Visualisation Islands

Interactive Visualisation and Clustering of Search Result Sets

Vedran Sabol

Visualisation Islands

Interactive Visualisation and Clustering of Search Result Sets

Master's Thesis

at

Graz University of Technology

submitted by

Vedran Sabol

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

October 2001

Advisor:	Univ.Ass. Dr. Keith Andrews
Coadvisor:	Dipl.-Ing. Christian Gütl

Visualisation Islands

Interaktive Visualisierung und Clustering von Suchresultatssätzen

Diplomarbeit

an der

Technischen Universität Graz

vorgelegt von

Vedran Sabol

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

Oktober 2001

Diese Arbeit ist in englischer Sprache verfaßt.

Betreuer:	Univ.Ass. Dr. Keith Andrews
Mitbetreuender Assistent:	Dipl.-Ing. Christian Gütl

Abstract

The amount of knowledge available electronically is increasing exponentially. Huge amounts of information are available over the Internet and searching for a specific topic often results in a large number of matches. A significant portion of hits is often not at all of interest and the retrieved information contains no explicit relations between different hits, making it hard to obtain an overview and find relevant information.

Visualisation is a powerful technique for distinguishing relevant from non-relevant information and for locating information of interest easily and efficiently. This thesis describes Visualisation Islands, a system for topically organising documents returned in a response to a search query according to their similarity. Search results are visualised in the form of an explorable, intuitive, topically organised topographic map, where relationships between documents are encoded by proximity. Topically similar documents are grouped together forming densely populated areas visualised as mountains and labeled with corresponding documents' keywords. These areas are separated by lower areas or water containing less similar objects.

The topical map visualisation is constructed by applying clustering algorithms on documents in vectorised form, creating groups of similar documents, subsequently positioning the documents in the 2-D viewport space according to the similarity of their vectors by using a force-directed placement algorithm, and generating a topographic background image based on computed 2-D document coordinates. To provide platform-independence Visualisation Islands is implemented in Java.

Kurzfassung

Die Menge von elektronisch verfügbarem Wissen steigt exponentiell. Riesige Mengen von Informationen sind im Internet vorhanden und das Suchen nach einem spezifischen Thema resultiert sehr oft in einer grossen Anzahl von Treffern. Ein signifikanter Anteil dieser Treffer ist jedoch nicht von Interesse und die zurückgelieferte Information enthält keine explizite Zusammenhänge zwischen einzelnen Treffern, was den Überblick über die ganze Menge der Suchergebnisse und das Auffinden relevanter Information besonders erschwert.

Visualisierung ist eine mächtige Technik um die relevanten von den nicht relevanten Daten zu unterscheiden und die Information, welche von Interesse ist, leicht und effizient zu finden. Diese Arbeit beschreibt Visualisation Islands, ein System, das die Dokumente, die als Antwort zu einer Suchanfrage zurückgeliefert werden, vektorisiert und abhängig von ihrer Ähnlichkeit thematisch organisiert. Die Suchergebnisse werden in Form einer erforschbaren, intuitiven, thematisch organisierten topographischen Karte visualisiert, wobei Zusammenhänge zwischen den Dokumenten durch ihre Nähe repräsentiert werden. Thematisch ähnliche Dokumente werden gruppiert und bilden dicht besiedelte Bereiche. Diese Bereiche werden mit Schlüsselwortern zugehöriger Dokumente gekennzeichnet und als Berge visualisiert. Die Berge sind von niedrigeren Bereichen oder Wasserflächen, in denen sich weniger ähnliche Objekte befinden, getrennt.

Die thematische Visualisierungskarte wird konstruiert, indem zuerst die Clusteringalgorithmen auf die Dokumente in vektorisierter Form angewendet werden um Cluster von ähnlichen Dokumenten zu bilden. Danach werden die Dokumente, mit Hilfe einer force-directed placement Methode, abhängig von der Ähnlichkeit ihrer Vektoren im 2-D Viewport Raum positioniert. Anschliessend, basierend auf den berechenten 2-D Dokumentenkoordinaten, wird ein topographisches Hintergrundbild generiert. Um Plattformunabhängigkeit zu gewährleisten ist Visualisation Islands in Java implementiert.

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 want to thank my advisor Keith Andrews for his advice, continuous support, attention to my questions, and for correcting the draft versions of this thesis. Additional thanks go to my coadvisor Christian Gütl for helpful suggestions, to all members of the xFIND team for their assistance in technical aspects, in particular to Josef Moser for help concerning Java and xFIND, and to Michael Granitzer und Mathias Lux of KNOW-Center for their help in translating the thesis abstract into German. I would like to thank everyone at the IICM for the friendly cooperation and feedback, and many other colleagues and friends for providing constructive criticism and invaluable help.

I want to express special thanks to my family for standing helpfully by me, for their support and understanding. My most heartfelt thanks are due to those who supported and encouraged me during the last time of my studies as without them this thesis would not have been possible, I am indebted to you. Last but not least, a great thank you goes to my girlfriend for her enduring support and patience.

Vedran Sabol
Graz, Austria, October 2001

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Very large amounts of information are now available electronically over the Internet. Searching for a specific topic often results in receiving a huge number of search results, mostly in unstructured form, and much of the retrieved information may not be of interest. Standard search engines return search query results in the form of a linear list of hits sorted by estimated relevance, but the relationships between different hits are not made explicit. It is a difficult task for the user to obtain an overview and to tell which of the retrieved documents are actually of interest. Visualisation of search result sets can be a powerful technique for distinguishing relevant from non-relevant information and for retrieving it easily and efficiently. This thesis describes Visualisation Islands, a search client that addresses this problem by topically organising documents returned as a result of a search query. Visualisation Islands is an xFIND search engine client which processes documents returned in a response to a search query, organising them according to their content and similarity. The results are visualised in the form of an interactive, intuitive, topically organised topographic map, where relationships between documents are encoded by proximity. Topically similar documents are grouped together forming densely populated areas visualised as mountains or islands, which are labeled with keywords extracted from underlaying documents. These areas are divided by lower areas or water containing less similar objects. The visualisation offers the user a possibility to instantaneously locate areas of interest and explore documents located in those areas.

In Visualisation Islands, a document's content is described by a high-dimensional vector containing relevant keywords returned by the xFIND search engine. A similarity coefficient can be computed for every two vectors making it possible to compare the retrieved documents. When the documents are loaded, groups of similar documents are created by applying a standard clustering algorithm. These groups are used to display the document set in table of contents style. Then, a force-directed placement algorithm with stochastic sampling is used to map the documents from the high-dimensional vector space to a 2-D space by placing similar documents close to each other while dissimilar ones are moved apart. Based on computed 2-D coordinates, a 2-D or 3-D style topographic background is generated, forming mountains at areas where document density is high. The mountains are divided by areas with lower document density which are represented as lower areas or water, resulting in a map containing islands of topically similar documents. Document icons are displayed atop the generated background and the mountains are labeled with keywords from the underlaying documents, producing a intuitive, topically organised landscape (or seascape). It is easy for the user to locate areas of interest and explore the document set with operations such as zooming, searching, filtering, selecting, and viewing details and contents of documents of interest. The user can explore and process multiple sets of search results simultaneously and can reprocess any subset of the original set for more detailed insight. Visualisation Islands is written in Java providing cross-platform portability.

This thesis is organised in two parts. The first part consists of Chapters 2 to 5 which introduce and discuss the theoretical basics and describe similar approaches and systems. Chapters 6 to 10 describe the Visualisation Islands system, the design decisions, the implemented algorithms, and show the program in action. Appendices A and B constitute a user guide for beginning and advanced users.

Chapter 2 covers the field of Information Retrieval (IR), describing how information in human-readable form, such as HTML documents, can be analysed, organised, and searched for. Indexing of documents is described and concepts such as the vector space model and similarity measures are introduced. An overview of search engines and search engine types is also given.

Chapter 3 focuses on clustering algorithms, describing basic clustering concepts and a number of methods for organising sets of objects into clusters of similar objects. Several partitional, hierarchical and artificial intelligence methods are described, analysed, and compared. Scatter/Gather, a system which uses clustering methods for browsing large and very large document collections is also presented.

In Chapter 4, general principles of visualisation are given. Several document visualisation systems are presented and discussed. A closer look is taken at how visualisation concepts can be applied to improve the process of searching and presenting retrieved document sets. The presented systems are subdivided into the following groups: systems showing a compact representation of documents' content, systems visualising multidimensional data, systems supporting the user in building a search query, systems visualising document hierarchies, systems displaying the hyperlink structure surrounding documents and systems visualising document collections according to their content. The emphasis is on the last group of systems.

Chapter 5 describes xFIND, a scalable, distributed search engine which, in response to a search query, supplies Visualisation Islands with retrieved documents and their keywords. The advantages of a distributed approach over a conventional centralised approach are presented. The Search Results Explorer tool for visualisation of multi-dimensional data is presented. It is used as an xFIND client for visualisation of documents' meta-data.

The theoretical basics described in Chapters 2 to 5 were used as a foundation in the development of Visualisation Islands and had a strong impact on many design decisions. They are also the key to understanding how Visualisation Islands operates.

Chapter 6 describes the software architecture of the Visualisation Islands package. The motivation for development of such a system is presented and subsequently a description of the process of generating the visualisation is given. The object-oriented architecture of the program is described and the responsibilities and interactions between all packages and classes are specified.

Chapter 7 concentrates on the algorithms implemented in Visualisation Islands. Design decisions and a detailed description and analysis of the three processing steps for constructing the visualisation is given. The algorithms (or classes of algorithms) described are: clustering of the retrieved documents using single-pass, k-means, or hierarchical agglomerative clustering algorithms; mapping the documents from the high-dimensional term space to the 2-D viewport space using a force-directed placement algorithm; generation of a 2-D or 3-D style map background image.

In Chapter 8, Visualisation Island is demonstrated in practice. Several examples and typical use cases are presented and advantages of the systems are discussed.

Chapter 9 outlines ideas for further research and future extensions of Visualisation Islands.

Chapter 10 briefly describes the motivation for development of Visualisation Islands and gives an overview of implemented techniques. The content of this thesis is summarized.

Appendix A is a user guide for Visualisation Islands, all program functions and user interface interactions are described, with the exception of advanced parameters which are not intended for modification by a standard user.

In Appendix B advanced parameters and their effects on processing results are described. These parameters should only be modified by an expert user.

Chapter 2

Information Retrieval and Searching

The Internet and the World Wide Web are developing to a primary media of the future. As human knowledge is increasing and the volume of electronically available information is growing, an important question arises: how can the user find information? Together with this question the importance of Information Retrieval (IR) has been growing. Although it is not easy to give an exact and complete definition of Information Retrieval, this question leads us to attempt a following definition: Information Retrieval are procedures and technologies for finding the desired information in files saved in electronic form, or, trying to be more exact, Information Retrieval is the study of systems and technologies for indexing, searching, and recalling data, mostly in the form of text. Therefore, in this chapter techniques for identification and retrieval of matching documents from an information base in response to a query will be discussed. As the amount of information stored in Internet is already huge and is growing rapidly, it is very important that the documents returned as a result of a query are highly relevant and not just distantly related to the area specified by the query. Due to the fact that the response of a IR system is based on statistical methods, some relevant documents may not be returned while some other, much less or non-relevant are returned instead. It is a difficult and complex task to identify and return only the information which is of interest, and the variety of technologies used in this area is large.

Distinguishing between a Data Retrieval system and an Information Retrieval system gives a better understanding of what an IR system should be capable of (see Table 2.1). These differences are caused by the fact that Data Retrieval Systems, for example relational Data Base Management Systems (DBMS), work with structured records with a specified set of fields with exactly defined contents. The retrieval model used in this case is a Boolean model. The information is retrieved only if an exact match between the query and specified record fields occurs. In contrast, IR systems consist of documents that do not have an exactly defined internal structure, each having have attributes such as title, author, and keywords. The key to retrieving documents in such a system is to build an index based on the attributes, or based on the whole document content, the so-called full text index. In both cases a match in an IR system is only partial, depending on the similarity between the query and the contents of the index. Similarities between different documents in the index can also be determined as well as retrieval effectiveness.

2.1 Indexing

Looking for each query term in every document by sequential scanning is very slow and therefore possible only for a small number of documents. To organise the documents so that they can easily be found is not an easy task. The oldest approach is to assign the documents to thematic categories,

	Data Retrieval	Information Retrieval
Matching	exact	partial, best match
Inference	deduction	induction
Model	deterministic	probabilistic
Classification	monothetic	polythetic
Query Language	formal	natural
Query Specification	complete	incomplete
Searched Objects	fulfilling the search specification	relevant
Reaction on Data Errors	sensitive	insensitive

Table 2.1: Data Retrieval vs. Information Retrieval.

but this approach was not flexible and powerful enough. Several problems exist with this approach including the creation, definition and the representation of categories, how specific or general they should be, how the documents should be assigned to categories and how the categories should be searched and browsed. Many of these problems can be solved by the use of indexing. A great advantage of this approach is that the index can be generated automatically.

Indexing is the key to efficient searching in a system with a large number of documents. A standard index is a matrix, also called document term matrix, where rows represent documents and columns represent terms. Therefore, the rows of the matrix are called term vectors. In the simplest representation an entry of 1 in the matrix means that a term is present in the document and a 0 means that it is not present. To construct an index all documents have to be analysed in a step called content analysis, where the each document is scanned to extract some content, attributes or terms, resulting in a indexed representation of the scanned documents. Two kinds of indexes can be constructed, depending on what is indexed:

- Attribute index: the attributes of the IR system are indexed, such as title, authors, and keywords. This type of index is smaller, as well as easier and faster to generate, however it does not represent every aspect of the scanned documents, as many words contained in the documents are simply lost. The decisions regarding the extraction of significant words that characterise and discriminate a document are mostly based on statistical procedures, but also some artificial intelligence approaches can be used.

- Full text index: every word in every document of the IR system is indexed and almost every aspect of the scanned documents is therefore taken into consideration. Such an index is typically very large, but fortunately there exist a number of techniques for compressing it without loosing any of the collected information.

2.1.1 Inverted Index

The standard index representation is not very suitable for searching because it takes a long time to find out if a term is present in a document or not, as the system must scan all term vectors. To make the searching process fast and efficient the matrix is inverted. The inverted index is a collection of terms, each pointing to those documents that contain it. In an inverted index rows represent the terms and columns represent the documents. The rows of the matrix are called document vectors in this case. This structure is very suitable for effectively determining which documents contain the search query terms. The rows do not have to be scanned in this case as they contain by definition all documents containing the desired term. The retrieval model used in this case is called the Boolean model. Queries are composed of keywords that characterise the information that the user is looking for, and can be combined with the use of Boolean expressions. By comparing the query terms with the terms in the index matching documents are easily identified. However, it is not possible to calculate the estimated relevance and therefore the documents that match the query can not be ranked with respect to the search query.

2.1.2 Content Analysis and Term Operations

When a document is sequentially scanned the words and phrases that it is composed of are identified and their occurances can be statistically analysed. On the basis of this analysis keywords, which describe the document best and discriminate it from other documents, can be identified and extracted. Also, the place of appearing of terms inside of a document, for example in the title or at the beginning of the document, is an indicator of the their significance. HTML allows words in bold or italics to be identified and given a higher weight. The relative position of words in a sentence, and their significance, give a measurement for determining the significance of sentences. Thus, frequency data can be used to extract words and sentences to represent a document and automatically create summaries.

Stemming

Stemming means removing common suffixes as “ion”, “ed”, and “ing” from the terms in the processed document and in the search query, leaving them in their basic form where tense and plurality are lost. In this way many terms are replaced by only one term in its root form, reducing the index size a lot. This is a very powerful technique but it has to be used carefully, because it can introduce ambiguity and produce undesired effects if it is not used correctly. Detailed knowledge of the document language is required for correct and effective stemming, and correct decisions have to be made concerning the question if all parts of the document and all its attributes should be stemmed.

Case Folding

Case Folding is a technique that avoids the problem of search failures due to case sensitivity and reduces the size of the index considerably. Every letter of each term is converted to its lowercase equivalent, both for the terms in the document term matrix and for the terms in the search query. This is a simple and effective procedure, but imposes a problem if a case sensitive search is required. In this case a separate document scan is needed where exact pattern matching is performed.

Thesaural Substitution of Synonyms

There are many words that are different but have the same meaning. Searching for one of them would miss the others. To overcome this problem and also to reduce the size of the index, these synonyms can be replaced, in the documents and in the search query, by a single term having the same meaning. This is achieved by the use of a thesaurus. Spelling differences between different English versions should also be taken into consideration at this point (for example visualization vs. visualisation).

Term Weighting

When using an index the simplest way to indicate the presence of a term in the document is to use a 0 for not present and a 1 for present. A more advanced solution is using weights for the terms, for example a high number of points it the term is in the title, a medium if it is a keyword and a small number of points if it occurs in the text. The sum of the points gives the term weight, giving a possibility to rank the documents relative to a specific term or to the search query. It is also possible to give a weight to search terms, giving a fine control to the user. On the other hand query construction and matching becomes more complicated.

There are two ways to represent the term weight in the index:

- relative term weight $= \dfrac{\text{term weight}}{\text{sum of all term weights in the document}}$
- normalised document vector

2.1.3 Keyword Relevance Filtering

Each document contains a number of words that occur more or less often in the text, but have no meaning for finding the information and can not be used to discriminate between different documents. Removing such words can reduce the size of the index significantly and improve the quality and speed of the retrieval process.

Stop List

There is a number of words that occur very often in every English document which do not contribute in discriminating between documents but increase the size of the index significantly and therefore should be eliminated. These words are, for example:

- Articles: “the”, “a”, “an”
- Conjunctions: “an”, “or”, “also”, “that”, ...
- Prepositions: “in”, “to”, “out”, ...
- Modal verbs: “have”, “be”, ...
- Questions: “what”, “why”, “who”, “where”, ...

A commonly used approach is to collect them in a list called a stop list. While scanning the document each identified word is compared to the entries in the stop list and is eliminated if it is found to be present. This procedure can reduce the size of the inverted index up to 40 percent, but if the index is compressed, which is mostly the case due to its huge size, the benefit of the stop list is much smaller. This is due to the fact that the words in the stop list are very common and therefore have a high compression ratio in the compressed index. Eliminating certain common words using a stop list can still be a very good idea as they, in general, do not help in discriminating between documents, and can even have an opposite effect, making it more difficult and time consuming to calculate the similarity between documents. In some cases omitting words can cause certain problems, for example if users want to find exactly those words, or they are the main terms of the document, or because they can change the meaning of surrounding words. Consequently a good stop list filter can be difficult to build.

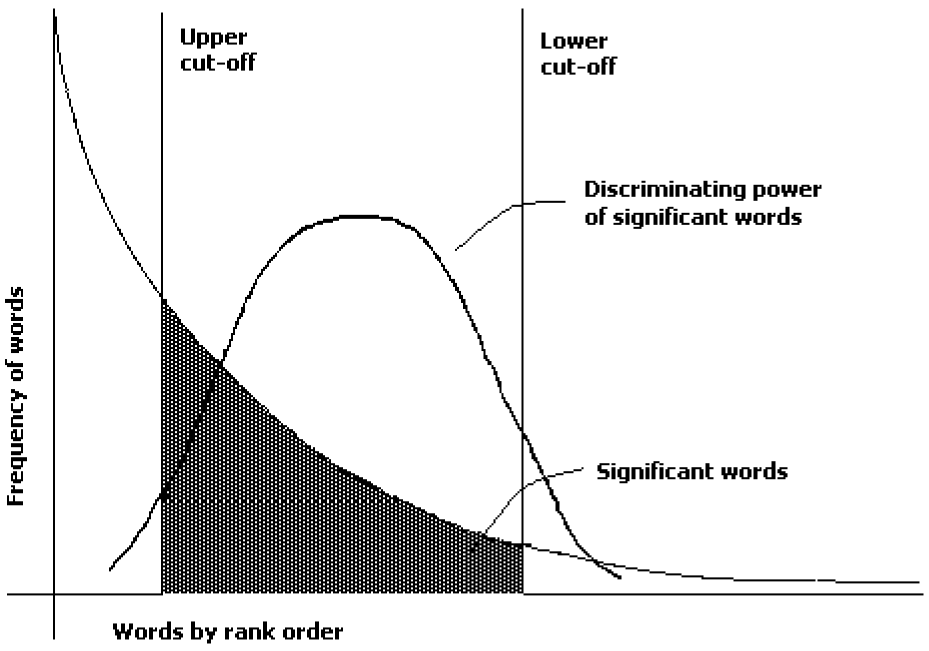


Figure 2.1: Discriminating power of significant words compared to their frequency.

Frequency Based Filtering

The frequency of occurrence of different words in a given position of text, f, and their rank order, r, that is, the order of their frequency of occurrence, give a curve similar to the hyperbolic curve, demonstrating Zipf's Law which states that the product $f * r$ is approximately constant. It is possible to specify an upper and a lower cut-off that exclude non-significant words. The words exceeding the upper cut-off are considered to be too common and those below the lower cut-off too rare, and therefore these words can not contribute significantly to the content of the document [LUH59]. The fact that words with very low frequency do not characterise a document well is quite clear, but even those words occurring more often are sometimes not good discriminators: for example a word like "space" is surely not a very good discriminator in a context of a NASA server. It can be generally assumed that the ability of relevant words to discriminate content of the articles containing them reaches a peak at position half way between the two cut-offs falls, in both directions, reaching a value of almost zero at the cut-off points. There is no general rule for determining the cut-off positions and they are mostly based on testing and experience.

2.1.4 Identifying Indicators of Document Content

One of the biggest challenges of document analysis and automatic indexing is finding those words or phrases which express the content of a document. Although by taking into consideration whole phrases the content of a document can be expressed better than with single words stripped of their environment, for reasons of simplicity only single words will be taken into consideration as content indicators. Each term T_j extracted from document D_i is given a weight, w_{ij}, and only words with a weight exceeding a specified threshold are considered good for representing the document's content. After completing the steps discussed in previous sections of this chapter, the term frequency tf_{ij}, of every term T_j in the document D_i is calculated. As already said, this value alone is not a reliable indicator, because words which occur too often are not good discriminators. Based on tf_{ij} and some other values good measures for suitability of words as discriminators and indicators of document content can be computed. Examples include (taken from [GRG97]):

- Discrimination value, dv, based methods take into consideration the frequency of the keyword in a document as well as the keyword's discrimination power to calculate w_{ij}. Each of the N documents can be represented as a point in a high-dimensional document space. The dimensions of this space are the words which the documents are composed of. The distance between points is a measure of dissimilarity between documents, similar documents are located close to each other, dissimilar documents are located far apart. Regions of the high-dimensional space where many similar documents are located have higher densities than regions populated by less similar documents. If a term T_j is assigned to all documents in the set, average distance of the documents, and with it the average density, changes to some degree. Discrimination value of the term T_j, dv_j is defined as the difference of the document density before and after the assignment of the word to all documents. A higher value of dv_j means that the word is a good discriminator. The value of w_{ij} is calculated as :

  $w_{ij} = tf_{ij} dv_j$

- Inverse document frequency based methods take into consideration the frequency of occurrence of a term T_j in a document D_i, tf_{ij}, as well as in how many documents the term occurs in, which is denoted by df_j. The inverse document frequency is a measure describing how well a term can be used for discriminating between documents. It is defined as:

  $idf_j = \log(\frac{N}{df_j})$

  where N is the number of documents. The value of w_{ij} is calculated as :

  $w_{ij} = tf_{ij} idf_j$

2.1.5 Measuring Retrieval Effectiveness

The standard measures for estimating the retrieval effectiveness are Precision and Recall. Precision is defined as the number of relevant documents retrieved divided by the number of all retrieved documents. Recall is defined as the number of relevant documents retrieved divided by the number of all relevant documents in the collection. Both have a value between 0 and 1 and are inversely related. It is a simple task to achieve one of these requirements at a time, retrieving only one document for maximum Precision or all documents for maximum Recall, so a system should attempt to maximize both precision and recall at the same time. If the set of retrieved documents is denoted by F and the set of all relevant documents in the whole collection as R, than Precision and Recall are defined as:

Precision: $P = \frac{|F \cap R|}{|F|}$

Recall: $R = \frac{|F \cap R|}{|R|}$

To evaluate the retrieval performance using both values at once a combined measure is needed like [FBY92]:

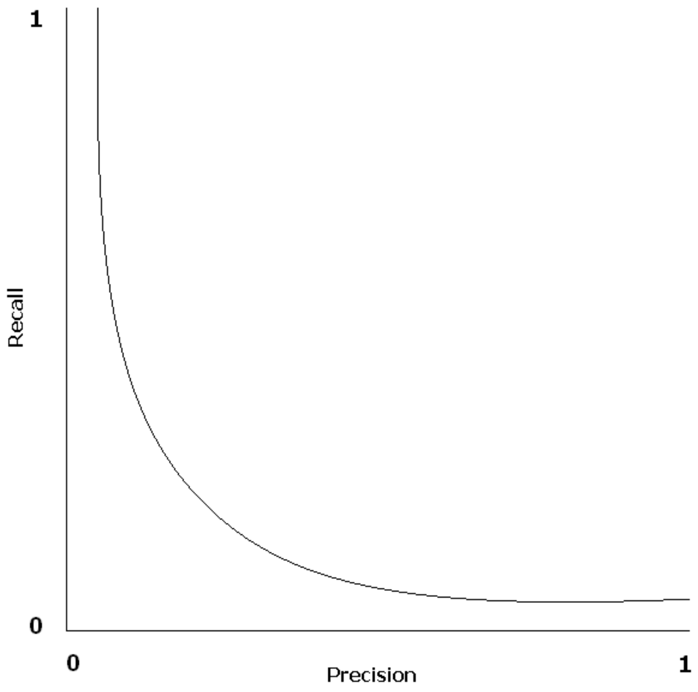


Figure 2.2: Precision and Recall [FBY92].

