the active image defined in prop4.

```
<APPLET name="Start SimpleApp in Window"
CODE="dino.system.bcm.DLauncher.class" height=400 width=420>

<PARAM name="set0"   value="dino.system.bcm.sessionkey           = %%sessionkey%%">
<PARAM name="set1"   value="dino.system.bcm.language             = %%language%%">
<PARAM name="set2"   value="dino.system.bcm.showtimeout          = 45000">
<PARAM name="set3"   value="dino.system.bcm.openwindow           = false">

<PARAM name="prop0"  value="dino.system.bcm.launcher.buttontype  = textButton">
<PARAM name="prop1"  value="dino.system.bcm.launcher.label       = Start Visualisation">
<PARAM name="prop2"  value="dino.system.bcm.launcher.std_image   = images/slee027i.gif">
<PARAM name="prop3"  value="dino.system.bcm.launcher.act_image   = images/dino027w.gif">
<PARAM name="prop4"  value="dino.system.bcm.launcher.start       = Click">
<PARAM name="prop5"  value="dino.system.bcm.launcher.xpos        = 240">
<PARAM name="prop6"  value="dino.system.bcm.launcher.ypos        = 290">
<PARAM name="prop7"  value="dino.system.bcm.launcher.height      = 20">
<PARAM name="prop8"  value="dino.system.bcm.launcher.width       = 210">
<PARAM name="prop9"  value="dino.system.bcm.launcher.bg_color    = 0xf3ad32">
<PARAM name="prop10" value="dino.system.bcm.launcher.color       = 0xa50852">
<PARAM name="prop11" value="dino.system.bcm.launcher.font        = Helvetica">
<PARAM name="prop12" value="dino.system.bcm.launcher.fontsize    = 14">
<PARAM name="prop13" value="dino.system.bcm.launcher.fontstyle   = bold">
<PARAM name="prop14" value="dino.system.bcm.launcher.fontcolor   = 0xfffffe">
<PARAM name="prop15" value="dino.system.bcm.launcher.homepage    = cannot_load_
application.html">
<PARAM name="prop16" value="dino.system.bcm.launcher.bg_image    = images/dino027i.gif">
<PARAM name="prop17" value="dino.system.bcm.launcher.buttoncolor = 0xa50852">
<PARAM name="prop18" value="dino.system.bcm.launcher.application = dino.app.
searchvis.messages.MakeVis">
<PARAM name="prop19" value="dino.system.bcm.applet_type          = 5">

<PARAM name="prop20" value="dino.system.alm.layout               = newwindow">
</APPLET>
```

Table 7.1: The HTML page to start a Dino application.

- prop 4: the action which has to be done to start the application, here *click*.
- prop 5,6: the position of the upper left edge of the button
- prop 7,8: the height and the width of the button
- prop 9,10: the background colour and the foreground colour of the launcher
- prop 11,12,13,14: defines a font by telling its name, size, style and colour
- prop 15: a HTML page defining where to go to if the application cannot be loaded
- prop 16: defines a background image, which can be the sign of the application.
- prop 17: the colour of the button
- prop 18: the name of the application to create and register a SPOM
- prop 19: the type of the applet. Here this type consists of a combination of two values. One represents the display layout which can serve as a target object for a visualisation. The other represents a standard layout.
- prop 20: sets the layout for the new application in the ALM (application layout manager). This value can be one of three namely window, newwindow or browser. This layout is set as soon as the start action has been made.

It is important that no number of the parameter name starting with prop is skipped. If so, the following numbers would be ignored.

7.2 Registration and Application Generation

In the HTML page an application is assigned. This application is responsible for the creation and registration of a new SPOM. It extends the Dino class `DRegisterMessage` which is a superclass to create messages to register with the system.

This superclass provides an action method which obtains the new SPOM. In this method a class extending the `DMessageReceiver`, the actual application, has to be defined. This `DMessageReceiver` is a superclass which gets the messages from the SPOM. It is the return value of this method.

7.3 The Application

The class SearchVis handles the appearance of the actual application. It implements the `DMessageReceiver` class and is responsible for the incoming messages from the SPOM. These messages are retrieved in the method `gotObject()` which can be called by the appropriate SPOM. Inside this method the message is controlled and according to it's type a specific reaction is made. As the application is not opened by Dino, but using this class,

```
public class MakeVis extends DRegisterMessage
{
  // My Application
  private SearchVis application_;

  // SPOM
  private DSpom mySpom_;

  public MakeVis ()
  {
  }

/**
* Creates and registers the application by creating and registering the SPOM
* This method is called by the Dino system during registering. It returns a
* DMessageReceiver.
*/
  public DMessageReceiver action(DSpom spom)
  {
    mySpom_ = spom;
    Color bgColor = null;

    try
    {
      bgColor = getProperties().getColor("dino.system.bcm.launcher.bg_color");
    } catch (NullPointerException e){}

    application_ = new SearchVis(spom, getProperties());

    return (DMessageReceiver) application_;
  }
}
```

Table 7.2: This class generates and registers the SPOM. This is done by extending the `DRegisterMessage`.

some actions to close the application have to be set. The method `killMyself()` kills the appropriate SPOM. Afterwards the application has no more possibility to communicate with the Dino system. To close the application window the method `disposeApplication()` is used.