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

where b is a measure of relative importance of precision and recall.

2.2 Vector Space Model

To construct a Vector Space Model [SYW75] a standard index has to be constructed. Each document is represented by a row in the document term matrix, which is actually a term vector consisting of zeros and ones to indicate if a particular term is present or not. In this way documents are represented as points in the document space, which is a multidimensional space, where each coordinate represents a unique keyword. The vector space model can be easily used in combination with the inverted index which is also constructed from the standard index by inverting the document term matrix .

As each document is assigned a vector and represented as a point in the document space, a possibility is created to define a measure of similarity for a pair of documents. A query can also be represented as a vector of terms in the space. Retrieved documents can be ranked according to their similarity values with respect to the search query. Furthermore, through the possibility to compare each pair of documents, a way is given to clusters documents in topically similar groups.

2.2.1 Proximity Measures

Proximity measures can be divided into similarity measures that express the similarity of a pair of documents as a number, and distance measures, which actually measure the dissimilarity between documents. The smaller the distance of two points in this space is the more similar the contents of the corresponding documents. Values used in the following definitions are: s is the similarity coefficient,

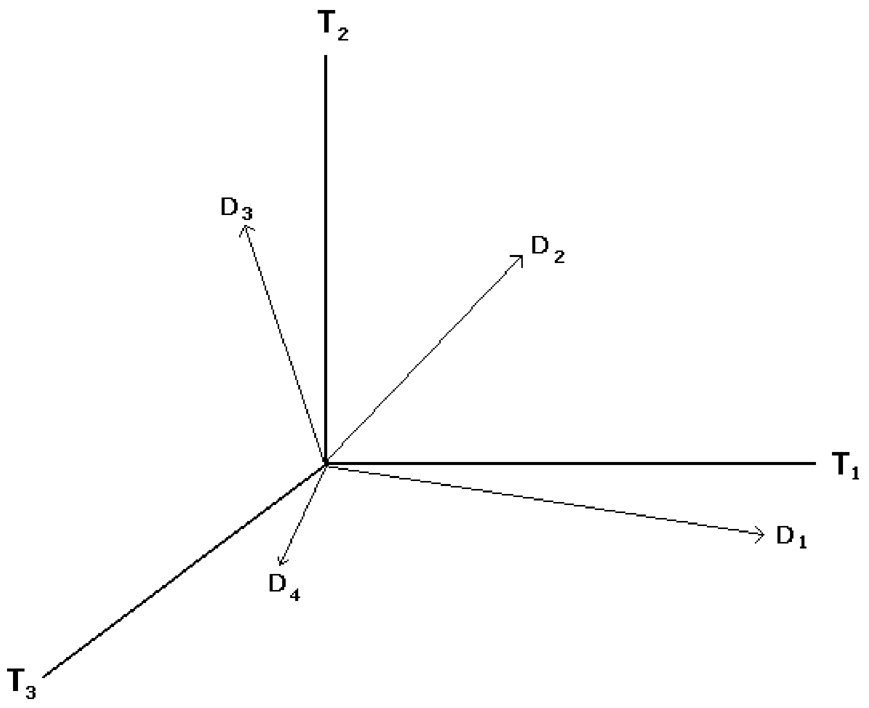


Figure 2.3: Vector Space Model with the dimensionality of 3

d is the Euclidean distance, D_i and D_j are the compared documents, k is the coordinate, $x_{i,k}$ and $x_{j,k}$ are the values of the k-th coordinate of the appropriate document and L is the dimension of the document space.

Distance measures are for example [RIJ79]:

- Euclidean Distance: $d_{D_i,D_j} = \sqrt{\sum_{k=1}^{L} (x_{i,k} - x_{j,k})^2}$

- City Block-, Manhattan, or Taxi Driver-Metric: $d_{D_i,D_j} = \sum_{k=1}^{L} |x_{i,k} - x_{j,k})|$

which are both special cases of Minkowski-Metric: $d_{D_i,D_j} = (\sum_{k=1}^{L} (x_{i,k} - x_{j,k})^n)^{\frac{1}{n}}$

To apply such distance measures the vectors should be normalised. Similarity coefficients do not require this and are therefore particularly good for the comparison of document vectors. Such coefficients are:

- "Dice" Coefficient: $s_{D_i,D_j} = \frac{2\sum_{k=1}^{L}(x_{i,k}x_{j,k})}{\sum_{k=1}^{L} x_{i,k}^2 + \sum_{k=1}^{L} x_{i,k}^2}$

- "Jaccard" Coefficient: $s_{D_i,D_j} = \frac{\sum\limits_{k=1}^{L}(x_{i,k}x_{j,k})}{\sum\limits_{k=1}^{L}x_{i,k}^2 + \sum\limits_{k=1}^{L}x_{i,k}^2 - \sum\limits_{k=1}^{L}(x_{i,k}x_{j,k})}$

- "Cosine" Coefficient: $s_{D_i,D_j} = \frac{\sum\limits_{k=1}^{L}(x_{i,k}x_{j,k})}{\sqrt{\sum\limits_{k=1}^{L}x_{i,k}^2 \sum\limits_{k=1}^{L}x_{i,k}^2}}$

2.3 Search Engines

Search Engines are used for searching the World Wide Web (WWW) for documents and resources. To perform this task they use a spider, also called a robot or a crawler, to browse the Web following the hyperlinks. These programs navigate the structure of the Web, which is given through the hyperlinks, to collect documents. Information like metadata, content, and structure is extracted from the documents, organised in a form of an index to allow fast searching, and presented as a response to a user query, mostly in the form of a linear list sorted by estimated relevance. Automated methods for indexing documents are neccessary as the number of documents in the Web is much too large for human beings to do the job. However, search engines have a number of shortages. Query formulation is a difficult task as the exact terminology might not be known to the user, but search engines hardly support the user in creating and refining the query. The precision of the search engines is often low, resulting in a great number of non-relevant documents. The presentation of results in the form of linear list sorted by relevance gives no possibility to differentiate between relevant and non-relevant results. Also, much of the information collected by the search engine, like the author, publisher, date of publication, keywords and similar, is not presented to the user. Finally, the number of returned results can be huge, and the search engines do not offer tools that would allow the user to organise the results and get an overview. As the number of document available in Internet is rising rapidly, these problems will become ever more acute. Despite all this, search engines are the only universal tools for locating information in the Web, and can be a powerful means of information retreival if their strengths can be used to the maximum and ther shortcomings avoided, which is also the purpose of this thesis.

2.3.1 Search Engine Types

The number of search services in the Internet is large and each system has a number of special features. The biggest differences between them arise from the method of gathering the information. Some operate completely automatically, others rely on humans to classify the Web content, and some use other search services to gather the results.

- A general use Search Engine is a database system designed for exploring and indexing the Internet. A special program called a spider, or robot, or crawler follows the URLs and retrieves files found on different servers in Internet. The collected files are analysed for contents and attributes and stored in a database structure that enables fast and precise searching in response to a user query. The process of crawling through the Web, collecting information, and organising it into an index is completely automated. Different serch engines use different approaches to extract information from collected documents, different data structures to organise it, and compute the relevance of documents with respect to the user query in different ways. The methods and strategies used affect greatly the quality of returned results, the speed, and the number of documents indexed by the engine. The biggest problem probably is the growth of the Internet and the huge amount of documents present in the Web, so that search engines typically cover only a small portion of available documents. Another problem is how to deal with dynamically generated content. Most of the well-known search tools fall in this category, probably everybody will recognize some of the following names: Fast (All The Web), Northern Light, AltaVista, Google!, Excite, Go (Infoseek), WebCrawler, HotBot, GoTo.

- Meta-Search Engines, like Dogpile, Mamma, Metacrawler or SavvySearch, are not “real” search engines as they do not maintain a database of collected information, but use other engines to do the job. In a response to a user query they send requests to a number of “real” search engines and combine the results. As each search engine can scan only a small portion of Internet, combining results from more of them increases the number of documents taken into consideration. On the other size it is very hard to give each document a relevance rating since all search engines have different methods of calculating it.

- Subject Directories, like those offered by Open Directory, Look Smart, Yahoo, Lycos and Magellan, rely on humans to organise Internet contents into categories based on their subject. The user can choose a cathegory of interest and is offered a list of resources to browse. By choosing a series of progressively more narrow search terms that are offered as descriptions of the list, the user finally obtains a list of resources that meet all chosen descriptions. Subect directories cover a much smaller portion of the Internet than the serach engines, but can offer high precision and relevant results, since they were built by humans.

- Link Guides are sets of hyperlinks on a subject, created and maintained by experts and hobbyists familiar with the topic. Such subject guides guarantee high quality, but have a disadvantage of the user first having to find the right guide on the topic of interest. Guides to link guides exist to support the user and overcome this problem.

- Intelligent Web Agents are search tools that provide information about structure, environment and content of documents and act as an intelligent aid learning from user habits, suggesting documents that are similar to those already identified as interesting, or that were accessed very often in the past.

However, it should be emphasised that this division is not so strict any more. Many search engines use results of other engines or subject directories to improve the quality of returned results. Some systems, like Inktomi, do not offer direct services to the user at all, but are designed to power other search services, others, such as Northern Light and Vivisimo perform clustering of search results (see Chapter 3).

2.3.2 Estimated Relevance Calculation

To order the documents found in a response to a user query the search engine must provide means to calculate the estimated relevance of each document with respect to the search query terms. The simplest way to achieve this is to count the occurrences of each search term in the document. A significant improvement is brought by checking where and how the search terms appear in the document. Terms appearing in the title, at the beginning of the document or in bold should be given a higher weight. This method, called location/frequency method, is used by many search engines, however in diferent variations. Another criteria for determining the relevance is the number of links pointing to a document. It is assumed that important documents will be referenced more often than those less

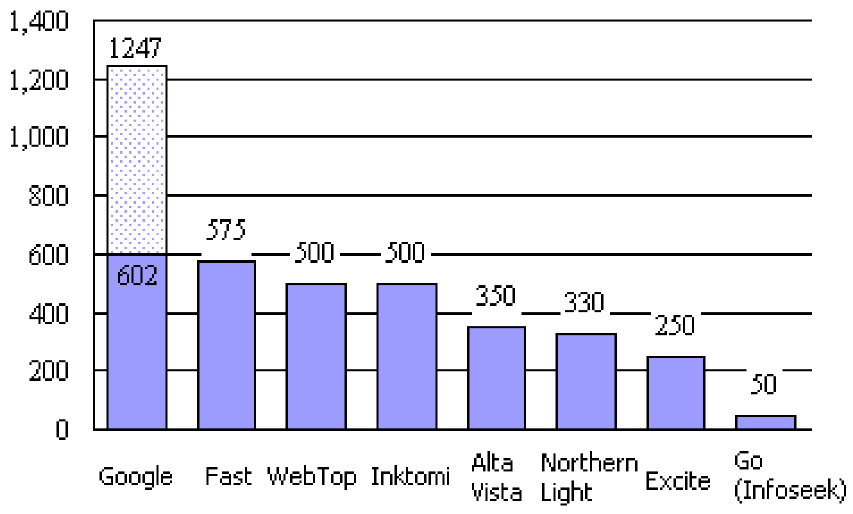


Figure 2.4: Millions of Web pages indexed by popular search engines as of November 1st, 2000. [SEW01]. Google indexed 602 million pages, but because of the way that Google handles the link data, it can return pages it has not actually visited, giving it a coverage of almost 1.25 billion pages. Taken from [SEW01].

important. Metadata could also be used to improve the calculation of the estimated relevance, but they are rarely used as they are prone to manipulation. Some search engines exclude documents from the index if they detect some kind of manupulation that should push the page higher in the list, for example repeating a certain word very often to increase its frequency. The frequency of access to a document is another property that can be used for calculating the relevance, which is often used by intelligent web agents.

2.3.3 Search Engine Features and Characteristics

Search engines may seem very similar to the user at the first look, but each of them operates more or less diferently and offers some features that should improve the quality of results. Here is a list of features and options offered by different search engines:

- Number of indexed documents. Can be larger than 10^9 in the case of Google and Wise Nut.
- Percentage of dead links returned by the system. Typically between 1% and 15%, depends on the actualisation frequency.
- Boolean capabilities: Boolean operators like AND, OR, NOT as well as the used of brackets supported in the advanced search. Even fuzzy operators are possible (Excite).
- Proximity operators for search terms such as ADJ, (adjacent) and NEAR available in the advanced search.
- Stemming, plural/singular conversions, stopword removal.
- Switchable case sensitivity.

- Limiting the search according to some criteria like language, date, document type, domain and even subject.
- Sorting the result to some criteria other than relevance, like date, size or site.
- The ability to return more similar documents after an interesting document was found.
- Classification of search results, supported by Northern Light.
- Support for concept searching for narrowing down the search, as provided by Excite.
- The ability to search only selected fields like the title or URLs.
- Full text search.
- Exact phrase search.
- Duplicate detection.
- Search refining by suggesting more words.
- Presentation of summaries of the results.

Chapter 3

Clustering Techniques

3.1 Overview and Basic Principles

Clustering is a name for methods, statistical or artificial intelligence based, which form categories of objects from unorganised sets of objects. Objects, which are assigned to the same category, or cluster, must be similar according to a certain criteria, while objects that are different must be assigned to different clusters. The underlying hypothesis for the use of clustering in document retrieval can be stated as follows: closely associated documents tend to be relevant to the same requests. This hypothesis is known as the Cluster Hypothesis. Clustering algorithms are able to structure large quantities of documents and to distinguish relations between them. This procedure is similar to the automatic classification of documents, with the difference that in the case of classification, also known as supervised clustering, the categories are already known before processing, while in the case of clustering, also known as the unsupervised clustering, the categories are created dynamically during the processing. Documents are mostly categorised thematically on the basis of their content, which means depending on the keywords and terms they contain, but the grouping can also take place according to any other criteria.

Clustering algorithms can be applied to almost any kind of data, not only documents. A single data item is typically a vector of dimensionality L where its scalar components are called attributes. To apply clustering methods on a set of documents a way has to be found to express the similarity of a pair of documents as a number. Such functions are called similarity functions and were discussed in Chapter 2.2.1. Different clustering methods have been successfully used with images, as well as with all kinds of multidimensional data. The number of approaches and different principles used is very large and continuously new methods are being developed, each having different characteristics tuned for application in specific areas. In this chapter the most common algorithms and principles used for clustering of documents are explained.

3.1.1 Definitions

For better understanding all concepts, values and notations are described here in a short overview:

- A data item or object D is a vector of dimensionality L with the form:

  $D = \vec{x} = (x_1, x_2, \ldots, x_L)$

  where the scalar components x_i, $i = 1 \ldots L$ are the vectors attributes. In this thesis the data items are documents.

- A set of N vectors is denoted by:
  $S = \{D_1, D_2, \ldots, D_N\}$
  which is often viewed as a $N \times L$ matrix.
- A cluster containing n objects can be viewed as a set:
  $C = \{D_1, D_2, \ldots, D_n\}$
- A set of M clusters is denoted by:
  $CS = \{C_1, C_2, \ldots, C_M\}$
- Different similarity measures s as well as a distance measure d were defined in chapter 2.2.1.
- A fractional degree of membership, used by fuzzy clustering methods, of the object D_i to the cluster C_j is denoted with u_{ij}.

3.1.2 Characteristics of Clustering Algorithms

Clustering methods can be distinguished according to a number of different characteristics and criteria. Here is a short overview:

- Partitional vs. hierarchical: Clustering methods are usually divided according to the cluster structure which they produce. The partitional methods divide a set of N documents into M clusters, producing a “flat” structure of clusters. In most cases no overlaps are permitted, which means that each object belongs to only one cluster that it is most similar to. The more complex hierarchical methods produce nested pairs of clusters or documents by interconnecting them gradually, until each document is integrated into the hierarchy.
- Agglomerative vs. divisive: Hierarchical methods can be agglomerative or divisive. The agglomerative methods, which are more common in praxis, produce up to N-1 connections to pairs, beginning from single documents each representing one cluster. The divisive methods begin with all documents placed in a single cluster and perform up to N-1 divisions of the clusters into smaller units to produce the hierarchy.
- Monothetic vs. polythetic: Monothetic means that only one feature is taken into consideration to determines the membership in a cluster. Polythetic means that more features are taken into consideration at once.
- Exclusive vs. overlapping: If the clustering method is exclusive, it means that each object can be assigned to only one cluster at a time. Overlapping strategy allows multiple assignments of any object.
- Fuzzy vs. hard clusters: Fuzzy methods are overlapping in their nature, assigning to an object a degree of membership between 0.0 and 1.0 to every cluster. Hard clusters allow a degree of membership to be either 0 or 1.
- Deterministic vs. stochastic: Deterministic methods produce the same clusters if applied to the same starting conditions, which is not the case with stochastic methods.
- Incremental vs. non-incremental: Incremental methods allow successive adding of objects to an already existing clustering of objects. Non-incremental methods require that all items be known in advance, before the actual processing takes place.

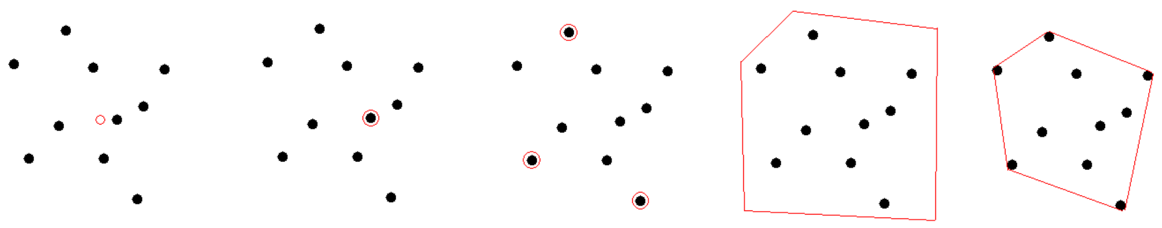


Figure 3.1: Different cluster representations: 1) centroid, 2) one element, 3) a group of bounding elements 4) a bounding polygon 5) convex hull.

- Order sensitive vs. order insensitive methods: Order sensitive methods produce clusters that depend on the order in which the objects are added. In order insensitive methods the order of items does not play a role, the method always produces the same clusters .
- Ordered vs. unordered clusters: An example of an ordered classification is a hierarchy, which can be helpful for searching.

3.1.3 Representation of clusters

The representation of a cluster and their relations depend on the clustering algorithm used. The representation used should be a cluster profile best representing the objects in the cluster. The most common way to represent a cluster is through its centriod, although some other representation methods are available, as can be seen from the following overview:

- Representation through cluster members by using one object or a set of selected members.
- Centroid is the most common way of representing a cluster. It is usually defined as the center of mass of all contained objects:

  $$c = \frac{\sum_{i=1}^{n} D_i}{n}$$

  Vectors for the objects contained in a cluster should be normalised. If this is not the case large vectors will have a much stronger impact on the position of the centroid than small vectors. If only relative profiles of the objects and not their sizes should be taken into consideration all vectors must be normalised.
- Geometric representation. Some boundary points, a bounding polygon containing all members of the cluster or a convex hull constructed from the cluster members can be used.
- Decision tree or predicate representation. A cluster can be represented by nodes in a decision tree, which is equivalent to using a series of conjunctive logical expressions like $x < 5 \wedge x > 2$.

3.2 Partitional Algorithms

Partitional methods try to reach an approximate solution by creating an initial partition and then assigning each document in to the best fitting partition, reaching a state of local minimum. The great advantage of this method over hierarchical methods is that running times of $O(NM)$ can be achieved. On the other hand there are a number of disadvantages, which differ from algorithm to algorithm, like the sensitivity to the choice of initial partition or the dependance on the order in which the documents are processed. However, due to good execution times, partitional methods are very often used when very large numbers of documents must be processed.

3.2.1 Single Pass Clustering

Single pass methods [HIL68] are probably the mathematically simplest clustering techniques, since the documents set is processed only once in a single pass. This makes single pass methods faster than the methods requiring multiple iterations. The number of clusters generated by this method is not fixed. It depends on the similarity threshold, s_{th}, which is used to decide if a document is similar enough to be assigned to a certain cluster. The following algorithm illustrates the principle of operation:

1. The first document D_1 is used to build the first cluster C_1.
2. Take the next document D_i and compare it to all existing clusters $C_1 \ldots C_M$ with the goal of finding the best fitting cluster C_j. The similarity to the found cluster is s_{ij}.
3. If s_{ij} is larger than the similarity threshold s_{th} the document is assigned to the found cluster, C_j, otherwise a new cluster C_{M+1} is created with the current document D_i as its first member.
4. The previous two steps are repeated until all N documents are assigned to a cluster.

Single pass methods have a disadvantage of creating large clusters at the beginning with the size of the clusters getting smaller as more clusters are created. The created partition depends strongly on the order in which the documents are processed. As the quality of the partition is relatively poor, it is a good idea to refine it with a few iterations of a reallocational algorithm like K-Means.

3.2.2 K-Means Clustering

K-Means [ROC66] is one of the most popular clustering algorithms which performs well on data having isotropic hyperspherical clusters. It is a reallocational method that requires an initial partitioning, with a predefined number of clusters, of the documents as a starting point. To improve the given partitioning the documents are moved from cluster to cluster with the goal of reducing the squared error, sometimes also called the distortion. It has been shown by [SEI84] that the partition converges towards a local minimum of the squared error function in finite number of iterations. The quality of the reached local minimum depends on the choice of the initial partition. The K-Means algorithm has a great advantage over hierarchical methods in having a time complexity of $O(MN)$, which, for a constant number of clusters M, can be regarded as linear. However, there are a number of important issues regarding the quality of results, like the sensitivity to the initial partition, the number of clusters that should be formed and the choice of continuous or non-continuous centroid updating, just to mention a few. The number of clusters, M, is chosen when forming the initial partition and can not be changed during processing. The general form of the algorithm is as follows:

1. An initial partition of the document set into M clusters $C_1 \ldots C_M$ is given.
2. The N documents $D_1 \ldots D_N$ are compared one by one to the M clusters and reassigned to the cluster C_j that is most similar to it.
3. Centroids of the M clusters are recalculated.
4. Previous two steps are iterated until in the allocation of the documents to clusters hardly changes, or until some other criterion is satisfied, for example a maximum number of iterations is reached or the decrease of the squared error goes below a specified value.

The squared error is the sum of squared distances between each cluster member and the cluster centroid. It is given by:

$$e^2(S, CS) = \sum_{j=1}^{M} \sum_{i=1}^{n_j} \|x_i^{(j)} - c_j\|^2$$

where j stands for the sum over M clusters and i stands for the sum over n_j-th cluster's members.

Number of Clusters and Initial Partition

As the K-Means algorithm works with an already created partition the number of clusters is predefined and the algorithm keeps it unchanged even if the number was not suitably chosen. An even bigger problem involves selecting M cluster seeds which are used to build the initial partition. The K-Means algorithm is extremely sensitive to the initial choice of cluster seeds, which makes this a crucial problem that has not been solved adequately yet. A technique that is very often applied is running the algorithm several times with different initial configurations and then choosing the one with the least squared error, which increases the running time significantly. A number of other strategies can be applied, like positioning the seeds uniformly in the information space or performing a hierarchical agglomerative clustering on a small sample of the document set as is done in Scatter/Gather (see Section 3.5) with the Buckshot and Fractionation algorithms. Another interesting method, that is based on the idea of clustering the cluster centroids of different partitions of the object set obtained with random seeds, is a refinement algorithm working on subsamples of the document set. To find the seeds for k initial clusters it operates as follows:

1. J small random samples of the data, S_i, $i = 1, \ldots, J$ are clustered with K-Means choosing an random initial partition. The sets CM_i, $i = 1, \ldots, J$ are cluster centroids of the found solutions, which form the set $CM = \cup_i CM_i$.
2. The set CM is clustered with KMeans using the sets CM_i as seeds producing cluster centroids FM_i.
3. The initial point set is chosen as the FM_i having a minimal squared error over the set CM.

Clustering of CM over the sets of initial points CM_i provides smoothing to avoid solutions corrupted by outliers. A it was reported in [BRF98], by using 10 samples of 10% the size of the original set, the solution shows significant improvements, specially for data with high dimensionality.

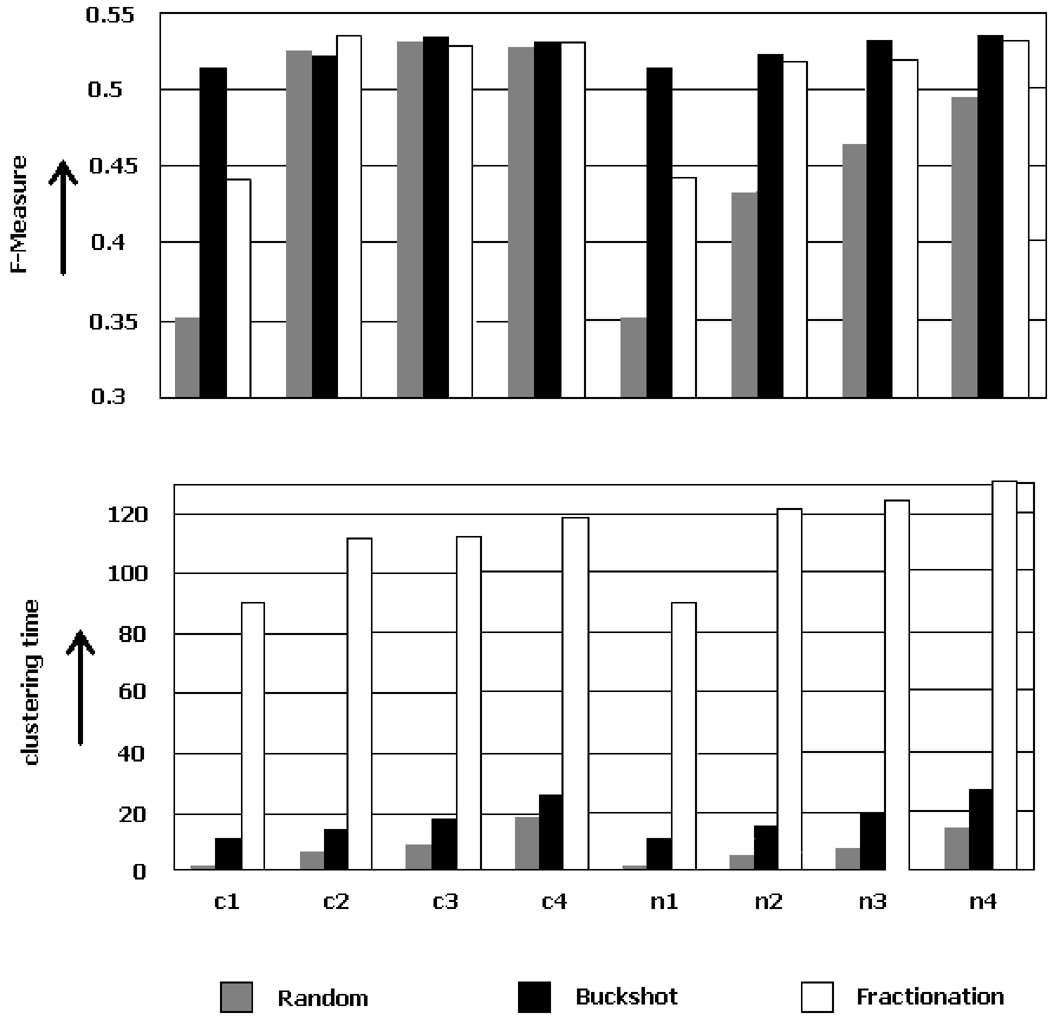


Figure 3.2: Effects of seed selection techniques and centroid updating policy on quality and speed of clustering. Taken from [LAA99].

Continuous Versus Non-continuous Centroid Updating

The other issue concerns the decision when to update the cluster centroids. The first possibility is to update the cluster centroids after each iteration is completed, the second is to update the cluster centroid continuously each time an element is inserted to or removed from the cluster. Although the first possibility seem to be a faster solution, evidence was presented by [LAA99] that this method requires more iterations than the second one and has a greater sensitivity to initial partitions that are not ideal. Especially when using random initial partitions, continuous updating brings significantly better results and needs less iterations.

In the Figure 3.2.2 the letter “c” stands for continuous and the latter “n” for non-continuous centroid updating. The number indicates the number of performed iterations. The F-Measure is a measure for quality of the clusters created by an algorithm compared to those created by humans. For this purpose each document as assigned a topic label T by a human judge. A F-Measure for each cluster C with respect to a topic T is defined as:

$F(C,T) = \frac{2PR}{P+R}$

where:

- $P = Precision(C,T) = N_1/N_2$,
- $R = Recall(C,T) = N_1/N_3$,
- $N_1 =$ number of documents judged to be of topic T in cluster X
- $N_2 =$ number of documents in cluster C
- $N_3 =$ number of documents judged to be of topic T in the whole set

$F(T)$ is the system's value of the F-Measure for the topic T and it is defined as the largest value $F(C,T)$ over all created clusters. The overall F-Measure is defined as:

$$F = \frac{\sum\limits_{T \in TS} (|T| \times F(T))}{\sum\limits_{T \in TS} |T|}$$

where TS is the set of Topics and $|T|$ is the number of documents judged to be of topic T.

Halt Criteria

The simplest halt criteria is to specify a maximum number of iterations, but sometimes it is not easy to guess the appropriate number of iterations, since it depends on the number and the configuration of the documents in the set. A better idea is to monitor the shift in cluster centroids or the squared error of the configuration, and when one of them falls below a specified value the algorithm stops. However, depending on the document set, their number and the implementation, it is possible that the shifting of documents will never stop completely. Therefore, the algorithm should actually stop when the improvement in either of these two values falls under a specified level.

3.2.3 ISODATA Clustering

The ISODATA Classification method [TOG74a] is a modification of the K-Means method, additionally incorporating a number of heuristic trial and error procedures for splitting, combining, and discarding clusters to reach an absolute minimum and therefore obtain an optimal set of clusters. At the beginning of every iteration it evaluates the clusters produced by the previous iterations. Large clusters can be split into two smaller ones depending on a combination of factors, like the maximum standard deviation of the cluster, or the number of objects in the cluster. When the distance between the cenroids of two clusters falls below a certain threshold, the clusters are merged together. If the number of objects in a cluster falls below a pre-defined number, the cluster is destroyed and the objects it contained are assigned to other existing clusters. Using these procedures the ISODATA clustering algorithm is theoretically able to find the optimal partitioning from every initial partition and it can be used on data sets with clusters that are not necessarily hyperspherical. However, the difficulty with this approach is that there is a number of additional parameters concerning splitting, merging and discarding clusters. These values must be carefully chosen to obtain the best results, and the optimal values may be different for different data sets.

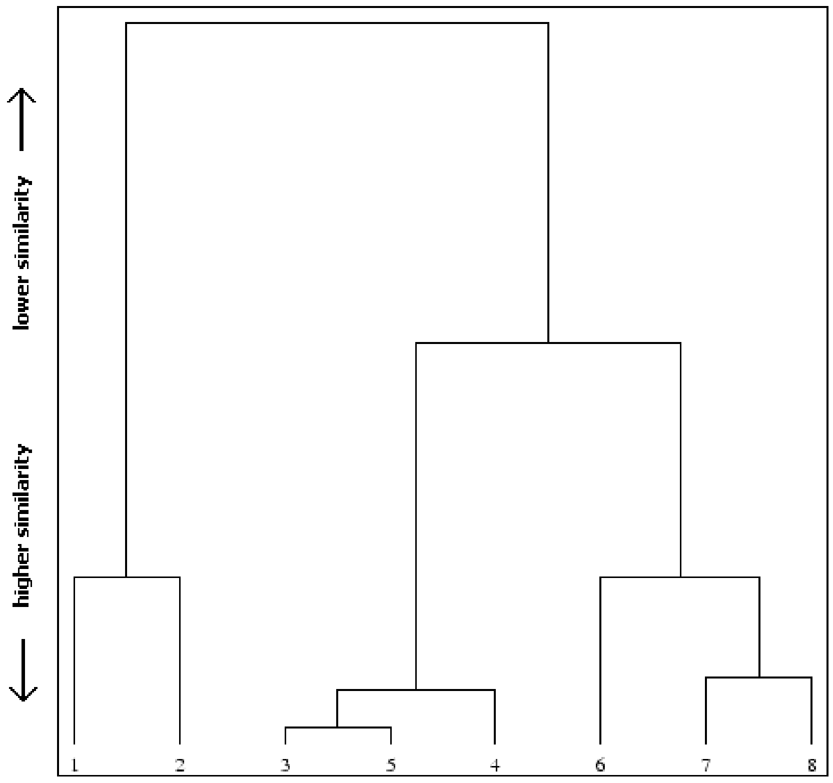


Figure 3.3: A dendrogram.

3.3 Hierarchical Methods

Hierarchical methods differ from partitional methods in the fact that they do not create a flat partition of N objects into M clusters, but build a hierarchy of clusters. This hierarchy can be visualised as a dendrogram (see Figure 3.3), which, by breaking it at different levels, gives different clusterings of the original data set. Data set objects are placed at the bottom of the dendrogram. The height of the connection depends on the similarity of connected objects, which are in this context both the data set objects as well as clusters. Connections in the lower part of the dendrogram connect objects which are more similar to each other, while those in the upper part connect the less similar ones.

Hierarchical methods can be subdivided into agglomerative and divisive methods. The first begin with the individual data objects and group them together according to a predefined similarity measure, until a desired number of clusters remains. The divisive methods work in the opposite direction, beginning with one cluster containing all data set objects and dividing it into more and more smaller clusters. Both methods have high computing times what makes their application on large numbers of objects impractical.

3.3.1 Hierarchical Agglomerative Clustering

Agglomerative clustering algorithms compute a sequence of partitions beginning from finest, where each cluster contains one data set object, to the coarsest where all objects are placed in a single cluster. The number of desired clusters can be chosen by cutting the resulting dendogram at a certain level. Agglomerative methods begin by creating N clusters, one for each data set object, which are then paired together step by step according to some similarity measure, until a certain number of clusters remains. Most of these algorithms use the same base pattern, differing mainly in the method of building the pairs and in the representation of the clusters.

Hierarchical agglomerative clustering (HAC) [VOO86] is a very popular greedy strategy giving very good results, but having a penalty of long execution times, as its time complexity is $O(N^2 \log N)$, or worse, depending on the implementation. It requires the generation of the similarity matrix which is a triangular matrix containing the pairwise similarity values for each two data set objects. The time and space complexity for this operation is $O(N^2)$. In its general form the algorithm is made up of the following steps:

1. Create the $(N-1) \times (N-1)$ similarity matrix.
2. Find a pair of two most similar objects in the matrix and merge them to a single cluster.
3. Update the proximity matrix to reflect the merge operation.
4. Repeat the previous two steps until the number of objects is reduced to a certain number, and than stop.

This method is quite sensitive to noise and outliers, in all of its forms. Another big problem is the halt criteria: at a certain point there can be a number of clusters left that are all dissimilar to each other. Merging the best pair at this point has no sense and it would decreases the quality of the resulting clustering significantly and increase the square error. A possible workaround would be to define a similarity threshold, s_{th}, that stops the algorithm if no pair can be found that has a similarity larger than s_{th}. However, a good value for s_{th} is different for every data set. Another approach for addressing this problem is to watch the squared error development. If the next merger would cause a strong increase of the squared error the algorithm should stop. The difficulty lies in deciding what constitutes a strong increase and what not, which is, again, different for each set of objects.

Different versions of the hierarchical agglomerative clustering algorithm exist. The main difference between them is how the similarity between clusters is calculated. Most popular methods are the single-link and the complete-link method. Other methods are for example the average-linkage method which is a combination of the previous two, the centroid based method where cluster centroids are used to calculate the similarity, the median method and the Ward method, each having different characteristics.

- Single-link method: This is method is also called nearest neighbor method or minimum distance method. The similarity of two clusters is defined as the similarity between the two most similar members each being a member of a different cluster:

  $s(C_i, C_j) = max\{s(D_k, D_l) : D_k \in C_i, D_l \in C_j\}$

  The single-link method tends to produce a small number of large clusters that are not very tightly bound and suffers from a chaining effect, sometimes merging clusters that are not very similar in their content. However, it can produce clusters that are non-isotropic, for example elongated or concentric clusters can also be identified.

- Complete-Link method: The complete-link method is also called farthest neighbor or maximum distance method. Similarity between two clusters is defined as the minimum off all pairwise similarities between members of the two clusters:

  $s(C_i, C_j) = min\{s(D_k, D_l) : D_k \in C_i, D_l \in C_j\}$

  This method produces a relatively large number of small, tight and compact clusters and it proved more useful than the single-link method in many applications, for example in document retrieval.

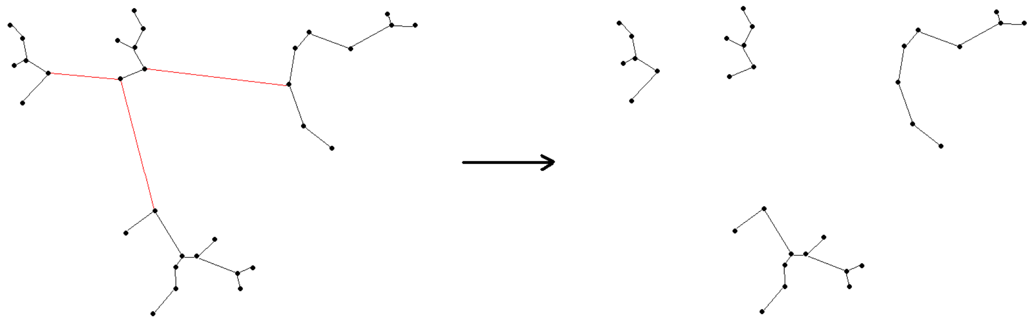


Figure 3.4: MST clustering: by removing the 3 longest links 4 separate clusters are created.

- Average-link method: The average-link method is a a compromise between the previous two. It tries to avoid the extremes of either too large or too compact clusters.
- Centroid method: Here the cluster centroids are used to calculate the similarity values between any two clusters. This method generally yields very good results. The problem in some applications of the method is that when merging a large and a small cluster, the small cluster has little influence on the centroid of the merged cluster, which can lie completely inside of the larger of the two paired clusters.

3.3.2 Divisive Clustering

Divisive clustering works top-down. Many different algorithms for divisive clustering were developed having completely different properties, but most of them suffer the problem of quadratic execution time, and are therefore not suitable for very large numbers of documents. The general form of the method can be described by the following steps:

1. All objects are assigned to one big cluster, which is than split with some division procedure.
2. Find the cluster C_i, which should be split into two smaller clusters according to some criteria for example the cluster with largest variance.
3. The last step is repeated until the desired number of clusters is reached.

The different algorithms differ mostly in the way the clusters are split. For example the Median-Cut [HEX82] and the similar Mean-Split [WIT85] algorithms, developed for finding representative colors for image quantisation, recursively subdivide the 3-dimensional color space into rectangular hyperboxes with the partition plane perpendicular to the coordinate axis with the largest color spread and passing through the median point, for the Median-Cut, or through the mean point, for Mean-Split, of the projected color distribution along this axis.

Minimum Spanning Tree Clustering

Minimum spanning tree clustering is a well known graph-theoretic divisive clustering method (see Figure 3.4). The minimum spanning tree, or MST, is the tree of minimum length connecting a set of objects in space. The length of the tree is defined as the sum of lengths of the links connecting the objects in a tree. The MST is similar to trees created by the single-link hierarchical clustering method. After the MST is constructed, which has an approximately quadratic time complexity, links are deleted from the MST, beginning with the longest link, and proceeding in order of decreasing length. Each time a link is removed from the MST a cluster is split into two smaller clusters, both being more compact then their parent. The MST is similar to trees created by the single-link hierarchical clustering method as connected sets obtained after each deletion are actually single-link clusters. This method does not perform well in data sets with many outliers, which is, together with its worse than linear running time, its biggest drawback.

3.4 Artificial Intelligence Techniques

Artificial intelligence techniques use approaches other than the statistical methods described so far. Fuzzy methods differ from the hierarchical and partitional algorithms in that they do not assign an object to a single cluster, but make use of partial memberships of every object to every cluster. Artificial Neural Networks (ANN) is a popular AI concept that has the ability to learn and adapt to new situations, similar to their biological counterparts.

3.4.1 Fuzzy Clustering

Hard clustering approaches generate partitions where each pattern belongs to one cluster. Fuzzy clustering [RUS69] uses the concepts of fuzzy logic and fuzzy sets, which are suitable for situations in which the boundaries between the considered mathematical sets are not sharply defined. This is, for example, the case in an image, where the boundaries between areas of different color are often unsharp. A fuzzy set can be defined by extending the standard concept of a set by adding a membership grade between 0 and 1 to every element of the set, where a higher membership value stands for higher confidence in the assignment of the object to the cluster. A function assigning a membership grade to every element of the fuzzy set is called the membership function μ. The design of this function is the most important problem in development of fuzzy clustering algorithms. In fuzzy clustering methods every cluster is a fuzzy set of all data set objects. The clustering is performed by associating each data set object D_i with every cluster C_j by assigning a degree of membership u_{ij}. Therefore each cluster, represented as a fuzzy set, can be defined as:

$$\begin{aligned} &C_j = \{\mu_{C_j}(D_1), \mu_{C_j}(D_2), \ldots, \mu_{C_j}(D_N)\} \qquad \text{with} \\ &\mu_{C_j}(D_i) \in [0,1] \quad \text{and} \quad \sum_{i=1}^{N} \mu_{C_j}(D_i) = 1 \end{aligned}$$

The fuzzy partitioning of the document set S composed of fuzzy clusters $C_1, C_2, \ldots, C_M$ can be represented through a $N \times M$ membership matrix U, where the elements of the matrix are given by $u_{ij} = \mu_{C_j}(D_i)$, with $u_{ij} \in [0,1]$. With these definitions a general form of the Fuzzy C-Means partitional algorithm can be given:

1. Analyse a sample of the input data set to find M prototype clusters, the seeds. The values u_{ij} of the membership matrix U are calculated by determining the membership degree of every data set object in each cluster.
2. Calculate the value of the fuzzy squared error for the current partition using the matrix U. The objects are reassigned to clusters in such a way to minimise the squared error function. The fuzzy centroids and the membership matrix U are recomputed.

3. Check for convergence. The algorithm converges when the membership functions of the $(p+1)$-th iterations are the same or did not change significantly as those of the p-th step. If this is true stop, otherwise repeat the previous step.

The fuzzy squared error function is the sum of the squared distances of all data set elements D_i represented by vectors $\vec{x_i}$ from the centroids $\vec{c_j}$ of each cluster C_j weighted by the membership grade u_{ij}:

$$e^2(S, U) = \sum_{i=1}^{N} \sum_{j=1}^{M} u_{ij} \|\vec{x_i} - \vec{c_j}\|^2$$

Fuzzy C-Means is much better than the hard K-means algorithm at avoiding local minima and it can converge to a global minimum of the fuzzy squared error function. However, for some data set configurations it will converge only towards a local minimum.

3.4.2 Neural Networks

Artificial Neural Networks (ANN) have, as their biological counterparts, the ability to learn their interconnection weights adaptively. Objects represented by vectors are presented at the input and are associated with the output nodes. The weights between the input nodes and the output nodes are iteratively changed in a process called learning until some termination criteria is satisfied. ANNs process numerical vectors, grouping similar vectors automatically based on data correlations and representing them by a single neuron. The process of training the neural network is computationally very intensive, so ANNs can not be used to produce results on-the-fly and are not suitable for interactive applications. Examples of ANNs used for clustering purposes are the Self Organising Map networks [KOH88] and Adaptive Resonance Theory networks [CAG90].

Self Organising Map

Self Organising Map (SOM) is an example of an Artificial Neuronal Net based on unsupervised learning and therefore well suited for clustering tasks. It reduces the dimensionality of the input data and produces a 2-D map out of a multidimensional data set by converting the input feature space to a topological map with bounded regions. Vectors that are close in the high-dimensional space will also be close in the 2-D projection. SOM differs from neural networks like the feedforward or multi-layer perceptron, that are more widely used, through its network architecture and approach to learning. The network is made of only two layers, one for input and one for output. Every unit in the input layer is connected to all units in the output layer. The output units are mostly ordered as a 2-D array to form a topological map.

SOM (see Figure 3.5) is similar to the Linear Vector Quantisation (LVQ) classification method that adjusts the boundaries between pre-defined categories in such a way to minimise the misclassifications. In the training phase the best matching output node is found for each input training vector. If the known class of the input vector differs from the class of the best matching node then the next best matching node is found and, if it has the correct class, the weights of the best matching node are moved a little farther from the training vector and the weights of the next best matching node are moved a little closer to it. The boundary between classes are moved closer to the optimum position. The SOM differs from LVQ in the fact that output nodes do not correspond to some known classes but to unknown clusters that are found during the processing. During the training phase the SOM finds the output node with the smallest distance from the current training vector and changes the weights

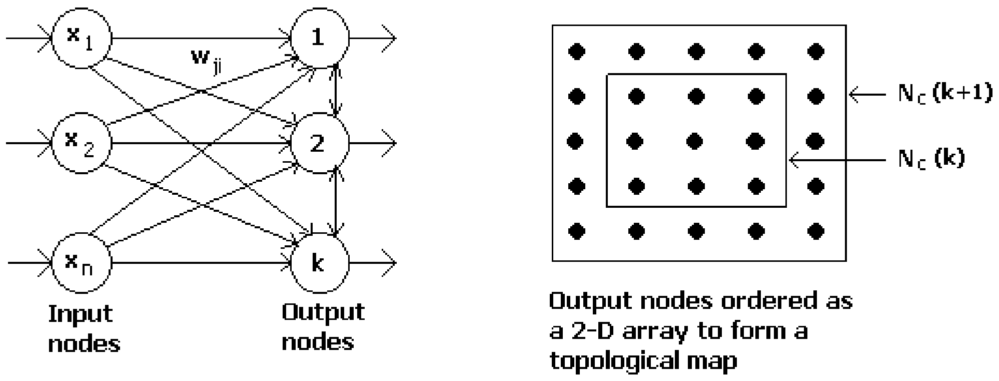


Figure 3.5: SOM network architecture

w_{ji} to increase the similarity to the current input. The weights of neighbours of the matching node are also updated. By applying different training vectors, different winning output nodes and their neighbours are influenced in such a way that the weights associated with the output nodes are changed to map the distribution of the input vectors. When the training is completed the weights of every output node model the features of a cluster in the input data set.