7.4 Discussion

The connection to the Dino system offers several possibilities for communication. The application can contact other user applications or system applications via their SPOM. To keep some kind of order Dino provides a system SPOM to control and register all other SPOMs. If a connection to a system application which is not actually running is wished, the system SPOM can creates this application.

Dino offers some extended java objects. This means that for example a button can be a `TextButton` or an `ImageButton` and has some special features. In general the elements realised in Dino are more comfortable and also much quicker than the overall java elements.

Unfortunately Dino is still at the beginning of its development and therefore the possible features are still not all implemented.

```
public class SearchVis implements DMessageReceiver {

  private DSpom spom_;
  private int id_;

  private DProperties properties_;

  private MainFrame vis_frame_;


  public SearchVis(DSpom spom, DProperties properties) {
    spom_ = spom;
    properties_ = properties;

    id_ = spom_.getId();

    // Start the application //
    vis_frame_ = new MainFrame(this);
    vis_frame_.show();
  }

  public synchronized void killMyself() {
    System.out.println("killMyself");
    spom_.send(new DMsgKill(), DMessage.SYSTEM, 0, null);
    disposeApplication();
  }

/**
 * Since Dino didn’t open my frame, I have to close it myself!!
 */
  private void disposeApplication() {
    System.out.println("disposeApplication");
    vis_frame_.dispose();
  }

  public void gotObject(DMessage message, DMessageInfo info) {
    if (message instanceof DMsgQuit) {
      killMyself();
    }
    else if (message instanceof DMsgKilled) {
      DMsgKilled msg = (DMsgKilled) message;
      if (msg.id() == id_) disposeApplication();
    }
    else if (message instanceof DMsgShutdown) {
      disposeApplication();
    }
  }
}
```

Table 7.3: The application is generated implementing a `DMessageReceiver`.

Chapter 8

Evaluation of Different Parameters

This chapter tests the reaction of the *Visualising Search Result Sets Using a Force Based Method to Form Cluster of Similar Documents* application depending of the different options.

This is done to find out the behaviour of the system according to those values. Such values are the size of the two sets and the distinction value p. Afterwards some characteristic values were determined, which can be used as general settings.

8.1 Set Size

This approach uses two sets to realise a linear complexity. Both sets have a fixed length which can be changed in the Expert Option Frame (see Section 6.3.2). To join an appreciation of the overall behaviour of the algorithm according to the set size, four different possibilities were tested.

8.1.1 Large Neighbour Set and Large Random Set

It is clear that large sizes are not very sensible as they increase the computation complexity. Nevertheless some higher values were tested to have some relative visualisation result.

Those combinations lead to good clusters which were very separated from each other. Unfortunately this distinction is reached at the cost of the document representation. This means that the documents overlap a lot. Therefore it is not obvious to find out which documents are present in which cluster. It can be done by moving the mouse at the documents and regarding the information in the status line, but this is an extra effort.

The animation of the visualisation suffers of the high complexity. As the many computations during each iteration cause quite long interruptions of the flow.

8.1.2 Small Neighbour Set and Small Random Set

As the constellation above has some problems the opposite is tested. Both set sizes get the same small value. This reduceds the computation complexity a lot. Nevertheless this results have some severe shortcomings.

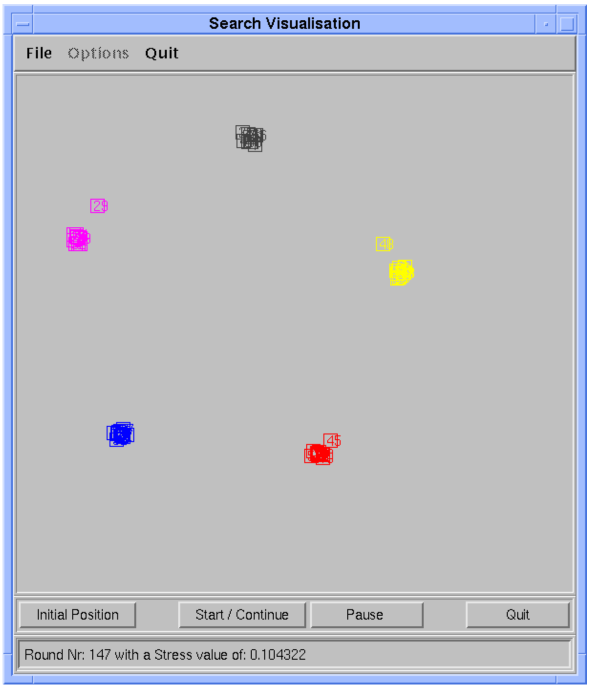

Figure 8.1: Large set sizes. In this example both sets had a size of twenty. The search result consisted of fifty elements.