A general form of the Kohonen's training algorithm is as follows:

1. Initialise the weights vector w_j to small random values.
2. Present a new input vector x.
3. Find the output node c whose weights are closest to the input vector. This is the winning node. Euclidean distance is used as proximity measure: $d = \|x - w_j\|$.
4. Modify the neighbourhood size $N_c(k)$ and the learning rate $\alpha(k)$, where k is the iteration, reducing them linearly and slowly during the training phase.
5. Go to step 2 until learning is complete. The process can take as many as 10000 to 100000 iterations to complete.

The process is sensitive to the initial configuration and can produce a suboptimal partition for bad initial weights. The convergence depends on several parameters like the neighbourhood size $N_c(k)$ and the learning rate $\alpha(k)$. For example if the initial value of the neighbourhood size $N_c(0)$ is too small the convergence can fail to occur. After the training phase is finished the network can be used for recognition of patterns. The weights are saved and the output nodes are labelled according to the class they represent. A vector is presented at the input which fires the best match output node, the winning node. The winning node is lit up or the node's label is displayed to show to which cluster the presented vector belongs to.

3.5 Scatter/Gather

Scatter/Gather [CKP92] is a clustering-based document browsing system, developed at Xerox PARC. It introduces two fast, linear time clustering algorithms and uses clustering as an information access tool instead of using it for information retrieval purposes, as it has proved unsuitable and too slow in this area. Following the cluster hypothesis stating that relevant documents are more similar to each other than to non-relevant documents, an interactive document browsing system based on clustering was developed. The system builds clusters containing similar documents, which are represented by textual summaries. These summaries are made up of titles as well as keywords that characterize the cluster. Based on the summaries the user can select several clusters or documents for further study, forming a subcollection, which is then reclustered to refine the results and give a more detailed view in the structure of the chosen selection. The operations of scattering a group of documents and then gathering some of the formed clusters that appear interesting for another scatter/gather round, and repeating the steps, make the groups become smaller and more detailed. Scatter/Gather can be used on results returned by conventional search engines or on any given collection of documents.

3.5.1 Browsing as an Information Access Method

Typically, information retrieval systems are driven by a query. Sometimes it is very hard to formulate a good and precise query and if the user wants to get an overview on the document set, such systems are not very useful. Scatter-Gather uses a metaphor of a conventional textbook: if the user is interested in getting an overview, the table of contents, which shows the logical structure of the text, is the best solution. The table of contents gives clues about of which questions can be answered by the text. If the user has a specific question with terms defining the question, the solution is to consult the index. Scatter/Gather presents cluster-based, dynamic table of contents to the user for navigating a collection of documents, and also offers a query based text search method.

The Scatter/Gather orders documents into topically coherent groups and characterises them by presenting descriptive textual summaries consisting of topical terms and titles that sample the contents of the cluster. The clusters can be used to identify interesting documents or sets of documents, which can be reclustered into more refined groups and to eliminate irrelevant documents. The user can select clusters or documents and form a subcollection that is reexamined by reclustering it on-the-fly. By iteratively repeating the examination different topics can be seen depending on the subcollection chosen to be clustered.

For this purpose Scatter/Gather needs a clustering algorithm which can cluster a large number of documents in a time which is tolerable for user interaction and which offers good clustering quality. Hierarchical methods offer good results but have quadratic execution time and are therefore too slow. Partitional algorithms approach linear running time but the quality of the results is often inadequate. Two linear partitioning algorithms using techniques from the hierarchical algorithms were developed for this purpose.

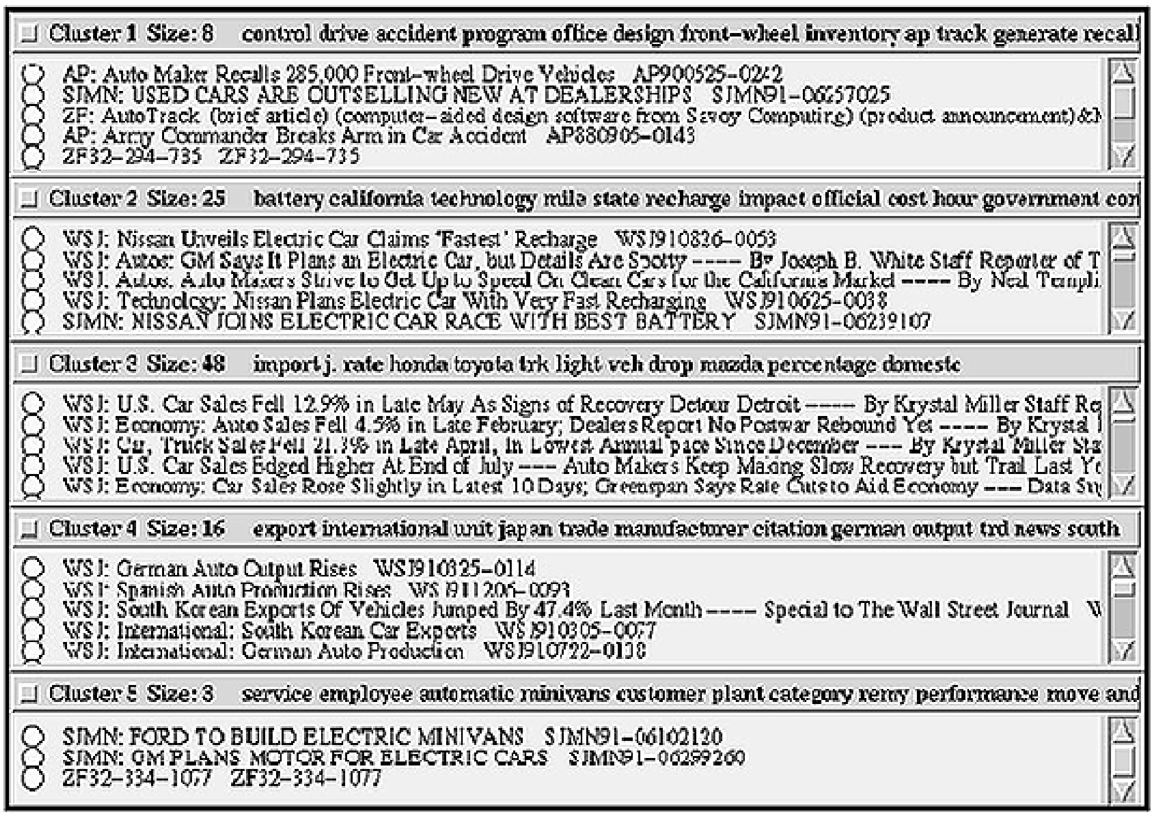
3.5.2 Clustering Algorithms: Buckshot and Fractionation

Partitional seed-based clustering algorithms like the popular K-Means can be divided in two phases:

1. Identifying the M seeds for the initial partition that is constructed by assigning each document in the data set to the best fitting cluster.

2. Refining the initial partition iteratively.

The k-means algorithm is known to be very sensitive to the choice of initial partition, producing poor results for bad choices. The Buckshot and Fractionation algorithms are procedures for finding the centres for an initial partition. The initial partition is created by assigning each document to the nearest centre creating M clusters. These are then refined by the K-Means algorithm or a modification of it. To find the seeds both algorithms use a clustering technique that can produce



```
Cluster 1 Size: 8    control drive accident program office design front–wheel inventory ap track generate recall
  AP: Auto Maker Recalls 285,000 Front–wheel Drive Vehicles   AP900525–0242
  SJMN: USED CARS ARE OUTSELLING NEW AT DEALERSHIPS   SJMN91–06257025
  ZF: AutoTrack (brief article) (computer–aided design software from Savoy Computing) (product announcement)&
  AP: Army Commander Breaks Arm in Car Accident   AP880905–0143
  ZF32–294–735   ZF32–294–735
Cluster 2 Size: 25    battery california technology mile state recharge impact official cost hour government cor
  WSJ: Nissan Unveils Electric Car Claims 'Fastest' Recharge   WSJ910826–0053
  WSJ: Autos: GM Says It Plans an Electric Car, but Details Are Spotty ---- By Joseph B. White Staff Reporter of T
  WSJ. Autos. Auto Makers Strive to Get Up to Speed On Clean Cars for the California Market ---- By Neal Templi
  WSJ: Technology: Nissan Plans Electric Car With Very Fast Recharging   WSJ910625–0038
  SJMN: NISSAN JOINS ELECTRIC CAR RACE WITH BEST BATTERY   SJMN91–06239107
Cluster 3 Size: 48    import j. rate honda toyota trk light veh drop mazda percentage domestc
  WSJ: U.S. Car Sales Fell 12.9% in Late May As Signs of Recovery Detour Detroit ---- By Krystal Miller Staff Re
  WSJ: Economy: Auto Sales Fell 4.5% in Late February; Dealers Report No Postwar Rebound Yet ---- By Krystal
  WSJ: Car, Truck Sales Fell 21.3% in Late April, In Lowest Annual pace Since December --- By Krystal Miller Sta
  WSJ: U.S. Car Sales Edged Higher At End of July --- Auto Makers Keep Making Slow Recovery but Trail Last Yo
  WSJ: Economy: Car Sales Rose Slightly in Latest 10 Days; Greenspan Says Rate Cuts to Aid Economy --- Data Su
Cluster 4 Size: 16    export international unit japan trade manufacturer citation german output trd news south
  WSJ: German Auto Output Rises   WSJ910325–0114
  WSJ: Spanish Auto Production Rises   WSJ911206–0093
  WSJ: South Korean Exports Of Vehicles Jumped By 47.4% Last Month ---- Special to The Wall Street Journal  W
  WSJ: International: South Korean Car Exports   WSJ910305–0077
  WSJ: International: German Auto Production   WSJ910722–0138
Cluster 5 Size: 3    service employee automatic minivans customer plant category remy performance move and
  SJMN: FORD TO BUILD ELECTRIC MINIVANS   SJMN91–06102120
  SJMN: GM PLANS MOTOR FOR ELECTRIC CARS   SJMN91–06299260
  ZF32–334–1077   ZF32–334–1077
```

Figure 3.6: Scatter/Gather interface, taken from [HEP96].

high quality clusters but has a bad running time of $O(N^2)$. This clustering technique, also called the cluster subroutine, is an implementation of the group average agglomerative clustering. However, to achieve linear running times the clustering subroutine is only applied locally, on a small subset of the document set.

Buckshot and Fractionation are both fast, linear time clustering algorithms, applying principles from hierarchical agglomerative clustering methods. Each has a different application area in the Scatter/Gather system. Fractionation, although having a linear running time, is significantly slower than Buckshot, but produces better, more accurate results. Therefore, fractionation it suitable for the non-interactive partitioning of large document corpora. A clustering of the whole document corpus can be produced and presented to the user at the beginning of the interactive Scatter/Gather session. Buckshot, being faster, is used for interactive reclustering of the selected document subsets during the interactive Scatter/Gather session. Although both algorithms are linear and produce accurate results, they are, and this specially applies to Fractionation, sensitive to data sets with many outliers which is consequence of using hierarchical agglomerative clustering (HAC) concepts and techniques.

The refinement procedure is essentially a K-Means algorithm with additional Split and Join procedures (also see ISODATA algorithm in Section 3.2.3). Split divides a cluster with poor coherency into two smaller clusters using the Buckshot algorithm without refinement steps. The criterion is the cluster self-similarity which is the average similarity between documents themselves and between the documents and the cluster's centroid. The Join procedure merges clusters based on the degree of overlap of their topical words.

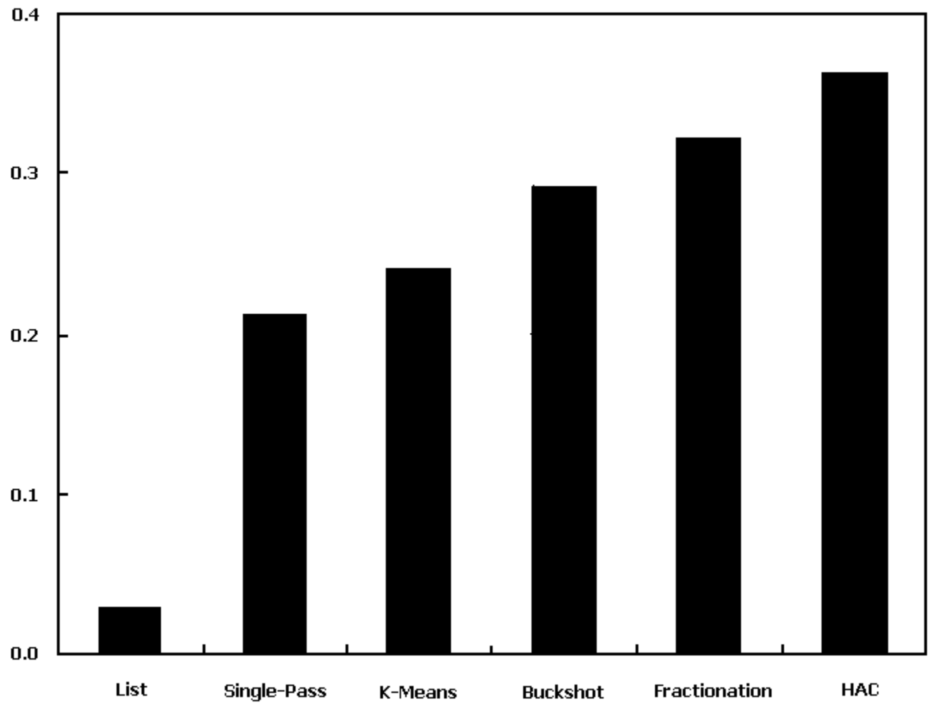


Figure 3.7: Precision of clustering algorithms compared to the ranked list, based on queries to the MetaCrawler meta-search engine. Taken from [ZAE98].

Buckshot Algorithm

Buckshot [CKP92] finds the initial cluster centroids, the seeds, by selecting a random sample of size $\sqrt{MN}$ from the document collection of size N, and clusters them with the cluster subroutine to produce M cluster centroids. These centroids are the initial seed for the K-Means algorithm. By keeping M small and constant compared to N, and by using the HAC algorithm on a subset of documents of size $\sqrt{MN}$, Buckshot reaches processing time that is linear with the number of documents. However, the sampling of the document set is risky if the clusters are small because they may not be included in the sample, but as Scatter/Gather mostly generates a relatively small number of larger clusters this is not so much a problem. As a consequence of using random samples the algorithm is not deterministic.

Fractionation Algorithm

Fractionation [CKP92] is an approximated HAC algorithm which operates on the whole document set using successive application of the cluster subroutine over fixed sized samples of the document set to find the seeds. In each iteration the search for the closest pair of clusters is performed on a subset, in a locally bound region, of the document set. By performing this search on a fixed size sample instead of the document set, linear time complexity can be reached. In the first step the algorithm divides the document set S into N/k subsets of fixed size, where $k > M$. Each subset is clustered separately with the cluster subroutine, forming a number of clusters in each group so that reduction in number, from documents to clusters, in each subset is approximately a factor of p. The procedure is iteratively repeated with all formed clusters instead of documents, until only M clusters are left. This algorithm has similar advantages and disadvantages to the hierarchical agglomerative methods: it produces high

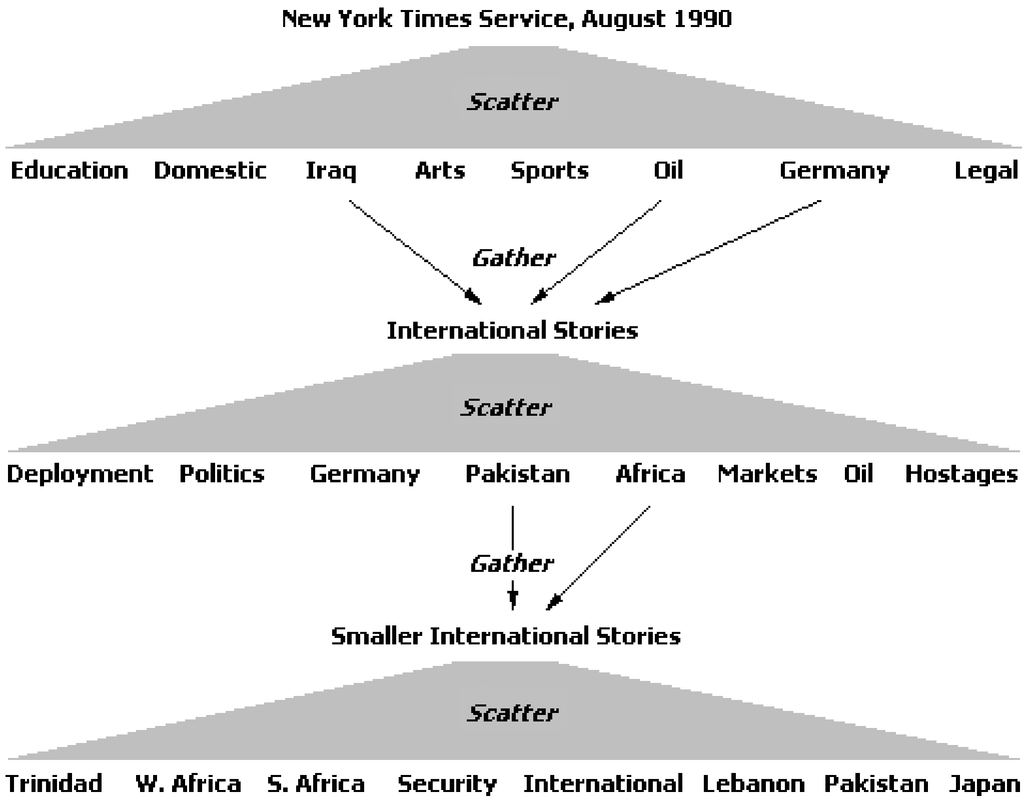


Figure 3.8: Scatter/Gather browsing, taken from [CKP92].

quality results on data sets with not too many outliers, and has the problem with the halting criteria.

3.5.3 Browsing of Very Large Document Collections

The linear time algorithms require less than a minute to process a few thousands of documents. This is fast enough for interactive browsing, but the system can not deal with very large document collections containing hundreds of thousands or millions of documents. Scatter/Gather tries to address this problem by offering constant interaction time document clustering, however combined with a longer, near-linear time, preprocessing [CKP93]. The hypothesis is that documents that are similar enough to be clustered together in a fine-grained clustering will also be clustered together in a coarse-grained clustering. This is called the Cluster Refinement Hypothesis.

A cluster hierarchy is a data structure that follows from the Cluster Refinement Hypothesis. It is a tree whose internal nodes are subtrees of hierarchies and whose leaves are single documents, as produced by HAC algorithms. Similar documents have a common ancestor low in the tree, while less similar documents have a common ancestor near the root. An internal node can be viewed as a meta-document representing all documents it contains. The hierarchy allows us to view the collection at different granularities, either as a smaller set of meta-documents closer to the root, or more finely as a larger set meta-documents closer to the leaves. The same can be applied to each meta-document because it can be viewed as a root of its own tree. The hierarchy is generated by recursively applying the Fractionation algorithm on the complete document set, in the first step producing M clusters in $O(MN)$ time , and than recursively on each generated cluster each time producing M subclusters, stopping when the number of documents in a cluster is small enough. If the tree is balanced so that each cluster contains approximately the same number of subclusters, the running time is $O(NM \log N)$ because there are $O(\log N)$ levels in the tree. The typical size of the generated hierarchy is approximately 10% of the size of the original document set.

After a hierarchy is constructed in a pre-processing step it can be presented to the user for browsing, this time in form of clusters containing constant numbers of meta-documents. This allows the gathering and reclustering of collections of meta-documents in constant time, typically less then a minute. In this case the faster Buckshot algorithm is used. The meta-documents contain profiles of constant size constructed from the most topical entries in the profiles of the contained meta-documents, thus not increasing the computation time at the higher levels of hierarchy. After a few Scatter/Gather steps, when the groups become small enough, individual documents can be located.

Chapter 4

Search Result Visualisation

4.1 Introduction and General Principles

A huge amount of information is available electronically over the Internet stored in textual, multimedia or some other file format. Searching for a specific topic results in a large number of search results in a completely unstructured form, mostly presented to the user as a list of documents sorted by estimated relevance to the search query and containing attributes like title, URL, and author. Although Information Retrieval (IR) systems often return large amounts of information and almost every relevant information can be found, it is hard for the user to tell which of the retrieved information is of interest and which is not. To allow a distinction of relevant and non-relevant information, to provide an overview and to avoid the problem of getting lost in Hyperspace different visualisations tools have been developed to help the user retrieve the desired information effectively, organise and display it in a meaningful way, and navigate and access it interactively. Additional information can be presented to the user by mapping metadata attributes of the search results, such as estimated relevance, document size, modification date or number of links to graphical attributes of the visual representation, such as the size, colour, shape, height. Relationships between documents can be displayed by proximity, containment, connecting lines or some other means of shaping the information space, giving the user a sense of overall composition and layout. Explorable visualisations provide very powerful and intuitive means to the user to deal with very large numbers of documents, to identify groups of documents similar in content, eliminate outliers, compare documents with regard to different attributes, and recognise trends and patterns in the data. However not all methods are suitable for all types of data. For example, for multi-dimensional data, an effective way must be found to map the relations in multi-dimensional space to 2-D or 3-D space that can be visualised. There is no general solution that would fit every kind of data. As a result a large number of different approaches were developed, and a small number of those that are suitable for search results visualisation will be presented in this chapter.

4.1.1 Metaphors and Coding

Visualisation systems should be able to present large amounts of information in an understandable and intuitive manner giving the possibility for interactive navigation through the information space and providing fast access to any information. Commonly documents are represented as dots or shapes in 2-D or 3-D space. 2-D representation slightly extended into a third dimension (also called 2.1-D) has proved powerful enough without posing excessively high cognitive loads on the user. Metaphors are used to help the user build up a mental model of the information space. These metaphors can be applied to encode information in the representation of each document, but can also represent relations between them, and are even used to shape the information space itself, for example with a topographical map. Both numerical and non-numerical information can be encoded, linearly or in some other way. When designing such a visualisation system, several decisions have to be made concerning the mapping of data set attributes to the attributes of the objects in the user interface. The overview of visualisation as well as the use of space and the details filling it must also be taken into consideration. Some often used coding principles are:

- Size Coding: Size is commonly use to indicate quantitative information like the size of a document, number of elements contained in an object, or the relevance of a document.
- Colour Coding: Colour can also be used to show quantitative information, but it is not as suitable as the size. Colour is better for coding the type of an object.
- Brightness Coding: This is very suitable for coding numeric information and is similar to size coding.
- Shape Coding: Shapes are mostly used to display non-numerical information such as the type of a document Shapes are metaphors used to code some information and are often derived from real world objects. Ssuch metaphors help the user build up a mental model of the information space. Simple geometrical forms like rectangles, triangles, circles, spheres, cubes, or more real-world-like objects are used to code documents, while structures like trees and landscapes are used to shape the information space and visualise relationships.
- Proximity Coding: The distance between any two documents in the information space can be used to code the similarity of the documents: the closer they lie the greater the similarity. This kind of coding produces groups of similar documents which are easy to identify visually. If the groups can be marked or labeled in some way, the user can locate the documents containing the desired information very easily.
- Position Coding: A position of an object relative to an axis can be proportional to a numerical value, but even non-numerical information can be coded this way as certain attributes can be placed on suitable positions on the axis.

4.1.2 User Interface Operations

A visualisation should offer the possibility to intuitively manipulate the displayed objects and to navigate the information space. Feedback is also an important component in interactive systems. Some of the operations and methods offered by such systems are:

- Pointing to (mousing over): By pointing, an object or a group of objects extra information can be displayed, such as the title, size, keywords or even a summary. This helps in getting a more detailed overview as well a locating points of interest.
- Selection: This operation is mostly used to choose objects that some other operation should be applied on. A selection of objects is obtained through operations like pointing and dragging, but can also be the result of a search operation.
- Dragging: Moving a group of objects or selecting them by marking a part of the display can be achieved by dragging.
- Searching: Although we are here talking about search results visualisation, it can be useful to perform more detailed searches on the search result set itself. Such a search can also include attributes and allows very fine extraction of items with certain characteristics.

- Zooming: Zooming is the technique of selecting a smaller region of the currently displayed, larger portion of space, for more detailed display. Zoom out is the opposite operation, changing the view from a smaller space area back to the previously displayed larger one.
- Panning: In a case when a viewed space is too large to be displayed on-screen at once, smooth motion from one part of the space to another can be achieved by sliding the scroll bars.
- Filtering: The operation of removing, temporarily or permanently, uninteresting elements from the display, according to a specified criteria, can give a better overview, especially when the number of displayed items is very large.
- Exchanging axis dimensions: Many systems visualising multidimensional data, for example document matadata, position objects in space according to the value of some attribute. By exchanging the dimension of an axis the objects are repositioned.
- Axis range setting: By defining the range of the attribute currently assigned to an axis, the user can filter out unwanted objects and get a more detailed view of those within the selected range.
- Details display: Visualisation systems should be able to give a closer look at a displayed object or an object group. A more detailed view or additional information on the object, a summary, or a complete view of the object should be given by performing simple mouse operations like pointing or clicking.
- Relating: The possibility to relate displayed objects helps to discover relationships between them.
- Maintaining history: Explorative visualisation is a sequential process, so that maintaining histories enables the user to retrace previous actions, and undo, modify or repeat them.
- Query Construction Support: Constructing a query is crucial to getting good search results. It should be a creative process where the system supports the user, for example by proposing a refinement of previous queries with relevance feedback (from previously returned objects), analysing the user's earlier behavior, etc.

4.1.3 Different Approaches to Search Results Visualisation

Documents which were returned as a result of a search query can be visualised according to different criteria and with different objectives. Depending on the organisation of the documents, on the environment that they were found in, on the purpose of the visualisation, and on analysis and processing performed on the retrieved documents, the following classification of search results visualisation systems can be given (combined systems are, of course, also possible):

- Systems showing compact representation of the content of the retrieved documents give clues to how the documents are composed, in which parts of the document the search terms can be found and what their frequencies are, as well as how each document relates to every term of the search query as a whole. As these visualisations are mostly simple and compact, they can be used inside more complex systems.
- In some cases it can be very useful to concentrate on document metadata instead of on their actual content. Analysing and comparing size, creation and last modification date, authors and similar attributes, helps finding wanted information and can lead to the discovery of interesting patterns in document distribution.

- Composing a good search query is key to getting good search results. As a result systems visually supporting the user in building the search query based on feedback or history can improve greatly the quality of the retrieved documents.

- As today almost all documents can be accessed through the World Wide Web, displaying and analysing the structure of links pointing to and from the retrieved documents can give a picture of the importance of found documents and lead to the discovery of more relevant documents that were not returned by the search query.

- Systems visualising document collections according to the similarity of their content try to group documents with similar contents together and possibly mark or label them, so that the user can quickly decide where to locate relevant information. The number of techniques used in this area is large and many completely different approaches are used and combined to achieve the goal.

4.1.4 Placement Models

Systems which deal with the content of the documents returned as a result of a search query, generally use a representation of the documents as n-dimensional vectors, called Vector Space Model (see Section 2.2). As high-dimensional spaces can not be directly visually presented to the user, a procedure is needed to map the documents from the high-dimensional space, to 2-D or 3-D space preserving as far as possible the relations in the multidimensional space. Many approaches are known each having different characteristics, but they all share the same proximity principle: similar documents close to each other in the high-dimensional space must lie close to each other in the display. Here is an overview of a few popular techniques for dimensionality reduction:

Multidimensional Scaling

Multidimensional scaling (MDS) takes a distance or similarity matrix of objects existing in a space with a number of dimensions equal to n, and iteratively places objects in a space with number of dimension equal to m so that the original distance matrix is represented as well as possible. MDS is not an algorithm or an exactly defined procedure, but it is a name for multivariate analysis techniques which attempt to rearrange the objects from the space with any number of dimensions to a space with some other number of dimensions approximately preserving the original distances. It moves objects in the space with the requested number of dimensions and uses a function minimization algorithm to evaluate how well the distances between objects can be reproduced by different configurations maximizing the goodness-of-fit. Stress is often used as a measures of goodness-of-fit, which means it is a measure for the degree to which the distances in the m-dimensional space differ from the original distances in the n-dimensional space. High stress values mean that a particular configuration poorly reproduces the original distances, while lower stress values imply a better fit. Several different stress definitions can be used, but most of them are based on the computation of the sum of squared distance differences:

$stress = \sum[d_{ij} - f(\delta_{ij})]^2$, where d_{ij} stands for the produced distances, δ_{ij} stands for the original distances, and $f()$ is a monotone transformation of the original distances.

Force-Directed Placement

Force-directed placement is a very popular iterative technique based on a spring model. A system of N masses, connected to each other with springs, is described using mechanical physics. At the beginning the masses are positioned at some initial positions exhibiting attractive and repulsive forces F_{ij} on each other. These forces depend on high-dimensional distances d_{ij} between the objects or on their similarities, and on the geometric distances g_{ij} of the objects in the layout space. Afterwards the masses are iteratively moved depending on the forces until the resulting forces are equal to zero and the system is in a minimal energy state. Additionally speeds as well as damping proportional to the speed can be taken into consideration to add smoothness, stability and realism to the simulation. The result is a separation in the layout, where high-dimensional distances are mapped to 2-D or 3-D distances in the layout in such a way that the high-dimensional relations of the objects are preserved. However, this approach has two big drawbacks. The first is that for each of the N documents in the search result set a force resulting from all of the other $N - 1$ documents has to be calculated, which results in quadratic running time per iteration. The second side effect is that the algorithm has a tendency to getting stuck in some local minimum before reaching the state of global minimum energy.

The magnetic spring model is an extension to the standard force-directed placement method trying to minimise the problem of getting stuck in a local minimum. A magnetic spring is a object having mechanical spring characteristics and additional magnetic features. Parallel, polar and concentric magnetic fields are introduced as well as compositions of those. This results in the standard attractive and repulsive forces caused by springs and rotative forces caused by the magnetic fields.

There are a number of techniques which address the problem of high time complexity. Clustering can be applied, where the documents are compared only to cluster centers and to documents from their own cluster. The problem is that the quality of the clustering algorithm determines the quality of the visualisation, and iterative linear clustering algorithms are in general less precise than those with quadratic time (see Chapter 3). Also much detail information is lost as most of the document to document comparisons are left out. A hierarchical space subdivision is another approach where documents are compared to other documents directly only if they are in the same subarea. Otherwise they are compared to a subarea as a whole. However, as documents move between subareas, significant overhead is required to update and maintain the subdivision hierarchy, making this solution rather complex.

A technique based on stochastic sampling of the document set, described in [CHA96a], preserves all advantages of the force based model, while having a linear execution time per iteration. The algorithm is relatively easy to implement and linear execution times are achieved by taking into consideration only a constant size sample of the complete set when calculating the force on each object in the set. Each of the N objects in the information space maintains two sets of constant size:

- Random set S_i: A new random set is constructed for each iteration, by selecting a fixed number of random elements from the remaining $N - 1$ elements, excluding elements that are currently in the object's neighbour set.
- Neighbour set V_i: The neighbour set dynamically maintains references to neighbour objects in the high-dimensional space. Before the first iteration it is filled with random elements. Real neighbours are found iteratively during each construction of the random set. Each time a new random element is selected for insertion into the random set it is compared to the furthest element of the neighbour set. If the selected random element lies closer than the furthest neighbour element in the high-dimensional space, the random element is inserted in the neighbour set and the worst neighbour is removed from the set. Otherwise, if the selected random element does

not lie closer than the worst neighbour, it is inserted normally into the random set. In this way neighbour elements get collected in the neighbour set through the iterations.

The force between each two objects i and j in the set, denoted by F_{ij}, is proportional to the magnitude $d_{ij} - g_{ij}$, where g_{ij} is the geometric distance and d_{ij} is the high-dimensional distance between the two objects. The high-dimensional distance is defined as $d_{ij} = (1 - s_{ij}^p)$, where s_{ij} is the scalar product of the vectors of the two objects, and p is an exponent to increase the discrimination between them. The force on every object, denoted by F_i is calculated by adding force components produced by the elements stored in both sets of the object.

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

When forces acting on every particle are calculated, the position and velocity are updated for each of them by a fourth order Runge-Kutta process. To provide stability and smoothness, viscous damping proportional to the velocity is added. The halting criteria is reached once a maximum number of iterations have been made or the stress value has little or no decrease over a certain number of iterations. This stress value is based on mechanical stress in spring systems. Calculating it for every iteration would increase the overall running time from linear to quadratic, therefore it is calculated only every $\sqrt{N}$ iterations.

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

Simulated Annealing

Simulated annealing is a technique derived from a statistical mechanics physical model, describing the cooling down of liquids form an unordered to an ordered, solid, state by iteratively lowering the temperature T and recalculating the state in every step. The difference from other iterative placement methods is that it introduces steps that spoil the momentary solution making this approach less prone to getting stuck in a local minimum. However this is only true if the cooling occurs very slowly. Depending on the speed of lowering the temperature the system reaches different final values of energy. If the cooling is fast only a local minimum is found. On the other hand if the temperature sinks slowly, a state representing the minimum energy is reached. Best results are achieved when the temperature is reduced in very small steps, which unfortunately results in long execution times, making this technique suitable only for relatively small numbers of objects. The behavior of simulated annealing systems is described through the Boltzmann probability distribution:

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

where E is the energy, T is the temperature, k the Boltzmann constant, and $p(E)$ the probability distribution of energy values. Deriving the Boltzmann distribution and setting the value of k to 1 for simulation purposes, the probability with which the system moves from the position with the energy E1 to a position with the energy E2 is obtained:

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

Therefore, the system can, although with a small probability, move to a state with a higher energy. This feature is useful for avoiding a state of local minimum as the systems can shake free of such non-optimal solutions.

4.2 Visualisation of the Content of the Retrieved Documents

The following systems offer simple visualisations designed to display the how each returned document relates to each term in the search query. They also have additional capabilities such as supporting the user in constructing the query or approximately displaying where in the document the desired term can be found enabling faster access to information.

4.2.1 Tile Bars

TileBars [HEA95] give users a compact representation of the content of the retrieved documents with respect to the keywords in the query (see Figure 4.1). They simultaneously display the length of the document, the frequency of keywords, and their distribution in the document. Between the title of each document and the document number there is an icon which represents the document. Each row of boxes in the icon represents one keyword or a group of keywords and each box represents a thematic section of the text. Thematic sections are found by an algorithm which analyses the document and partitions it into logical substructures. The shade of each box indicates how often a keyword occurs in that section of the text: a darker colour represents a part of the document where the given keyword appears often, a lighter shade a section where it appears less often. Vertically aligned boxes represent the same sections in the given document. The user can navigate directly to any section of the document by clicking on the boxes in the icon.

4.2.2 Visualising and Navigating an On-Line Library Catalog

A system for querying, navigating and visualising an on-line library catalog was presented by [VHN96]. An online thesaurus suggests additional query terms to support the user in constructing queries. The system also provides a simple visualisation which displays the total relevance for each document together with the relevance for each keyword (see Figure 4.2). Every search keyword is displayed on the left side of the figure, while right from it there is a row of bars. Each bar represents a document that was retrieved while the height of the bar represents the document's rank for that keyword. A taller bar means that the keyword is more relevant to that document. The last row represents the total rank for each document, which is calculated as a combination of ranks for every search keyword. Documents are ordered from left to right in decreasing order of total rank, where vertically aligned bars represent the same document.

4.3 Visualisation of Multidimensional Data

As multidimensional data sets differin size and dimensionality, different approaches exist trying to provide the best scheme for the data visualisation. A common approach is, for example, visualising documents as matrices of objects in the display, with different metadata values on the X and Y axis, allowing easy identification in trends and patterns in the document set. If the number of dimensions that should be viewed simultaneously is large, more than only two axis at once can be used. Such

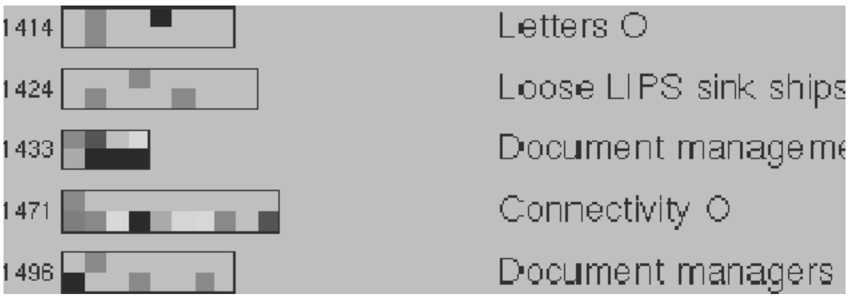


Figure 4.1: TileBars visualisation, taken from [HEA95].

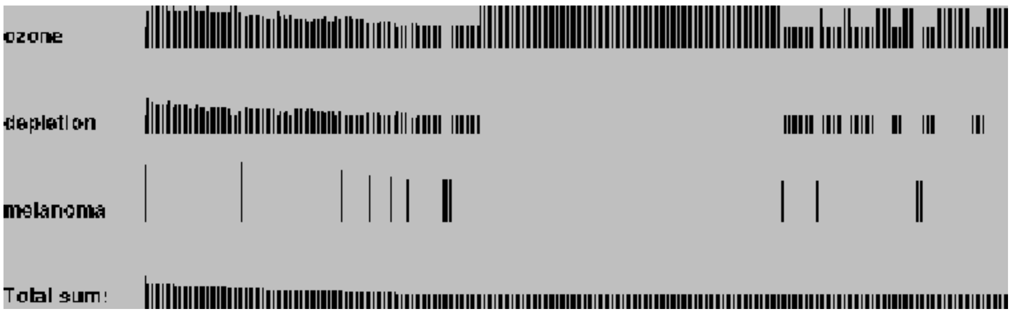


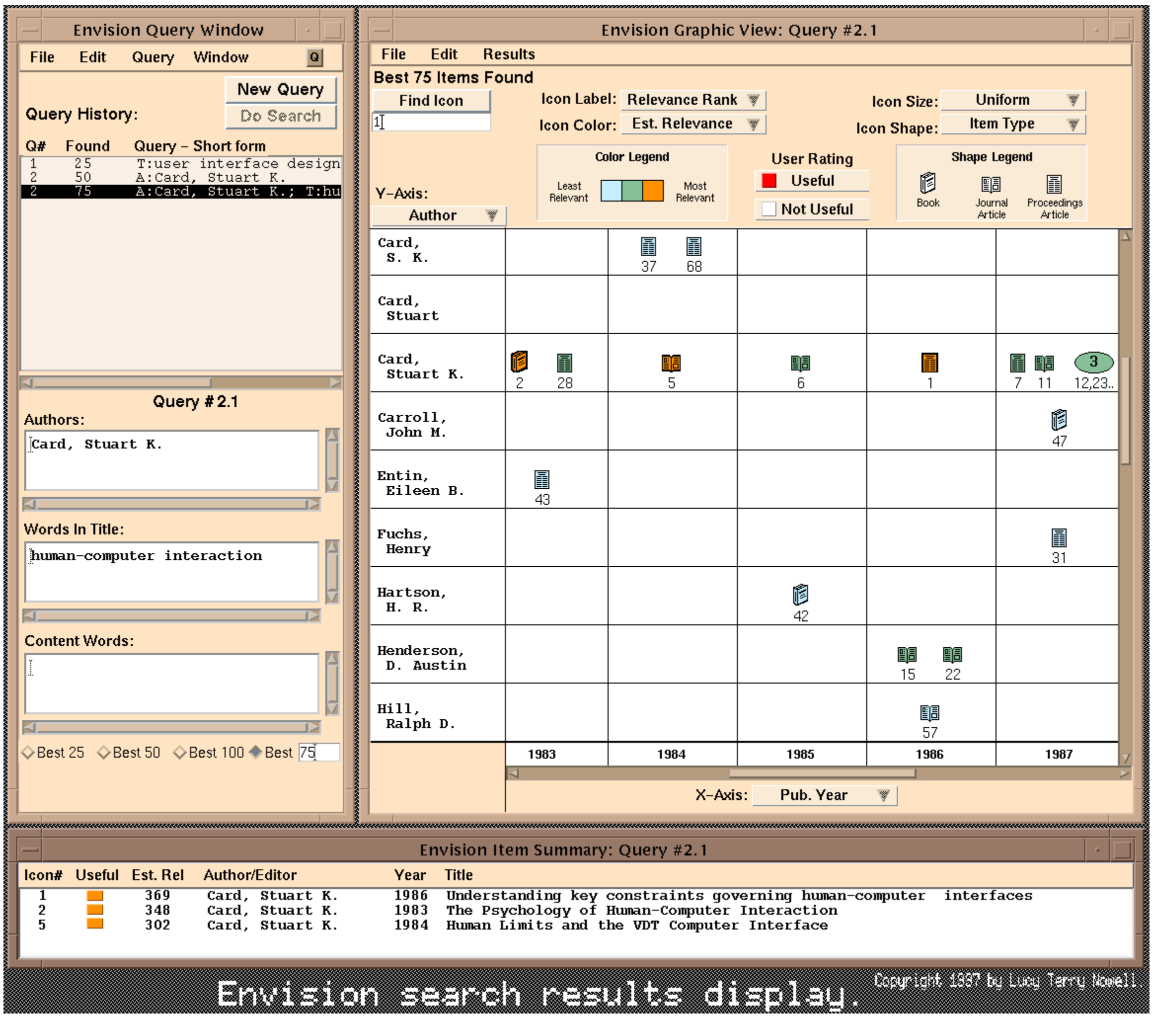
Figure 4.2: Visualising and navigating an on-line library catalog, take from [VHN96].

systems differ in the number of axis used, the way the axis are ordered in the display, and the flexibility of assigning attributes to each axis or to some other characteristic of the display objects.

4.3.1 Envision

Envision [NFH96] is a multimedia digital library of computer science literature, with searching and retrieval possibilities. The Envision system is made up of four main parts: a query server, an object-oriented database management system, a presentation server, and the Envision client. The user interface was designed to give the user full control of layout semantics and it allows the exploration of patterns in the literature by choosing and combining different document characteristics to be visualised. An extensive range of user actions is possible, such as accessing bibliographic information, abstracts, and viewing full content by double clicking the document icon. The Envision client window (see Figure 4.3) is divided into three main parts:

- The Query Window: The query window provides the possibility to search for authors, words in title and content words. The search interface maintains a query history, so the user can edit and reuse old queries.
- The Graphic View Window: contains the graphical representation of the search result set, the icon matrix.
- The Item Summary Window: provides bibliographic information in form of a list, corresponding to marked icons in the Graphic View Window.



Envision Query Window

File Edit Query Window

New Query

Do Search

Query History:

Q#	Found	Query – Short form
1	25	T:user interface design
2	50	A:Card, Stuart K.
2	75	A:Card, Stuart K.; T:hu

Query #2.1

Authors: Card, Stuart K.

Words In Title: human-computer interaction

Content Words:

◇ Best 25 ◇ Best 50 ◇ Best 100 ◆ Best 75

Envision Graphic View: Query #2.1

File Edit Results

Best 75 Items Found

Find Icon

Icon Label: Relevance Rank

Icon Color: Est. Relevance

Icon Size: Uniform

Icon Shape: Item Type

Y-Axis: Author

X-Axis: Pub. Year

	1983	1984	1985	1986	1987
Card, S. K.		37 68			
Card, Stuart					
Card, Stuart K.	2 28	5	6	1	7 11 12,23..
Carroll, John M.					47
Entin, Eileen B.	43				
Fuchs, Henry					31
Hartson, H. R.			42		
Henderson, D. Austin				15 22	
Hill, Ralph D.				57	

Envision Item Summary: Query #2.1

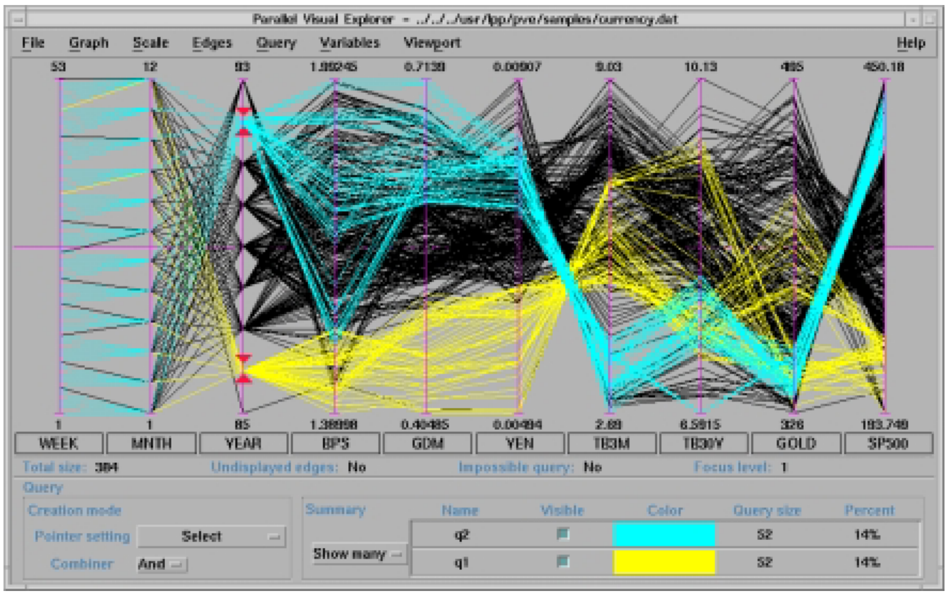
Icon#	Useful	Est. Rel	Author/Editor	Year	Title
1	🟧	369	Card, Stuart K.	1986	Understanding key constraints governing human-computer interfaces
2	🟧	348	Card, Stuart K.	1983	The Psychology of Human-Computer Interaction
5	🟧	302	Card, Stuart K.	1984	Human Limits and the VDT Computer Interface

Envision search results display.

Copyright 1997 by Lucy Terry Nowell

Figure 4.3: The Envision client, taken from [NFH96].

In the graphic view window, the search results are displayed as a matrix of icons. Each icon represents one (or more) documents returned as a search result and its position in the matrix depends on the document characteristics as well as on semantics of the axis chosen by the user. The user has complete control over the semantics of each axis and is able to view every possible combination of document characteristics by simply choosing which characteristic is displayed on each axis. An axis can represent attributes like estimated relevance, author names, publication years, index terms, document type etc. Changing icon attributes such as size, colour or shape the user can gain even more information about each document. Following choices are possible: icon size - relevance, icon colour - relevance or document type, icon shape - document type, label associated with each icon - relevance rank or a unique Envision document identifier. The most relevant document is the first icon in the cell. If icons overlap, they are combined to an elliptical cluster icon that shows the number of documents represented by it. The user can zoom-in to gain access to documents contained in such a cluster icon. Usability evaluation results showed great user satisfaction with Envisions style of search results presentation.