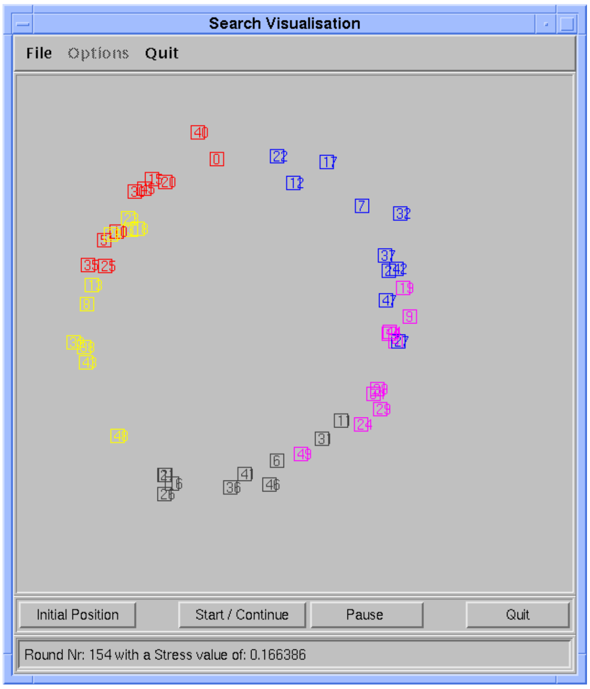

Figure 8.2: Small set sizes for the predefined iteration number. In this example both sets had a size of five. The search result set consisted of fifty elements.

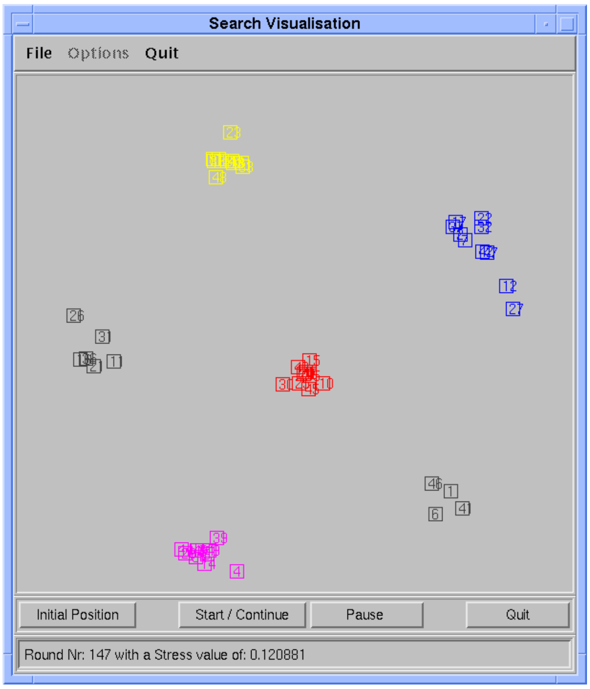

Figure 8.3: Large neighbour set size and small random set size. In this example the neighbour set had a size of twenty and the random set a size of five documents. The search result set consisted of fifty elements.

The considered set elements show only a very small range and therefore it takes much more iterations to get a good result. This happens because at each iteration only few forces are calculated. In this case the indirect neighbour linkage does not work that good as possible.

Using these properties the stress value received for the the predefined maximum iteration number is still quite high. To improve this visualisation more iterations are needed. This causes again an increasement of the complexity.

Therefore a good value has to have a small and a large set size to combine the advantages of both.

8.1.3 Large Neighbour Set and Small Random Set

Another constellation which has been tested, implies large neighbour sets and small random sets. The overall complexity of this parameters produces good results.

Testing some different combinations showed an unexpected result. This constellation produces errors. The visualisation shows not only the wished cluster number, but one or more cluster is divided into two separate cluster.

The explanation of this effect is quite simple. As the neighbour set is large many non-similar elements are included. Those elements cause the problem as their forces are included during each iteration. The random set, which represents the less similar objects is too small to have a sufficient affect.

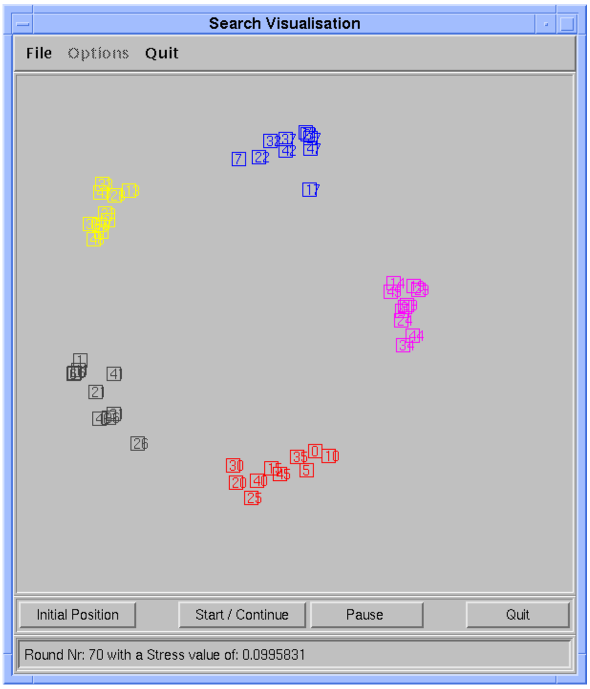

Figure 8.4: Small neighbour set size and large random set size. In this example the neighbour set had a size of five and the random set a size of ten documents. The search result set consisted of fifty elements.