Total size: 384 Undisplayed edges: No Impossible query: No Focus level: 1

Query

Creation mode

Pointer setting: Select

Combiner: And

Summary: Show many

Name	Visible	Color	Query size	Percent
q2	▪	🟦	52	14%
q1	▪	🟨	52	14%

Figure 4.4: The Parallel Visual Explorer displaying 10 parallel coordinates, taken from [CHT95].

4.3.2 Parallel Coordinates

Often the number of dimensions that should be displayed is much higher than the number of spatial dimensions that are displayable on screen. This problem can be addressed by coding the extra dimensions into some attributes of display objects, like size, colour or shape, but this approach is also limited. This method, called parallel coordinates, was described and invented by [IND87].

Parallel Visual Explorer (PVS) [CHT95] data analysis software, enables the visualisation of datasets with very many variables enabling the user to easily discover relations among the them (see Figure pvevisfig). PVS displays several dimensions as parallel vertical axes. For every display object the data values for each dimension are then displayed on a corresponding axis. The number of dimensions displayed at once is limited by the number of parallel axis on the screen. For every multi-dimensional object n values $x1, x2, ..., xn$ are marked on axis 1 to n, and these points are than connected by lines each object is visualised as a polygonal line. Parallel Visual Explorer is a powerful visualisation tool for discovering patterns and similarities for every kind of multidimensional data. Relationships between objects can be easily identified as their lines have, locally or globally, similar shapes.

4.4 Systems Supporting the User in Building the Search Query

A common problem for the user is how to describe best what she or he is looking for, as the used terminology is often very broad or it is only known to experts. Constructing a query is crucial to obtaining good search results. Thus, it should be a creative process where the system supports the user, proposing a refinement of previous queries with feedback from previously returned objects, analysing the user's earlier behaviour, etc. In this way the user not only gains access to information that was not retrieved with the original query, but also learns about the subject and the terminology commonly used in the subject of interest, and acquires the ability to form better and more exact queries.

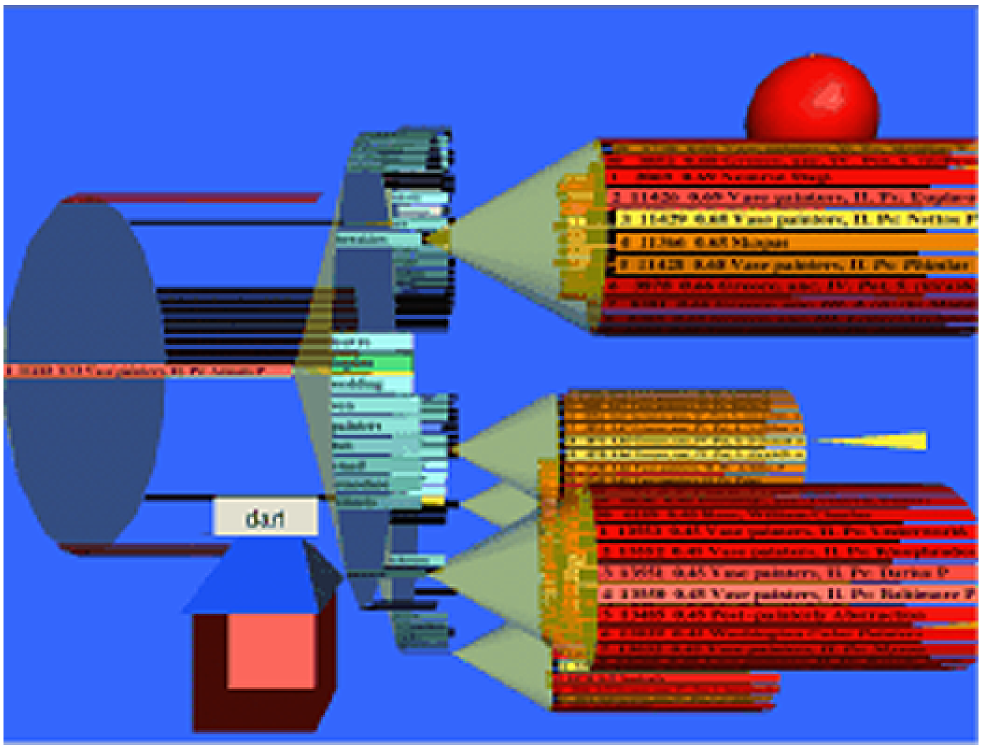


Figure 4.5: Screenshot of LyberWorld showing the NavigationCones visualisation, taken from [LYB01].

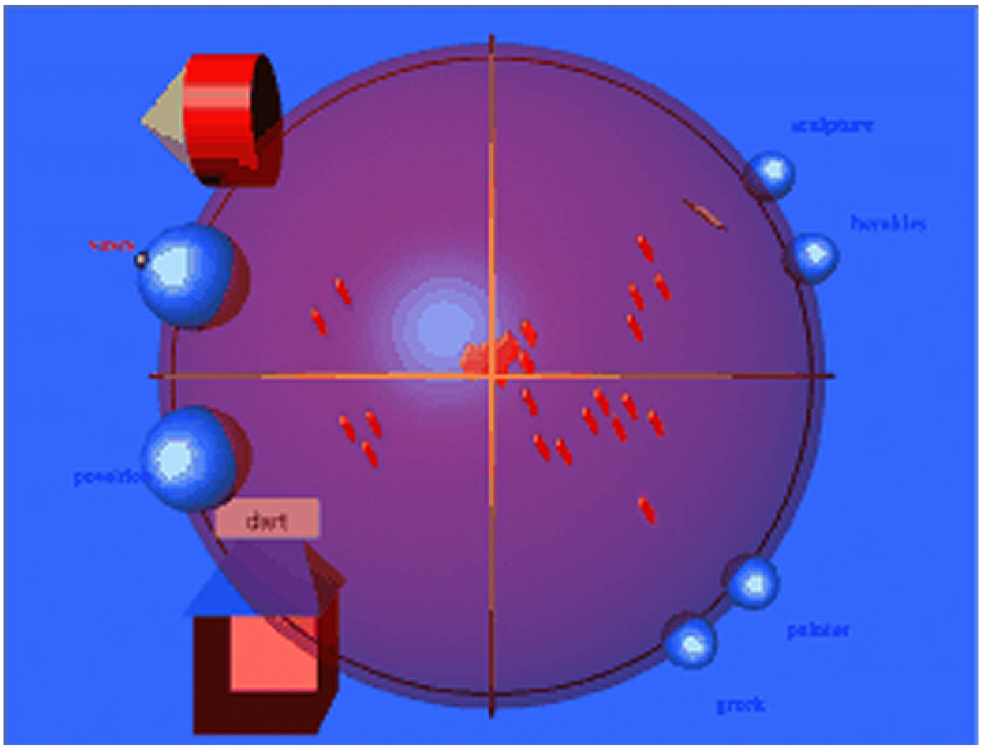


Figure 4.6: Screenshot of LyberWorld showing the RelevanceSphere visualisation, taken from [LYB01].

4.4.1 LyberWorld

The LyberWorld [HKW94] search visualisation system offers two main visualisations, the NavigationCone and the RelevanceSphere. NavigationCones (see Figure 4.5) visually supports the user in forming the search query and also gives a visualisation of the retrieval history showing how far the data space has been explored. The first level LyberTree is a cone-tree which is built as a result of a standard search query and it contains all documents found by the specified query. The returned documents are represented using labels and are placed in the 3-D space so that more relevant documents are shown in front, closer to the view point. The user can rotate the tree to view and select the documents that were classified as less relevant and were originally displayed behind the more relevant ones and eventually covered by them. If the number of documents is very large a spiral structure can be used instead of a cone. The construction of the search query can now be continued and refined by clicking on an interesting document. This opens a term tree containing the keywords extracted from the current document. Selecting a keyword starts a new, refined search query and the retrieved documents form a new cone-tree, which is appended to the already present structure. This procedure enables the user to refine the search query step by step and simultaneously follow the whole retrieval and query history. Any of the retrieved documents can be opened using the LyberRoom, a 3-D visualisation of a virtual room where the documents are projected on a wall for viewing.

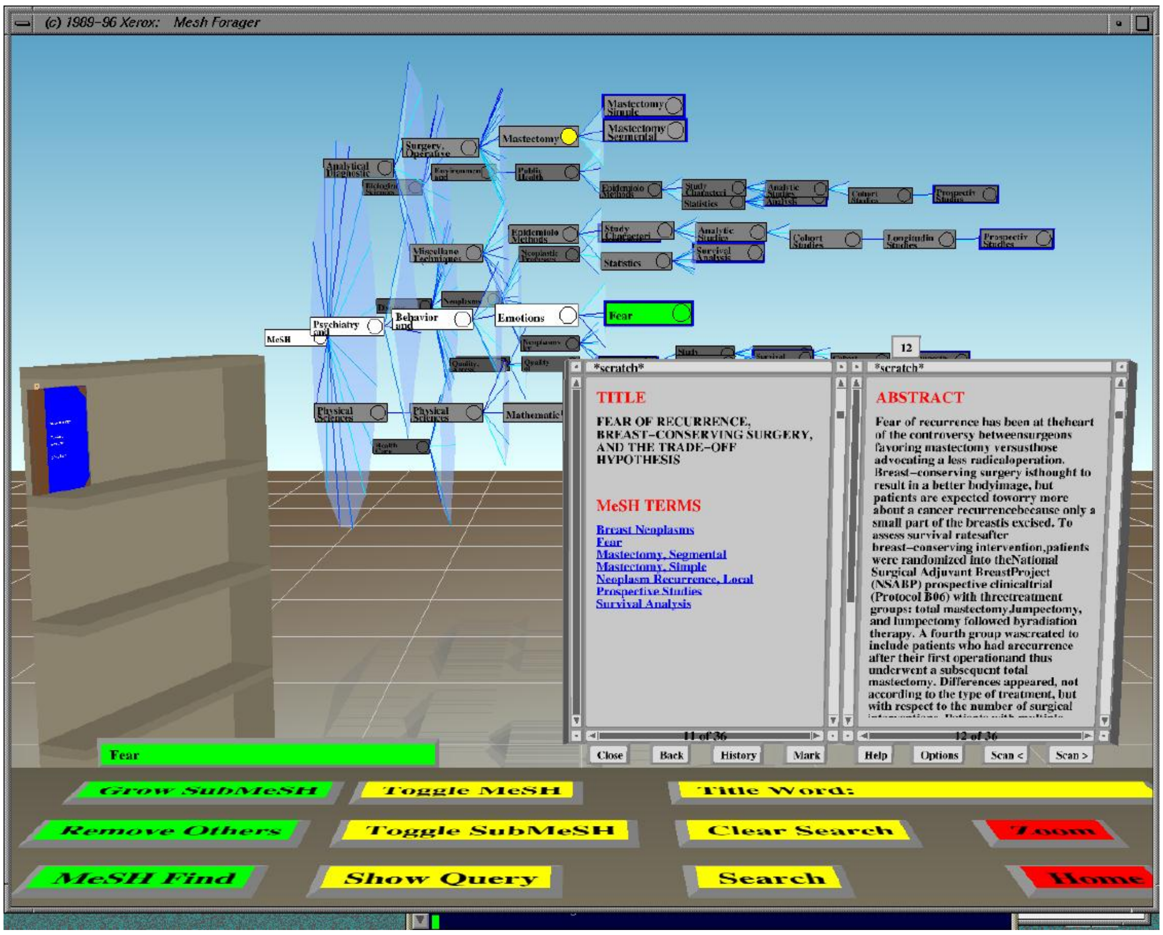
The second visualisation offered by the system is the RelevanceSphere (see Figure 4.6) which is used to display the relevance of the documents with respect to each of the search terms in the query. In this way the documents are visually clustered enabling the user to determine which document is related to which query term as well as which documents are related themselves. All documents are displayed inside of a sphere while the search query terms are placed on the surface and are represented by small spheres. The distance of the document to the surface of the sphere represents the overall relevance, so relevant documents are placed closer to the surface, and those less relevant closer to the center of the sphere. The documents are also attracted by the query terms spheres depending on their keyword profiles. By moving the query term spheres over the surface of the RelevanceSphere the documents are automatically reclustered and placed to new positions, following only certain query term spheres. The user can easily decide which documents are relevant to which query term.

4.5 Systems Visualising Document Hierarchies

Documents sre often organised and stored in hierarchy or a hierarchical structure can be dynamically built after a search is performed, with the goal of organising the found documents. Very large amounts of information can be stored in a hierarchical structure enabling the user to classify it and keep an overview. Systems managing such structures can handle single or multiple memberships of stored objects. However, the number of documents or files that are nowdays commonly stored in such a system is becoming so large that hierarchies could not be displayed in whole any more, and new methods of visualising and navigating had to be developed. A hierarchy that is too large to fit on screen forces the user to scroll and the visualisation fails to give an overview that is simple and easy to understand and navigate. Different approaches, some using 2-D and some 3-D geometry, have been developed to visualise hierarchies with a very large number of nodes.

4.5.1 Cat-a-Cone

Cat-a-Cone [HEK97] is an interactive system for specifying searches and visualising retrieval results using large category hierarchies (see Figure 4.7). It enables the user to simultaneously display retrieved documents and multiple categories in the category hierarchy to which the documents belong.



TITLE

FEAR OF RECURRENCE, BREAST–CONSERVING SURGERY, AND THE TRADE–OFF HYPOTHESIS

MeSH TERMS

Breast Neoplasms
Fear
Mastectomy, Segmental
Mastectomy, Simple
Neoplasm Recurrence, Local
Prospective Studies
Survival Analysis

ABSTRACT

Fear of recurrence has been at theheart of the controversy betweensurgeons favoring mastectomy versusthose advocating a less radicaloperation. Breast–conserving surgery isthought to result in a better bodyimage, but patients are expected toworry more about a cancer recurrencebecause only a small part of the breastis excised. To assess survival ratesafter breast–conserving intervention,patients were randomized into theNational Surgical Adjuvant BreastProject (NSABP) prospective clinicaltrial (Protocol B06) with threetreatment groups: total mastectomy,lumpectomy, and lumpectomy followed byradiation therapy. A fourth group wascreated to include patients who had arecurrence after their first operationand thus underwent a subsequent total mastectomy. Differences appeared, not according to the type of treatment, but with respect to the number of surgical

Figure 4.7: Cat-a-Cone system, taken from [HEK97].

Documents inserted into the category hierarchy are classified according to general subject areas and other semantic attributes. The documents retrieved as a result of a search query are stored in a so-called WebBook. This is a 3-D animated book with scrollable pages, with the user's query shown on the cover of the produced WebBook as its title. A document can belong to more than only one category in the category hierarchy, which is displayed as a ConeTree behind the WebBook. The user can move through the pages of the book and watch the changes caused by his action happening in the category hierarchy representation. These changes are animated in 3-D. When the user turns the pages of the book, the subtrees of the category hierarchy rotate, expand, and contract according to which categories the current document belongs to. Only those categories, together with their ancestors, to which the currently viewed document belongs to are shown in the category hierarchy. For each document in the 3-D WebBook there are two scrolled pages, the left page containing the title and the links to corresponding categories in the category hierarchy and the right page containing the document's contents and/or its abstract. By clicking on a category on the left page, the corresponding category label in the ConeTree is rotated and displayed in the foreground. At the same time all the labels of its ancestors are highlighted. Books that are results of some previous searches are stored in bookshelf, allowing the user to view the results of the previously performed queries. Cat-a-Cone is implemented in Common LISP, running on Silicon Graphics IRIX machines.

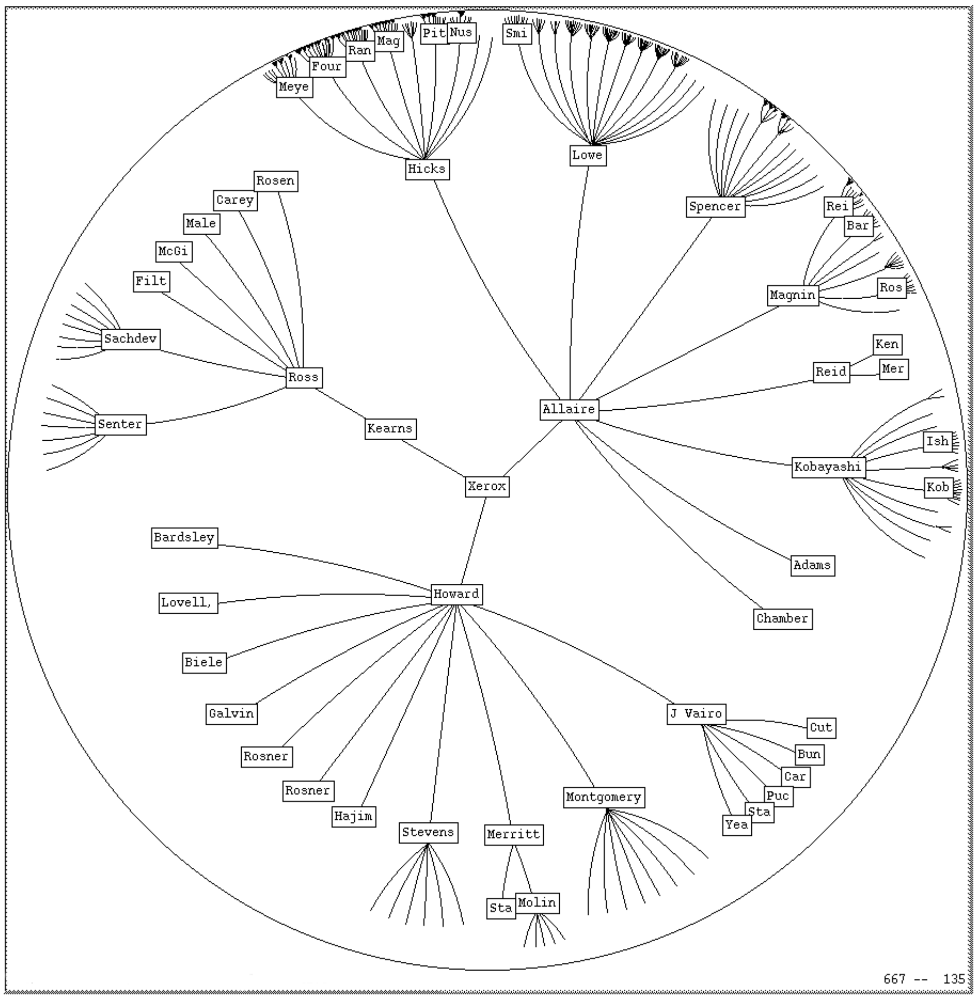


Figure 4.8: Hyperbolic trees, taken from [LRP95].

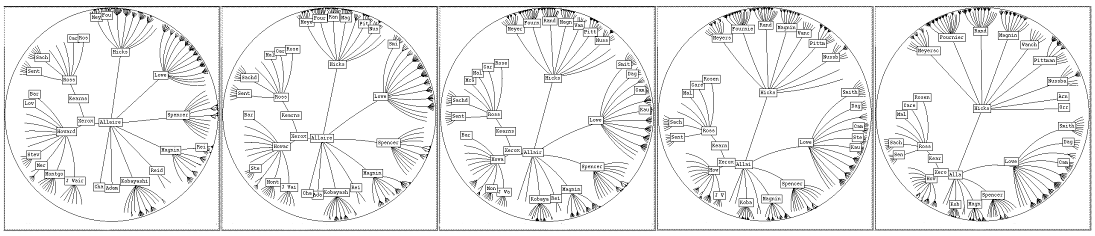


Figure 4.9: Hyperbolic trees: Changing the focus. A node is being brought into focus. As other nodes move away from center the amount of space they are displayed in becomes smaller. Taken from [LRP95].

4.5.2 Hyperbolic Trees

Tree views are probably the simplest method to visualise hierarchies and they are provided as standard graphical components by almost all user interfaces, like Microsoft Windows, the Macintosh File Finder or any X-Windows based window manager. They are well known to everybody who ever worked with modern computers and therefore need not be explained into detail. Normally not all subtrees of the hierarchy are expanded at the beginning. The user navigates the tree by choosing which subtrees to expand or collapse, going as deep or as wide in the hierarchy as needed. The greatest disadvantage of these systems is that by opening subtrees the displayed part of the hierarchy expands and needs an ever bigger display area. In such a case scrollbars are used to access parts that are currently not visible, but a complete overview of the structure of the entire hierarchy is lost. This is a huge problem when managing very large hierarchies and it mostly leads to the user becoming lost and loosing orientation.

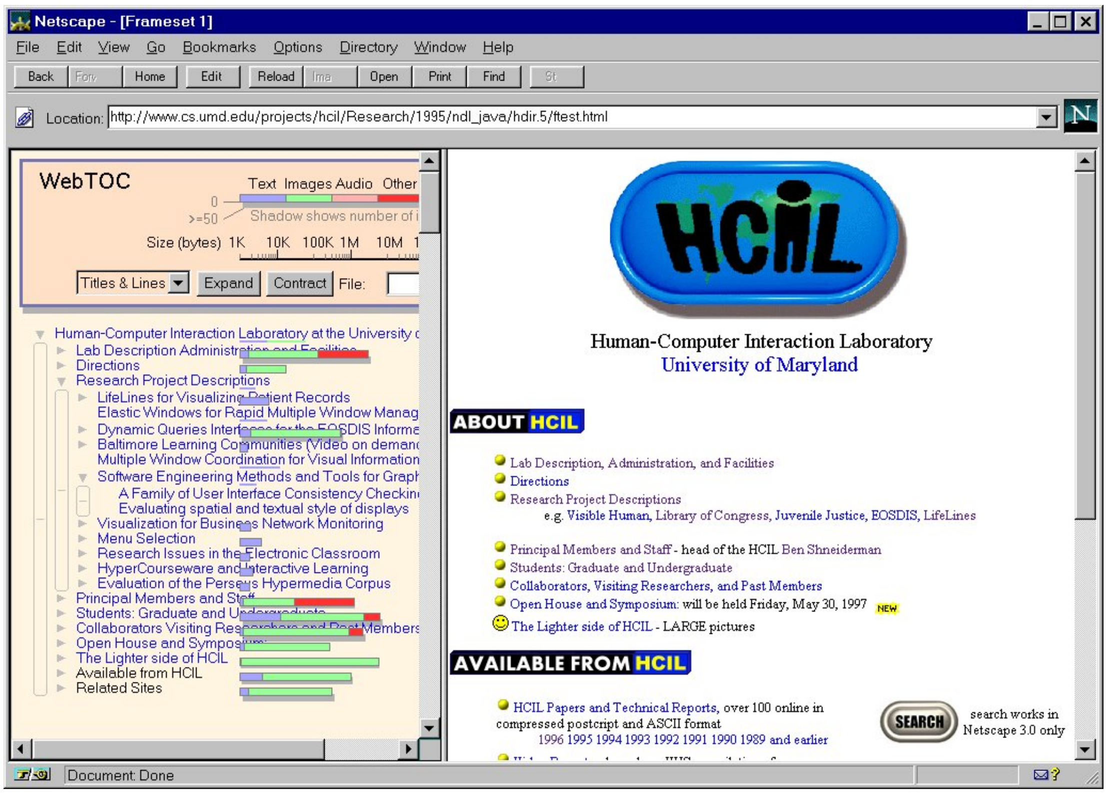
Hyperbolic trees [LAR94] make use of hyperbolic geometry for visualising large hierarchies (see Figures 4.8 and 4.9). A large part of the display area shows only a small, focused, part of the hierarchy, but still the entire hierarchy is displayed. The part of the hierarchy that the user is concentrating on is displayed large and detailed, while the rest of the hierarchy is small and less detailed. This technique allows smooth, animated focusing on a part of the hierarchy that the user is interested in by magnifying the components that move towards the center, while other components become smaller as they move away from the center. It is an example of the principle of focus + context. The hyperbolic browser can display up to 10 times as many nodes as conventional tree representation and at the same time provides navigation around the hierarchy with mouse clicks and dragging, making it easy for the user to explore the hierarchy without getting lost in it. As a only 2-D geometry is used the required computing power is kept low which makes the system suitable for implementing on any platform including Java.