8.1.4 Small Neighbour Set and Large Random Set

The final possibility consists of a small neighbour set and a large random set. This parameter values cause again a good overall complexity.

It was found out, that a great difference between the list sizes produces good results. Nevertheless this constellation causes again interruptions of the animation.

To eliminate this effect a small neighbour set and a larger, but still not large, random set was tested. The result was astonishingly good. This is due to the indirect linkage and to the document distribution in the sets. The neighbour set consists of really similar documents and the random set offers a good selection of less similar elements.

The animation of the visualisation looks good and the results which are retrieved allow a quite good distinction. Sometimes this distinction is not as good because of the random values, but in general it is sufficient. The complexity of the visualisation using these values is a linear one.

8.2 Distinction p

The distinction p is a significant factor to produce low stress values. It discriminates the different similarity values.

This is done by using p as an exponent of the similarities. This causes high similarities to separate from the smaller ones. The small similarities have quite similar values. Therefore the high-dimensional distance is only small for very similar objects. All other have nearly the maximum value.

To find out which value offers the best distinction some values have been tested. It cannot be fixed that this result is completely right as the stress depends also on the chosen random set elements. Nevertheless the values in Table 8.1 can be regarded as an overall rule of thumb.

distinction p	*final stress*
1	0.2321
2	0.1195
3	0.1060
4	0.0995
5	0.1027
6	0.1062
7	0.1056
8	0.1108
9	0.1323
10	0.1379

Table 8.1: Showing the relation between distribution value p and stress

It can be said that higher values of p widen the subrange of the high-dimensional distance. However this improvement is only reached up to a certain border. The reason for this is, that too high p values cause the high-dimensional distance to differ in very small ranges. Otherwise the result is still better using very high p values than using none.

The best value was estimated to be a p value of four. This value was therefore taken as the general value.

8.3 Stress

The stress value is an important parameter because it is a termination threshold. An other termination threshold is the maximum iteration number which causes an application to stop as soon as this value is overstepped.

Stress acts in a different way. If this value is remained under the application is not automatically stopped. Another fact is inspected. This concerns the previous stress value. As long as the stress goes down it is not stopped. Therefore a minima is found.

Table 8.2 shows that it cannot be forecast which result can be reached using a certain stress threshold. This happens because of the implementation of the termination. The smallest possible stress value is taken. Normally switches to higher stress values are allowed. This changes when the threshold is fallen short of. Then a step up causes the algorithm to stop.

stress threshold	*final stress*
0.1	0.0996
0.2	0.1056
0.4	0.3382
0.6	0.1256
0.8	0.3312
1.0	0.1564

Table 8.2: Confronting the stress threshold with the actual final stress values.

8.3.1 Discussion

In general it is more useful to choose a small stress threshold, such that the algorithm does not get stuck in local minima very easily. It is more useful to set the iteration termination threshold if it takes too long. An other practical possibility to terminate is the break button. This button offers the chance to have a closer look at the actual clustering. Afterwards the algorithm can be continued or stopped.

The stress cannot be forecasted as the documents to calculate the forces are taken randomly. This cause the new positions and therefore also the new stress.

According to the above reasons the predefined stress threshold is 0.1.

Chapter 9

Outlook and Future Work

The visualisation of document generated by *SearchVis* seems to be very simple and understandable. The user can intuitively find similar documents and as the visualisation is animated the effect of the forces can be observed. This animation helps to shorten the computation time. Generally the computation time is in a reasonable range, as this algorithm has a linear complexity, nevertheless looking at large document sets this time may be quite long. The user can shorten this time by stopping the application as soon as the layout looks sufficient. Another possibility to improve the performance can be done by switching off the animation. Then simply the final layout is presented. This is a quite reasonable action, as the "normal" end-users are probably less interested in the jiggery.

The major problem of this approach is that it has never been tested with real values, neither real search result sets, nor real document similarities. This is due to delays in the implementation of document similarities in Hyperwave on the one hand and on the lack of a search interface in Dino on the other. Therefore the most important step will be to extend the methods to request the search result sets and the similarity management. These extensions require a communication with the Dino system via the SPOM. Hopefully the necessary basis will exist in the near future. Meanwhile the document sets are taken from a test file. The similarities can also be taken from the file or be generated randomly. This can be defined in the similarity panel which offers. Using random similarities the number of clusters which should be formed can be defined by the user.

Since the computation of the document similarities is done via the server's full text index another aspect has to be regarded. This is the additional load which is put on the server and the network to calculate and transfer the document similarities as the number of similarities is $O(n^2)$. This might influence the interaction time considerably especially for large search result sets. To keep the load small, the search result sets must be capped to reasonable values, for example to the first one hundred results.

An important point which has to be done is a usability evaluation as soon as real similarities exist. This is necessary to find out if an effective and helpful layout to locate similar documents can be found in a reasonable time.

The implemention of relevance feedback is less important, but would probably be a enormous improvement. This means that the most relevant documents could be retrieved in a simplified way. As soon as a relevant document is found it is possible to start a new search looking for document which are similar to this specific document. Those similar documents can be already in the search result set, but they do not have to be there. However, the main work to implement such a feature has to be done in the server, not in the application.

Chapter 10

Concluding Remarks

In this thesis, SearchVis, an approach to visualise search result sets using a force-based method to form clusters of similar documents has been introduced. In this application the documents are moved iteratively from their initial to a final position. The movement depends on the forces acting between the documents, which are derived from the actual document distances and the document similarities. Looking at the final placement, clear groups or clusters of similar documents should be visible.

All the necessary underlying topics were presented and discussed, including the different methods of information retrieval in Chapter 2, the iterative placement methods in Chapter 3, and Chapter 5 which discusses the Internet, Hyperwave and Dino. Another chapter, Chapter 4 presents some different systems to visualise information and gives an overview of existing possibilities.

The main chapters are Chapter 6, 7 and 8. There, all relevant information concerning the application can be found. That means the implementation details, the description how to add an application to dino and an evaluation of all parameters which can be set via the option panels.

Nevertheless there are still some open questions. The main one concerns the usability of this application with real search results and similarity values. It seems reasonable that the similarities of the documents should still be shown clearly by the application, no matter if real or random similarities are used. However, the usage of real similarity values might cause the clusters to be less clear. Therefore a usability evaluation should be performed to find out the real behaviour.

Appendix A

The Test File Format and Test Files

As no real search result sets and similarities were available, a test file format was used. The format of the test file used in *SearchVis* is the same as the one used for *HarSearch* [Rod97]. This file 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 number of global object ids (GoIDs) has to be equal to the specified number of documents. The relevance score indicated how good this document matches the query.

The similarity values have to be in the form of a lower triangular similarity matrix. Each line represents the similarities of one particular document to all the documents having a lower index. Therefore the first similarity value represents the similarity of document two (line) to document one (column). The second line contains the similarities of document three to document one and document two and so on. The single similarity values have to be seperated by a semicolon.

For better readability this test file format allows comment lines. Those comments are indicated by the first character in the line. The lines have to begin either with “!” or with “#”. An example for a test file may look like Table A.1.

A.1 The Test Files and Their Results

The used test files consist of ten, thirty, fifty and hundred documents, their relevances and the similarity triangle. In the following, the test files containing ten and thirty documents are shown. Additionally a final layout for each test file is displayed. The test files for fifty and hundred documents are not included, as they do not really differ in their structure, but are quite large.

```
!number of objects
30
# GoIDs of the objects and score

0x811b9908 0x000ee197;1.0
0x811b9908 0x00067bcc;1.0
0x811b9908 0x000680a2;0.95
0x811b9908 0x000680ba;0.92
0x811b9908 0x000680c9;0.87
0x811b9908 0x001242f8;0.83
0x811b9908 0x000680cd;0.82
0x811b9908 0x0002e6db;0.78
0x811b9908 0x0038b148;0.74
0x811b9908 0x001034ef;0.71
[...]

! Similarities for 30 documents and 4 groups
! ----------------------------- 30 / 4 -----

0.28;
0.18;0.13;
0.15;0.11;0.27;
0.80;0.19;0.12;0.26;
0.19;0.87;0.24;0.20;0.25;
0.23;0.19;0.83;0.17;0.26;0.26;
0.18;0.75;0.11;0.27;0.14;0.81;0.25;
0.82;0.26;0.24;0.05;0.94;0.02;0.11;0.27;
0.14;0.93;0.21;0.26;0.28;0.88;0.25;0.93;0.27;
0.06;0.25;0.80;0.22;0.24;0.13;0.80;0.14;0.03;0.22;
0.18;0.22;0.27;0.87;0.15;0.29;0.23;0.11;0.24;0.21;0.19;
0.80;0.17;0.29;0.06;0.80;0.22;0.18;0.16;0.83;0.22;0.16;0.25;
0.21;0.80;0.23;0.22;0.13;0.80;0.20;0.89;0.31;0.83;0.36;0.21;0.20;
[...]
```

Table A.1: An example test file format.