Implementations of this technique are developed by InXight Software Inc.,a subsidiary of Xerox Corp. Applications for visualisation of data organization using hyperbolic trees [IHT01] were developed, enabling the user to explore the Web and his computer's contents while simultaneously viewing the structure of the Web and the directory structure on his computer. The user can perform different operations such as searching for and identifying broken links. Another interesting tool by Inxight is Murax [IMU01], designed for searching and browsing of query results based on topic and content similarities. It uses language processing technology to analyze document content and establish relationships between the search words and phrases and to produce results by similar subject, concept or phrase. Results are displayed grouped by subjects, in a form of virtual table of contents, starting with those that best match the search query.

4.6 Visualising the Hyperlinks Connecting Retrieved Documents

Most web search engines present search results as a ranked list of titles and/or URLs, possibly together with some additional data like the size, date, number of contained pictures and similar. The contexts in which the pages exist, and their relationships to one another, are not displayed. The following systems take additionally into account the connectivity of the found documents, which is the links from and to the document. Some systems follow these links trying to find more relevant documents which were possibly skipped by the search engine or were erroneously classified as not relevant to the query.

A variety of techniques is used to produce the visualisations. As the hyperlinks define an directed graph on the search results set, a hierarchical structure is often used to organize the returned documents and display them in a tree-like structure or in a virtual table of contents. Iterative placement


Netscape - [Frameset 1]
File Edit View Go Bookmarks Options Directory Window Help
Back Forw Home Edit Reload Ima Open Print Find St
Location:
WebTOC
Text Images Audio Other
0
>=50
Shadow shows number of i
Size (bytes) 1K 10K 100K 1M 10M 1
Titles & Lines
Expand
Contract
File:
Document: Done

http://www.cs.umd.edu/projects/hcil/Research/1995/ndl_java/hdir.5/ftest.html

Human-Computer Interaction Laboratory at the University c
Lab Description Administration and Facilities
Directions
Research Project Descriptions
LifeLines for Visualizing Patient Records
Elastic Windows for Rapid Multiple Window Manag
Dynamic Queries Interfaces for the EOSDIS Informa
Baltimore Learning Communities (Video on demand
Multiple Window Coordination for Visual Information
Software Engineering Methods and Tools for Graph
A Family of User Interface Consistency Checkin
Evaluating spatial and textual style of displays
Visualization for Business Network Monitoring
Menu Selection
Research Issues in the Electronic Classroom
HyperCourseware and Interactive Learning
Evaluation of the Perseus Hypermedia Corpus
Principal Members and Staff
Students: Graduate and Undergraduate
Collaborators Visiting Researchers and Past Members
Open House and Symposium
The Lighter side of HCIL
Available from HCIL
Related Sites

HCIL

Human-Computer Interaction Laboratory
University of Maryland

ABOUT HCIL

- Lab Description, Administration, and Facilities
- Directions
- Research Project Descriptions
  e.g. Visible Human, Library of Congress, Juvenile Justice, EOSDIS, LifeLines
- Principal Members and Staff - head of the HCIL Ben Shneiderman
- Students: Graduate and Undergraduate
- Collaborators, Visiting Researchers, and Past Members
- Open House and Symposium: will be held Friday, May 30, 1997 NEW
- The Lighter side of HCIL - LARGE pictures

AVAILABLE FROM HCIL

- HCIL Papers and Technical Reports, over 100 online in compressed postcript and ASCII format
  1996 1995 1994 1993 1992 1991 1990 1989 and earlier

SEARCH search works in Netscape 3.0 only

Figure 4.10: The WebTOC system, taken from [WET01].

models are also used, with the number of links connecting the documents, called the connectivity, used as a measure of relatedness between documents.

4.6.1 WebTOC

WebTOC [NPM97] is a tool which visualises and quantifies web sites using a hierarchical table of contents (see Figure 4.10). It is made up of two parts: the WebTOC Parser and the WebTOC Viewer. Both are implemented in Java, the first as an application the second as an applet that it can be loaded on the fly by a browser. The Parser begins its search from a chosen Web page and follows all the links contained in this document. By doing so, a hierarchical representation of the documents found is created. The Viewer organizes the collected information in form of a Table of Contents, which can be viewed with standard Web browsers. Due to memory and time limitations the number of links that can be represented is about 10,000. Every line in this Table of Contents represents a link to a document that can be displayed by simply clicking on it. Additionally each document is represented by a coloured line with the length corresponding the document's size, while the colour of the line tells the user what type of file it is, for example text, image or audio. If a document contains links to other documents, the lines representing the documents it includes can be collapsed into a size bar that shows the size of all the documents that can be reached through the links contained in it. This size bar also has a shadow whose size represents the number of the document's children.

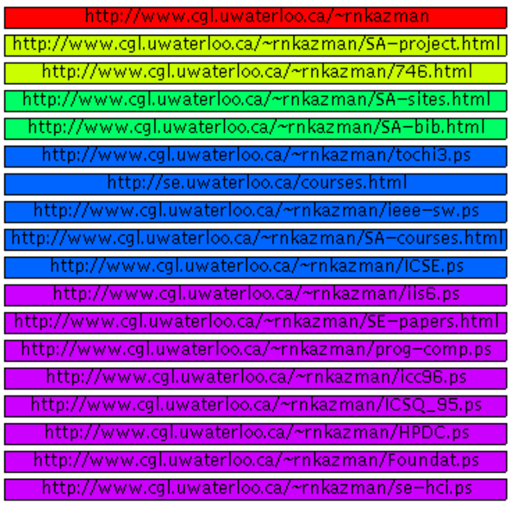
4.6.2 WebQuery

The WebQuery system [CAK97] is designed for searching and visualising the Web based on both connectivity and content. This is done by examining and following links among the documents which are returned as a result of a query, where the document that is most highly connected gets the highest rank. WebQuery extends the result set beyond documents returned by the search engine, by collecting and including documents that are highly connected to those returned by the original query. In this way searching is extended from being content-based to being also structure-based.

Results of the search engine are processed by the VANISH visualisation tool, which extends the original search results with the structural information collected by its own spider. The original hit set is expanded into a so-called neighbour set by finding all nodes that are linking to or are linked by the documents in the hit set. The node's total number of incoming and outgoing links, which is called connectivity, is computed, and the nodes are sorted in decreasing order of connectivity. In this way nodes that were not returned by the search engine are also considered. It has been found out that these nodes are often also relevant to the query, although they were not returned by the search engine because of a vocabulary problem, or because they were simply missed by the search engine's spider. Many uninteresting nodes matching the query are filtered out by the system. They are ranked so low that they are not visualised, or they are pushed away from the center.

WebQuery offers five different visualisations:

- The sorted list view (see Figure 4.11) shows nodes that have equal connectivity in the same colour.
- In bullseye view (see Figure 4.12) each circle contains nodes with equal connectivity, which is represented by a number inside of it. Selecting a node causes URL to appear.
- The planes view (see Figure 4.13) shows a 3-D visualisation where nodes are ordered in planes according to their connectivity and links are drawn between hyperlinked nodes. Those nodes that were returned by the search engine are shown as tetrahedra, while those found by WebQuery are shown as cubes.
- The cone tree view (see Figure 4.15) is constructed by traveling through the structure of hyperlinks, beginning at the artificial root node and adding nodes to the tree in the order in which they were collected. Each parent node is labeled by the connectivity value of its children. Nodes from the hit set are coloured according to their score returned by the search engine, where red indicates highest and blue indicates the lowest score. Nodes that did not appear in the original hit set are coloured black.
- The graph view (see Figure 4.14) uses a springs-and-weights algorithm that simulates the forces between nodes and calculates the positions of the nodes. The artificial parent nodes are shown in blue-green and are labeled with the connectivity values of their children. The used algorithm pulls nodes with high connectivity to the center, while those with low connectivity are pushed away from center. By default all nodes repel each other and only the connections between them result in forces that attract nodes that are connected with each other by links.



http://www.cgl.uwaterloo.ca/~rnkazman
http://www.cgl.uwaterloo.ca/~rnkazman/SA-project.html
http://www.cgl.uwaterloo.ca/~rnkazman/746.html
http://www.cgl.uwaterloo.ca/~rnkazman/SA-sites.html
http://www.cgl.uwaterloo.ca/~rnkazman/SA-bib.html
http://www.cgl.uwaterloo.ca/~rnkazman/tochi3.ps
http://se.uwaterloo.ca/courses.html
http://www.cgl.uwaterloo.ca/~rnkazman/ieee-sw.ps
http://www.cgl.uwaterloo.ca/~rnkazman/SA-courses.html
http://www.cgl.uwaterloo.ca/~rnkazman/ICSE.ps
http://www.cgl.uwaterloo.ca/~rnkazman/iis6.ps
http://www.cgl.uwaterloo.ca/~rnkazman/SE-papers.html
http://www.cgl.uwaterloo.ca/~rnkazman/prog-comp.ps
http://www.cgl.uwaterloo.ca/~rnkazman/icc96.ps
http://www.cgl.uwaterloo.ca/~rnkazman/ICSQ_95.ps
http://www.cgl.uwaterloo.ca/~rnkazman/HPDC.ps
http://www.cgl.uwaterloo.ca/~rnkazman/Foundat.ps
http://www.cgl.uwaterloo.ca/~rnkazman/se-hci.ps

Figure 4.11: WebQuery: sorted list view, taken from [CAK97].

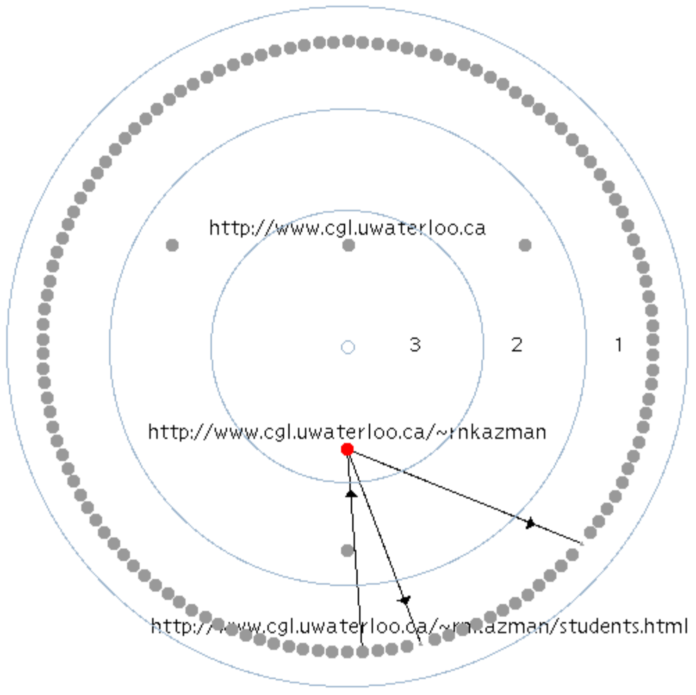

http://www.cgl.uwaterloo.ca
3
2
1
http://www.cgl.uwaterloo.ca/~rnkazman
http://www.cgl.uwaterloo.ca/~rnkazman/students.html

Figure 4.12: WebQuery visualisation: bullseye view, taken from [CAK97].

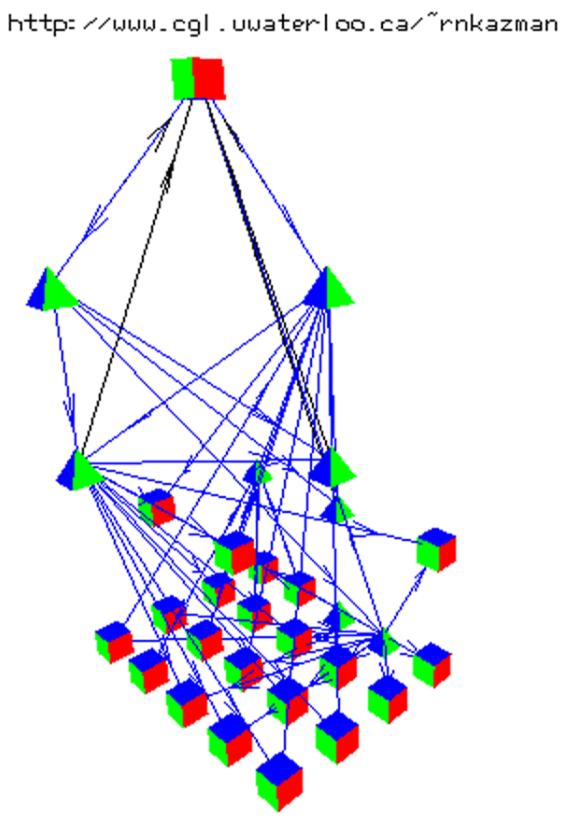


Figure 4.13: WebQuery: the planes view, taken from [CAK97].

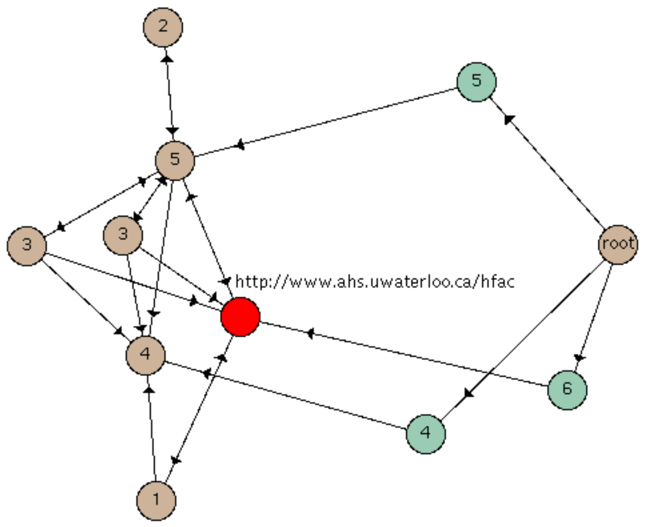


Figure 4.14: WebQuery graph visualisation, taken from [CAK97].

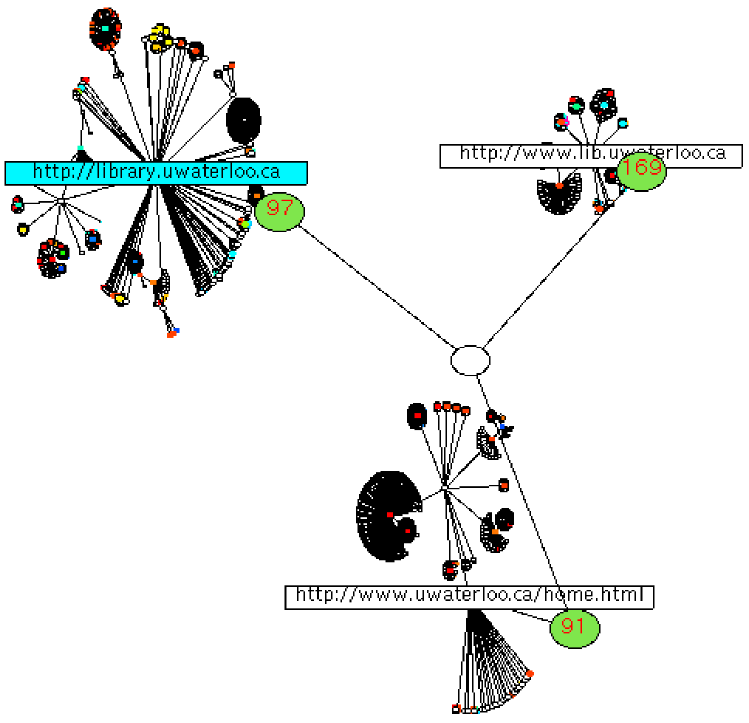

http://library.uwaterloo.ca
97
http://www.lib.uwaterloo.ca
169
http://www.uwaterloo.ca/home.html
91

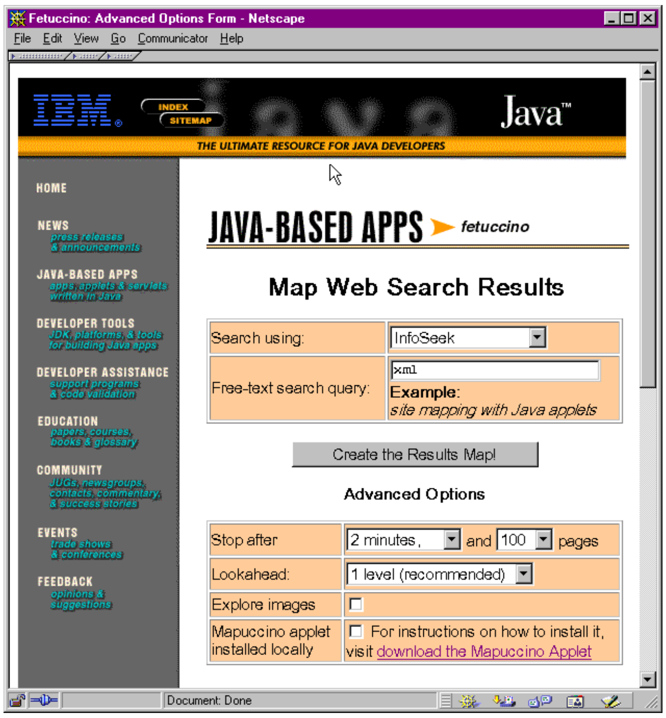
Figure 4.15: WebQuery: 2D cone tree visualisation, taken from [CAK97].

4.6.3 Fetuccino

Fetuccino [BHJ99] is an extension of the Mapuccino site mapping system and provides a user interface for visualisation of search results as an advanced graph layout. The system offers the following possibilities:

- Extending the standard static search with a dynamic search that explores the neighbourhood of the returned documents to find more relevant documents, check for broken links and even include dynamically generated documents.
- A dynamic two-phase search that allows the user to specify a domain query and a focused query, and apply the focused search in the specified domain.
- Visualising the results of the search process, showing the structure and relations of the returned documents in relation to the sites where they were found.

A standard search engine is used to obtain a set of documents in response to a user's search query. These documents are then analyzed and used as a starting point for the dynamic searching process. New documents are found by following hyperlinks from the original set of documents. The user can specify the depth of the search and the system will follow the hyperlinks of the documents retrieved by the dynamic search up to a specified depth (see Figure 4.16). As this searching process is time-consuming, the user can also specify a time limit after which the dynamic search stops.


Fetuccino: Advanced Options Form - Netscape
File Edit View Go Communicator Help
IBM
INDEX
SITEMAP
Java™
THE ULTIMATE RESOURCE FOR JAVA DEVELOPERS
HOME
NEWS
press releases & announcements
JAVA-BASED APPS
apps, applets & servlets written in Java
DEVELOPER TOOLS
JDK, platforms, & tools for building Java apps
DEVELOPER ASSISTANCE
support programs & code validation
EDUCATION
papers, courses, books & glossary
COMMUNITY
JUGs, newsgroups, contacts, commentary, & success stories
EVENTS
trade shows & conferences
FEEDBACK
opinions & suggestions
JAVA-BASED APPS
fetuccino
Document: Done

Map Web Search Results

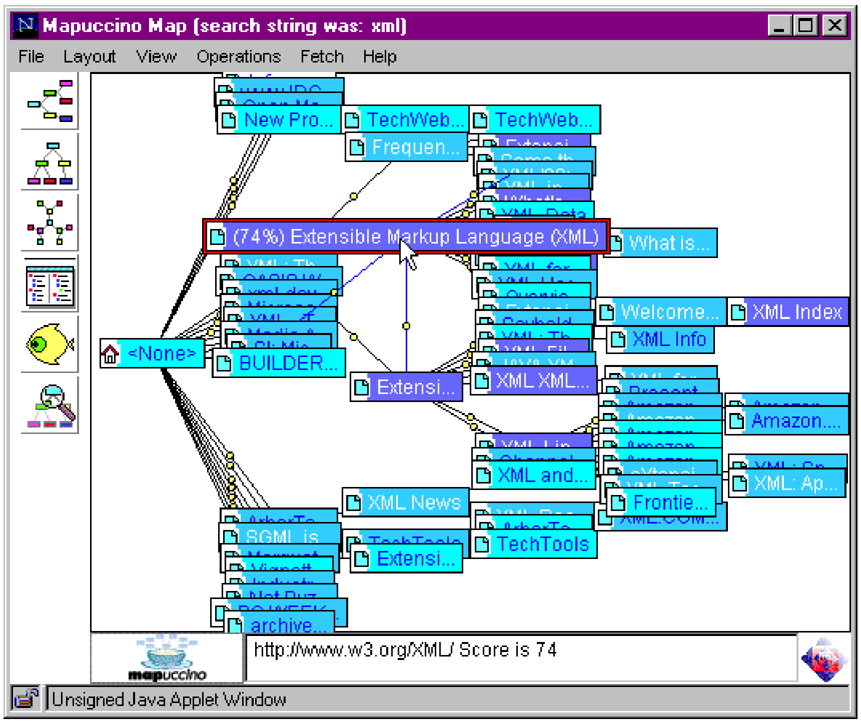
Search using:	InfoSeek
Free-text search query:	xml **Example:** *site mapping with Java applets*

Create the Results Map!

Advanced Options

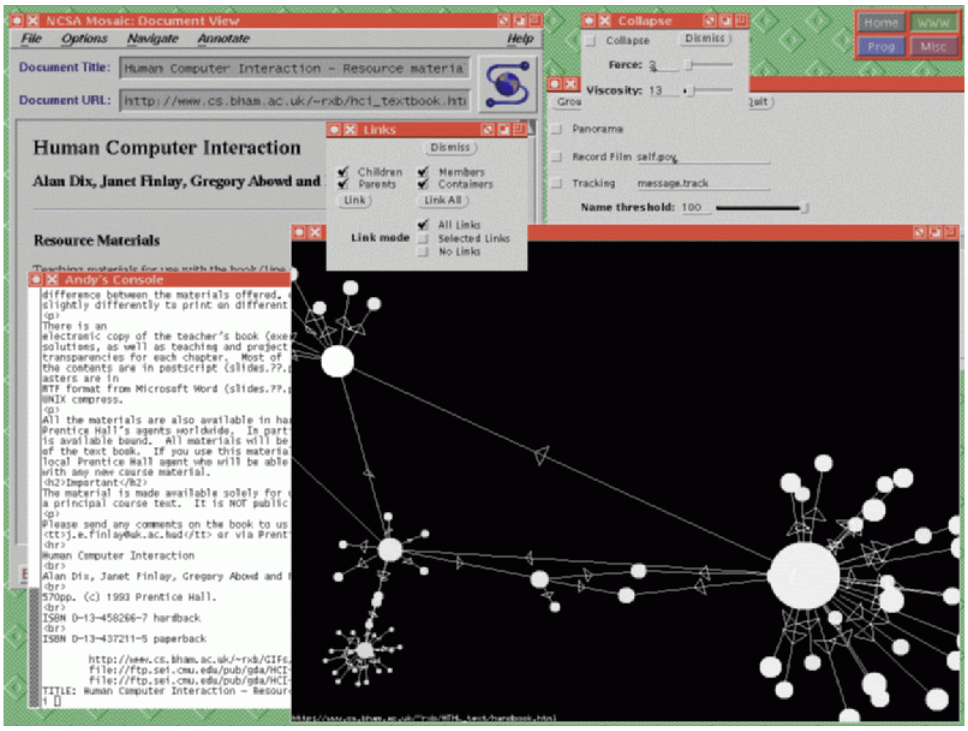
Stop after	2 minutes, and 100 pages
Lookahead:	1 level (recommended)
Explore images	☐
Mapuccino applet installed locally	☐ For instructions on how to install it, visit download the Mapuccino Applet

Figure 4.16: The Advanced Form in the Fetuccino service (light version), taken from [BHJ99].



http://www.w3.org/XML/ Score is 74

Figure 4.17: The Mappucino applet, taken from [BHJ99].



```
difference between the materials offered.
slightly differently to print on different
<p>
There is an
electronic copy of the teacher's book (exe
solutions, as well as teaching and project
transparencies for each chapter.  Most of
the contents are in postscript (slides.??.
asters are in
RTF format from Microsoft Word (slides.??.
UNIX compress.
<p>
All the materials are also available in ha
Prentice Hall's agents worldwide.  In part
is available bound.  All materials will be
of the text book.  If you use this materia
local Prentice Hall agent who will be able
with any new course material.
<h2>Important</h2>
The material is made available solely for
a principal course text.  It is NOT public
<p>
Please send any comments on the book to us
<tt>j.e.finlay@uk.ac.hud</tt> or via Prent
<hr>
Human Computer Interaction
<br>
Alan Dix, Janet Finlay, Gregory Abowd and
<br>
570pp. (c) 1993 Prentice Hall.
<br>
ISBN 0-13-458266-7 hardback
<br>
ISBN 0-13-437211-5 paperback

        http://www.cs.bham.ac.uk/~rxb/GIFs
        file://ftp.sei.cmu.edu/pub/gda/HCI-
        file://ftp.sei.cmu.edu/pub/gda/HCI-
TITLE: Human Computer Interaction - Resour
i
```

Figure 4.18: A screen-shot of the HyperSpace system, taken from [WDB95].