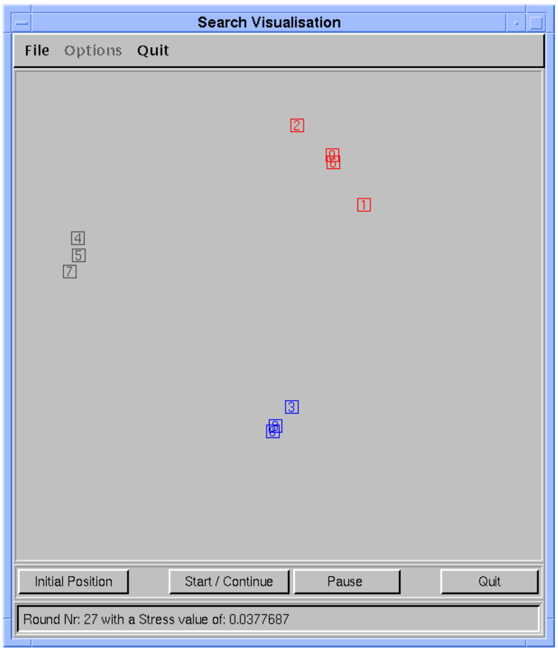

Figure A.1: The final layout for the test file format consisting of ten search results (Table A.2).

```
!number of objects
10

# GoIDs of the objects and score

0x811b9908 0x000ee197;1.0
0x811b9908 0x00067bcc;1.0
0x811b9908 0x000680a2;0.95
0x811b9908 0x000680ba;0.92
0x811b9908 0x000680c9;0.87
0x811b9908 0x001242f8;0.83
0x811b9908 0x000680cd;0.82
0x811b9908 0x0002e6db;0.78
0x811b9908 0x0038b148;0.74
0x811b9908 0x001034ef;0.71



! Similarities for 10 documents and 3 groups in which documents are similar
! ----------------------------- 10 / 3 -----------------------------------

0.87;
0.84;0.82;
0.20;0.22;0.25;
0.25;0.19;0.24;0.28;
0.15;0.23;0.20;0.17;0.88;
0.87;0.88;0.84;0.29;0.10;0.15;
0.23;0.24;0.21;0.21;0.90;0.88;0.24;
0.17;0.24;0.15;0.86;0.14;0.23;0.20;0.16;
0.13;0.18;0.10;0.84;0.23;0.22;0.25;0.15;0.93;
```

Table A.2: The test file used for ten documents and three clusters of similar documents.

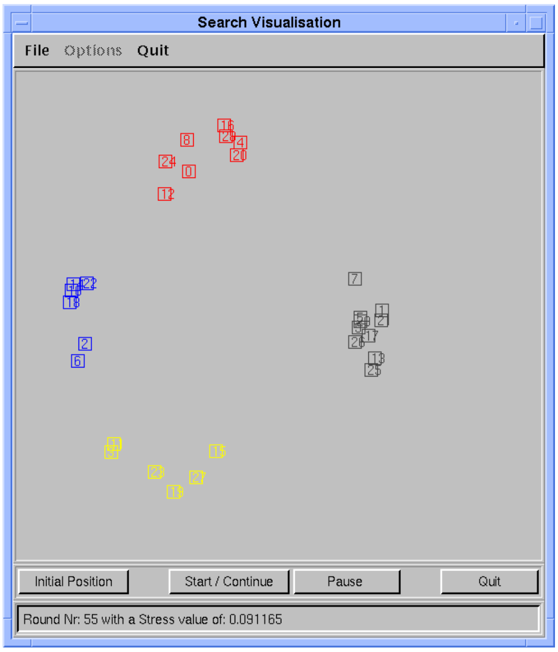

Figure A.2: The final layout for the test file format consisting of thirty search results (Table A.3).

```
!number off objects
30

# GoIDs of the objects and score

0x811b9908 0x000ee197;1.0
0x811b9908 0x00067bcc;1.0
0x811b9908 0x000680a2;0.95
0x811b9908 0x000680ba;0.92
0x811b9908 0x000680c9;0.87
0x811b9908 0x001242f8;0.83
0x811b9908 0x000680cd;0.82
0x811b9908 0x0002e6db;0.78
0x811b9908 0x0038b148;0.74
0x811b9908 0x001034ef;0.71

0x811b9908 0x000cac71;0.65
0x811b9908 0x0015d6ab;0.65
0x811b9908 0x0018e5c5;0.64
0x811b9908 0x00125ca2;0.63
0x811b9908 0x002429d2;0.62
0x811b9908 0x0006a158;0.58
0x811b9908 0x0012a896;0.57
0x811b9908 0x000f716f;0.56
0x811b9908 0x000fba76;0.55
0x811b9908 0x00134e1a;0.54

0x811b9908 0x000b38ca;0.46
0x811b9908 0x0013a740;0.43
0x811b9908 0x0036bdca;0.42
0x811b9908 0x001b7ad6;0.39
0x811b9908 0x0013d227;0.28
0x811b9908 0x000023fb;0.22
0x811b9908 0x0014c18e;0.2
0x811b9908 0x00159bb7;0.19
0x811b9908 0x003728d1;0.17
0x811b9908 0x0033cd50;0.05



! Similarities for 30 documents and 4 groups in which documents are similar
! ---------------------------- 30 / 4 -----------------------------------

0.28;
0.18;0.13;
0.15;0.11;0.27;
0.80;0.19;0.12;0.26;
0.19;0.87;0.24;0.20;0.25;
0.23;0.19;0.83;0.17;0.26;0.26;
0.18;0.75;0.11;0.27;0.14;0.81;0.25;
0.82;0.26;0.24;0.05;0.94;0.02;0.11;0.27;
0.14;0.93;0.21;0.26;0.28;0.88;0.25;0.93;0.27;
0.06;0.25;0.80;0.22;0.24;0.13;0.80;0.14;0.03;0.22;
0.18;0.22;0.27;0.87;0.15;0.29;0.23;0.11;0.24;0.21;0.19;
0.80;0.17;0.29;0.06;0.80;0.22;0.18;0.16;0.83;0.22;0.16;0.25;
0.21;0.80;0.23;0.22;0.13;0.80;0.20;0.89;0.31;0.83;0.36;0.21;0.20;
0.05;0.29;0.80;0.25;0.23;0.14;0.80;0.18;0.21;0.02;0.81;0.24;0.17;0.12;
0.15;0.16;0.13;0.80;0.24;0.11;0.18;0.23;0.24;0.23;0.10;0.80;0.10;0.20;0.19;
0.86;0.22;0.26;0.27;0.93;0.10;0.31;0.29;0.85;0.05;0.15;0.22;0.83;0.01;0.12;0.25;
0.20;0.80;0.29;0.04;0.15;0.88;0.25;0.85;0.14;0.81;0.26;0.17;0.13;0.97;0.19;0.27;0.23;
0.29;0.25;0.86;0.13;0.01;0.13;0.87;0.29;0.27;0.13;0.98;0.26;0.11;0.26;0.96;0.16;0.28;0.18;
0.26;0.23;0.17;0.81;0.17;0.12;0.18;0.21;0.23;0.22;0.14;0.80;0.25;0.23;0.19;0.80;0.26;0.15;0.18;
0.83;0.25;0.14;0.29;0.80;0.11;0.21;0.26;0.80;0.26;0.03;0.30;0.89;0.29;0.29;0.03;0.83;0.08;0.25;0.17;
[...]
```

Table A.3: The test file used for thirty documents and four clusters of similar documents.

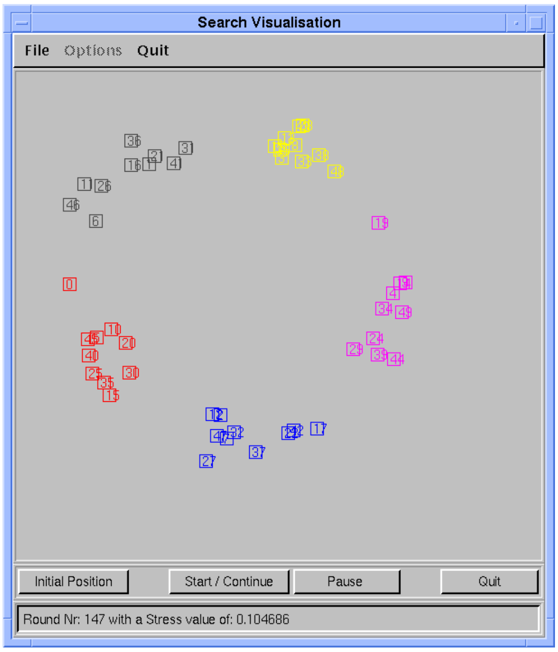

Figure A.3: The final layout for the test file format consisting of fifty search results.

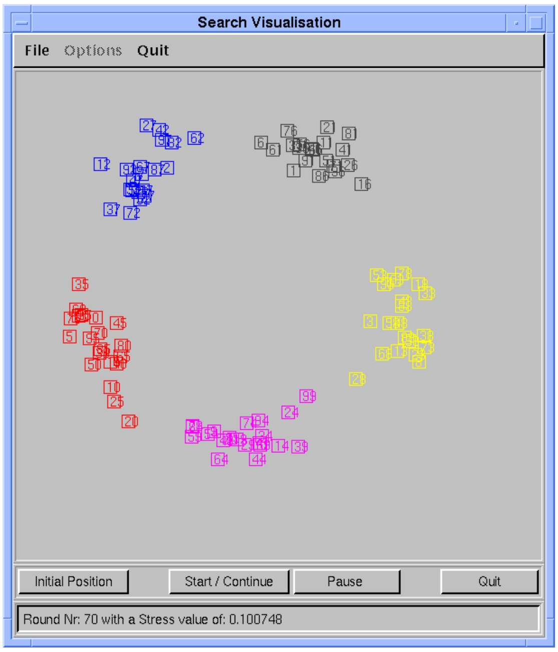

Figure A.4: The final layout for the test file format consisting of hundred search results.

Bibliography

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

[And94] Keith Andrews. Multimediale informationssysteme (lecture notes), 1994.

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

[Ber] Uc berkeley digital library project. `http://elips.cs.berkeley.edu/tilebars.html`.

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

[BSG+] Steve Benford, Dave Snowdon, Chris Greenhalg, Bob Ingram, Ian Knox, and Chris Brown. Vr-vibe: A virtual environement for co-operative information retrieval. `http://www.crg.cs.nott.ac.uk/crg/Research/pits/vrvibe/`.

[CC92] Matthew Chalmers and Paul Chitson. Bead: Explorations in information visualization. In *Proc. SIGIR'92*, pages 330–337, Copenhagen, June 1992. ACM. `http://www.ubilab.ubs.ch/paper/Cha92.ps.gz`.

[Cha93] Matthew Chalmers. Using a landscape methaphor to represent a corpus of documents. In *Proc. European Conference on Spatial Information Theory, COSIT '93*, pages 337–390, Elba, September 1993. `http://www.ubilab.ubs.ch/paper/Cha93.ps.gz`.

[Cha96] Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *Proc. IEEE Visualisations '96*, San Francisco, 1996. `http://www.ubilab.ubs.ch/paper/Cha96a.ps.gz`.

[DM] Nikos Drakos and Ross Moore. The latex2html translator. `http://www-dsed.llnl.gov/files/programs/unix/latex2html/manual/`.

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

[Fla96] David Flanagan. *Java in a Nutshell: A Desktop Quick Reference for Java Programmers*. O'Reilly & Associates, March 1996.

[GJS96] James Goslin, Bill Joy, and Guy Steel. *The Java Language Specification*. The Java Series. Addison-Wesley, 1996.

[Gro] UK TeX Users Group. TeX frequently asked questions. `http://http://www.cogs.susx.ac.uk/cgi-bin/texfaq2html?introduction=yes`.

[GY96a] James Goslin and Frank Yellin. *The Java Application Programming Interface Volume 1: Core Packages*. The Java Series. Addison-Wesley, 1996.

[GY96b] James Goslin and Frank Yellin. *The Java Application Programming Interface Volume 2: Window Toolkit and Applets*. The Java Series. Addison-Wesley, 1996.

[Hea95] Marti A. Hearst. Tilebars: Visualization of term distribution information in full text information access. In *Proc. CHI'95*, pages 59–66, Denver, Colorado, May 1995. ACM. `http://www.acm.org/sigchi/chi95/Electronic/documents/papers/mah_bdy.htm%`.

[Hem93] Matthias Hemmje. Lyberworld - eine 3d-basierte benutzerschnittstelle fuer die computerunterstuetzte informationssuche in dokumentenmengen. In *Der GMD-Spiegel 1'93*, January 1993.

[HKW94] Matthias Hemmje, Clemens Kunkel, and Alexander Willet. Lyberworld - a visualization user interface supporting fulltext retrieval. In *Proc. SIGIR'94*, Dublin, Ireland, July 1994. ACM. `ftp://ftp.darmstadt.gmd.de/pub/VISIT/papers/hemmje/SIGIR94.ps.gz`.

[HUG] TeX-Service HRZ Universitt Giessen. Kochbuch fuer latex. `http://bkfug.kfunigraz.ac.at/www/texhelp/cookbook/cookbook.html`.

[Inf] Infoseek - ultra search engine. `http://ultra.infoseek.com`.

[IPSa] GMD IPSI. Demos of the lyberworld project. `http://www-cui.darmstadt.gmd.de/visit/Activities/Lyberworld/Demos/`.

[IPSb] GMD IPSI. Lyberworld - ir based information system. `http://www-cui.darmstadt.gmd.de/visit/Activities/Lyberworld/description%.html`.

[KD95] Helmut Kopka and Patrick W. Daly. *A Guide to LaTeX 2e*. Addison-Wesley, second edition, 1995.

[Kro94] Ed Krol. *The Whole Internet: User's Guide and Catalog*. O'Reilly & Associates, second edition, April 1994.

[Lyc] Lycos search engine. `http://www.lycos.com`.

[Mau96] Hermann Maurer. *HyperWave: The Next Generation Web Solution*. Addison-Wesley, May 1996. `http://www.iicm.edu/hgbook`.

[Met] Metacrawler search engine. `http://www.metacrawler.com`.

[NFH+96] Lucy Terry Nowell, Robert K. France, Deborah Hix, Lenwood S. Heath, and Edward A. Fox. Visualizing search results: Some alternatives to query-document similarity. In *Proc. SIGIR'96*, pages 67–75, Zurich, Switzerland, August 1996. ACM. `http://ei.cs.vt.edu/papers/ENVreport/final.htm`.

[NP96] Pat Niemeyer and Josh Peck. *Exploring Java*. O'Reilly & Associates, May 1996. `http://www.ora.com/catalog/expjava/`.

[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. In *Information Processing and Management 29 (1)*, pages 61–81, 1993.

[PTV96] W.H. Press, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*. Cambridge Univ Pr, January 1996.

[Rod97] Alexander Rodiga. Harsearch similarity map: Visualising search results using a maximum similarity spanning tree. Master's thesis, Graz University of Technology, Austria, May 1997.

[SM95] Kozo Sugiyama and Kazuo Misue. Graph drawing by the magnetic spring model, May 1995. Academic Press Limited.

[Sno] Dave Snowdon. Vr-vibe screen shots. `http://www.crg.cs.nott.ac.uk/crg/Research/pits/vrvibe/`.

[Spo93] Anselm Spoerri. Infocrystal: A visual tool for information retrieval. In *Proc. Visualization'93*, pages 150–157, San Jose, California, October 1993. IEEE Computer Society.

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

[Wol78] M.A. Wolfe. *Numerical Methods for Unconstrained Optimization*. Van Nostrand Reinhold, 1978.

[Yah] Yahoo search engine. `http://www.yahoo.com`.

[Zwa97] Bernhard Zwantschko. Dino is alive! welcome to the dino project homepage, 1997. `http://www.iicm.edu/dino/`.