The map view shows the search results page as root, and the documents with the highest relevance as first-level children (see Figure 4.17). The shade of blue represents the relevance of each document, where a darker shade means that the document is more relevant. Following levels in the tree show the documents found by the dynamic searching process. It was found out that some documents with very high relevance were not among those found by the standard search, but were found by the dynamic search process. The system also checks for broken links which can be presented to the user. There is an extended version of Fetuccino, called Fetuccino Alfredo that offers the following extension: The user specifies two terms: a domain term, and a query term. In the first phase, the specified domain is found and in the second phase the query is evaluated inside of it.

4.6.4 HyperSpace

HyperSpace [WDB95] is a World Wide Web tool which adds visualisation to the process of browsing the Web. It displays the organization of the web by considering the hyperlinks connecting the documents, so that related topics are displayed close to each other and those unrelated are pushed away from one another (see CFigure hypspacefig). Every time a page is loaded into the browser it is also loaded into a separate program, that analyses the page and extracts the title and all links. This data is used by the HyperSpace viewer which represents each document as a sphere whose size corresponds to the number of links showing from or to the document. Links between documents are shown as arrows between spheres. Spheres normally repel each other while links provide attractive forces pulling highly interconnected documents together. In this way a 3-D system, where the spheres are randomly placed at the beginning, organises itself so that documents with no links between them are pushed apart, and those that are highly interconnected are clustered together. The viewer displays all links emanating from a document, as well as all known links pointing to it.

4.7 Visualising Document Collections According to Their Content

Documents are usually represented as vectors in a high-dimensional space. This model, called the vector space model (see Section 2.2), is the most commonly used model and allows the comparison of documents with respect to their content. A problem that all systems which visualise documents according to their mutual similarity or to their similarity to search query terms have in common, is how to perform the mapping from the high-dimensional space to the 2-D or 3-D space which can be visualised on screen. Most systems display the documents with the Euclidean distance between documents as a measure for their similarity, although some other approaches also being used, like the use of angular distance. A number of different techniques were developed, many of them iterative (see Section 4.1.4) like multidimensional scaling (MDS), simulated annealing or force-directed placement, others making use of artificial intelligence methods, like Self Organising Maps which are based on neural networks. Most of these techniques have the disadvantage of being very slow or having quadratic execution times and therefore can parctically handle no more than a few thousand documents. This problem led to the development of near linear or linear algorithms such as the Anchored Least Stress used by the SPIRE Project (see Section 4.7.10), or the force-directed placement with stochastic sampling used in Bead (see Section 4.7.8).

4.7.1 InfoViz VRML-Document-Finder

The InfoViz VRML-Document-Finder [IVD01] is a tool for 3-D visualisation of search results (see Figure 4.19). The systems is made up of several modules including a complete document retrieval and analysis system, a system for hierarchical clustering and automatic categorisation of retrieved documents and a 3-D visualisation system. Documents returned by the retrieval system as the result of a search query are analysed and clustered according to their similarity. The clustering system builds a tree representation of the topic hierarchy, where nodes are labeled according to the contents of the documents contained in the corresponding subtrees. The blue platforms are clusters representing subject areas marked by the label term on top. They are arranged in the form of the generated hierarchical tree structure with yellow and red cubes placed on them. The cube elements represent text documents assigned to the subject area underneath them. The colour of the cube gives information if the content of the document cannot be described by a more specific subject term, which is displayed by a red cube, or, if it can, which is displayed by a yellow cube, it will appear at a deeper, more specific, level of the subject hierarchy. The generated structure is displayed in 3-D giving the user the possibility of moving, navigating and exploring the virtual reality environment by free navigation or by through predefined viewpoints. The system requires the user to have a VRML-enabled web browser and consists of three main modules written in Perl:

- CGI control script: controls the program, executes database query and returns an unsorted list of matching document references.
- Information processor: builds up a structured document hierarchy from the unsorted list of document references.
- VRML generator: generates a VRML scene of the document hierarchy.

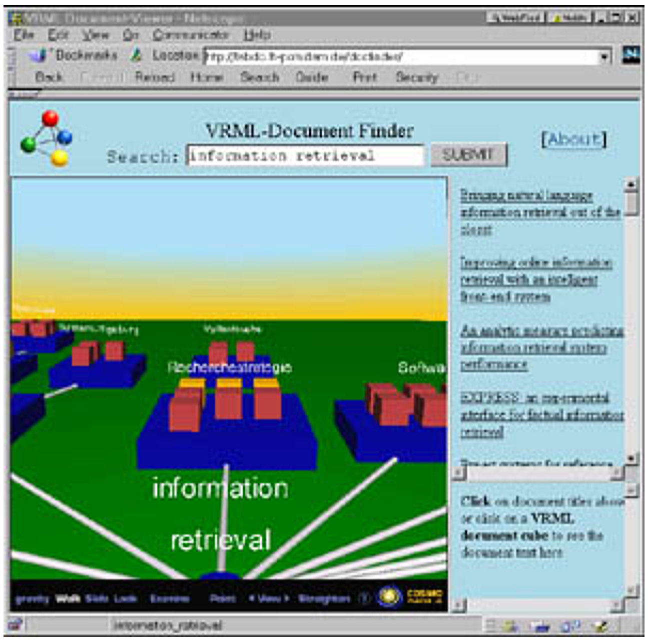

VRML-Document Finder
Search: information retrieval
SUBMIT
[About]
information
retrieval
Recherchestrategie
Click on document titles above or click on a VRML document cube to see the document text here

Figure 4.19: InfoViz VRML-Document-Finder: a VRML scene of document hierarchy. Taken from [IVD01].

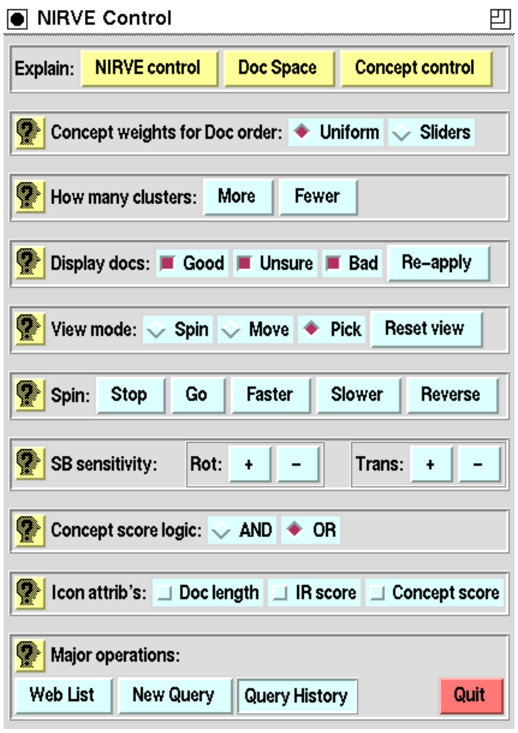

NIRVE Control
Explain: NIRVE control Doc Space Concept control
Concept weights for Doc order: Uniform Sliders
How many clusters: More Fewer
Display docs: Good Unsure Bad Re–apply
View mode: Spin Move Pick Reset view
Spin: Stop Go Faster Slower Reverse
SB sensitivity: Rot: + – Trans: + –
Concept score logic: AND OR
Icon attrib's: Doc length IR score Concept score
Major operations:
Web List New Query Query History Quit

Figure 4.20: The NIRVE Control Window displays a menu for the user to select among the various operations. Taken from [CLP97].

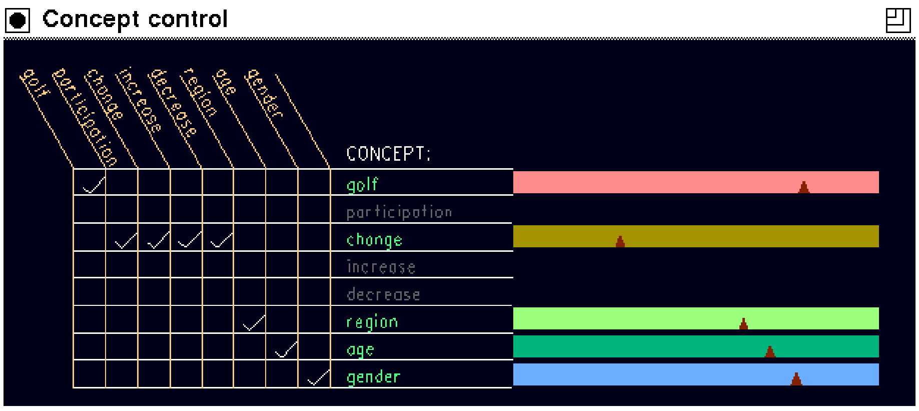
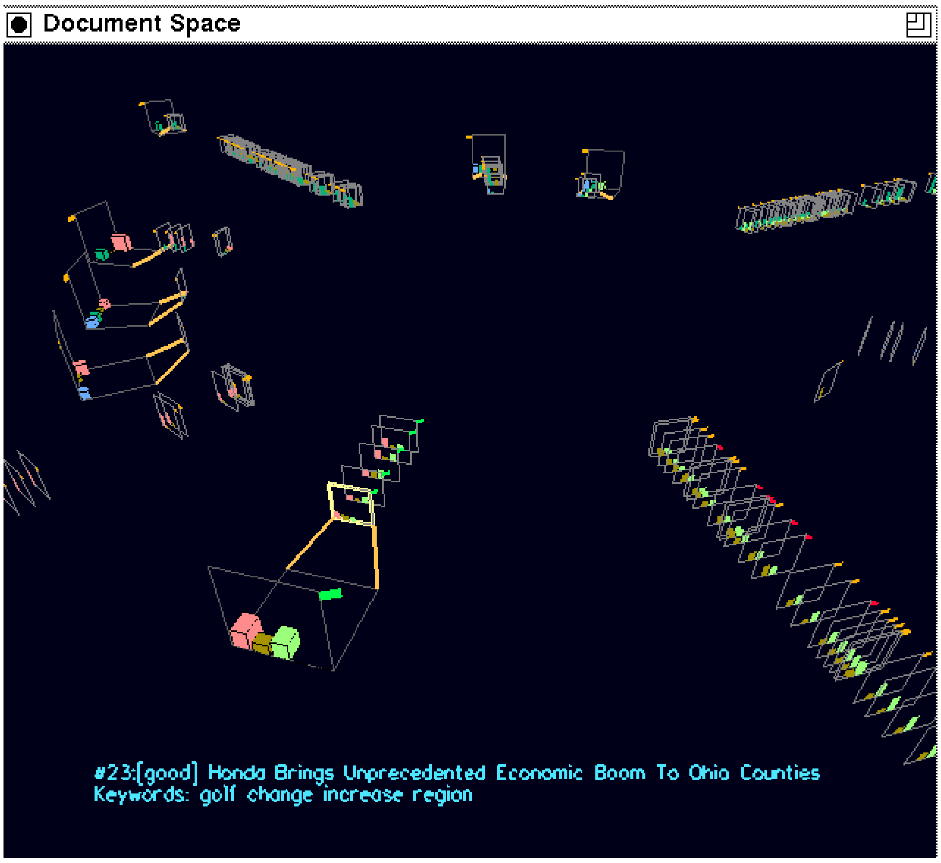


Figure 4.21: The NIRVE Concept Control Window, taken from [CLP97].



#23:[good] Honda Brings Unprecedented Economic Boom To Ohio Counties
Keywords: golf change increase region

Figure 4.22: The NIRVE Document Space Window, taken from [CLP97].

4.7.2 The NIST Information Retrieval Visualisation Engine (NIRVE)

NIRVE [CLP97] is a 3-D visualisation system which takes a set of documents delivered by a search engine as input, enables the user to build up concepts from keywords occurring in the documents, and than builds clusters of related documents according to constructed concepts. NIRVE allows the user to manipulate, browse, recategorise and filter the document set, to assign a relevance to documents and clusters or eliminate them and then view the results with this new set of parameters. The user can choose the number of clusters to be built from the returned set of documents, thus obtaining a smaller number of clusters containing more documents or more clusters containing a larger number of documents. By clicking on a document icon the system invokes a browser and displays the documents contents in it and is even able to automatically generate HTML summaries of a cluster, or of a chosen subset of documents. The following data is needed for performing the clustering and visualisation:

- A relevance score, computed by the search engine, representing how well a document matches the query.
- The sequential rank of the document depending on the score.
- Document title.
- Document URL.
- Document length.
- Number of occurrences of each keyword in the document.

NIRVE displays three windows to the user: the Control Window (see Figure 4.20), the Concept Control Window (see Figure 4.21), and the Document Space Window (see Figure 4.22). The Concept Control window allows the user to construct concepts by mapping query keywords, and to assign a weight to each concept. It is often useful to add synonyms and redundant phrases in the search query, but this creates extra dimensions in the n-dimensional document space, while the meaning of the keywords is the same. This has a negative effect on both quality and speed of clustering the documents. Concept mapping solves this problem by allowing the user to identify such keywords and map them in a smaller set of concepts. Each concept has a different colour assigned to it, so the user can easily distinguish between them.

The Document Space window displays the clusters and the documents inside them. Both the document icons, displayed in 2-D, and the cluster icons, displayed in 3-D, display their concept profile as a histogram where colours of the bars correspond to the colours of certain concepts. The documents are ordered in clusters which are created on the basis of the similarities of document's concept profiles and their weights. Similar documents have similar concept profiles and are placed close to each other inside of the cluster they belong to. The gaps between documents have different sizes according to document's similarities, and a cluster is defined as a maximal sequence of documents where no gap is larger than a specified threshold. User can set this threshold higher or lower to generate less clusters with a larger number of documents or more clusters with a higher number of documents respectively. The angular distance between clusters is proportional to their concept profiles in the concept space, so that similar clusters are placed closer to each other. There are three different modes for manipulating the document space. In spin mode, the display rotates around the center. In move mode, the mouse is used to translate and rotate the displayed clusters. In pick mode, the user can perform different operations on documents or clusters and assign a relevance status to documents and clusters. A small flag attached to the icon represents the current relevance value: red for bad, yellow for unsure, green for good. The user can choose to display only documents or clusters with a specified relevance status and filter out the others. As mentioned before, the system can automatically generate a HTML summary page for any number of chosen documents, for some selected clusters, or for all items in the Document Space window.

A number of other interesting approaches and papers, including the evolution of the NIRVE design [CLS00], can be found at the NIRVE homepage [NIR01].

4.7.3 Wilmascope

Wilmascope [WLM01] is a system for creating 3-D animated dynamic graph structures (see Figure 4.23). It is implemented in Java and makes use of the Java3D graphics library. The system in its current implementation is not used specifically to visualise documents, but the force-directed layout model it employs and a simple CORBA API it provides can be used to visualise any kind of data. The user can create graphs by adding nodes, attaching labels to them, and connecting the nodes with directed or non-directed edges. It is possible to group nodes into clusters which are visualised as transparent spheres surrounding the nodes. Global and cluster-oriented force-directed layout algorithms arrange the elements of the graph. In the employed force model, nodes push each other apart by default and edges are springs between nodes. A field orientation is used for directed edges. To prevent unconnected nodes from floating away, a long-range attraction force is employed. Additionally a planar force can be used to squash graphs onto z=0. The user can rotate, pan and zoom the visualised structure and adjust every force component.

4.7.4 Lighthouse

Lighthouse [LEA00] is a system for browsing search results offering visualization and clustering of documents in addition to the ranked list (see Figure 4.25). The search query can be sent to a number of popular search engines such as Google, Alta Vista, or Excite and the specified number of matches will be processed. The system can check for broken links by trying to access the results, so that unavailable pages can be filtered out. The resulting documents are displayed as spheres floating in 2-D or 3-D space, where their proximity is a measure for their similarity. A sphere representing the query can also be added to the result set. Document titles can be displayed in three ways: as a linear list sorted by estimated relevance, as labels attached to the corresponding spheres (see Figure 4.24), or grouped in clusters (see Figure 4.25). Documents having a higher estimated relevance are coloured with a green shade, while those with a smaller estimated relevance have a red shade. The clustering algorithm can build a specified number of clusters, or, in the second operation mode, a dissimilarity threshold can be used to place documents with the dissimilarity value below this threshold in the same cluster. Adaptive clustering is also available allowing the user to mark documents as relevant or non-relevant, which forces the system to recluster the set and place differently marked documents into different clusters.

4.7.5 SQWID

The SQWID (Search Query Weighted Information Display) system [CKO97], provides an interactive visualisation of the search results, allowing users to see the relevance of documents to different key terms (see Figure 4.26). It is implemented in Java. SQWID takes the results of a query, identifies key terms related to the result set and creates a node-link graph of search results containing two different types of nodes: term nodes and page nodes, where page nodes can represent a single document or a site. The visualisation contains three differently coloured term nodes and one page node for every

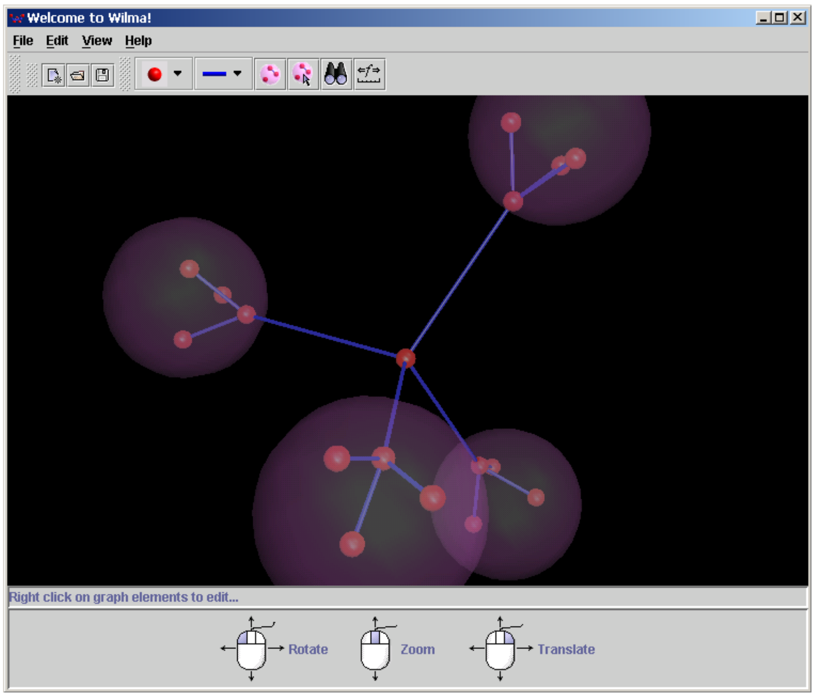
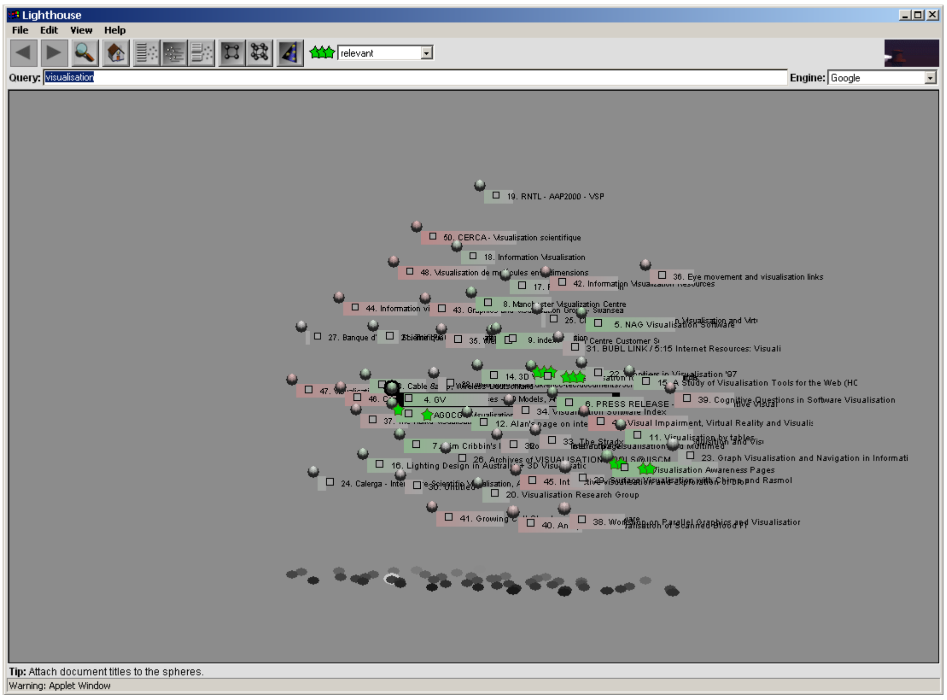


Figure 4.23: Wilmascope showing 17 nodes in 4 clusters.

search result. The three term nodes are always displayed in fixed places forming a triangle. Page nodes that are highly related to a term are pulled closer to it, while pages that are not closely rated to the a term node are pushed away from it. The position of a page node depends on the similarity to those three terms that are currently displayed as term nodes. Page nodes that take a position in the middle of the triangle are attracted to all three term nodes, nodes on the edges are attracted to two, and nodes that are pushed outside of the triangle are attracted to only one. If a page does not match any of the three terms, its node is pushed away from the all term nodes and it floats away to the edge of the view area. To avoid mutual covering of similar page nodes they repel each other by default. In this way they tend to spread out and the covering of each other is reduced (see Figure 4.27). Inside of each page node there is a coloured Tilebars representation, where the colours correspond to the colours of the term nodes currently displayed, and the colour intensities correspond to the rating for that page. Page nodes themselves can be combined into site nodes to prevent too many nodes from being displayed at once. The user can collapse the page nodes into site nodes to obtain a clearer overview, and can select term nodes from a number of identified key terms by simply choosing them from the menu upon which the systems recalculates the page node positions. It is also possible to adjust the number and date range of pages shown to filter out unwanted documents, explode site nodes back into page nodes (see Figure 4.28), view the links between sites and pages, as well as visit sites and view pages in a browser by double clicking its page node. On double-clicking a site node a site summary is automatically produced.

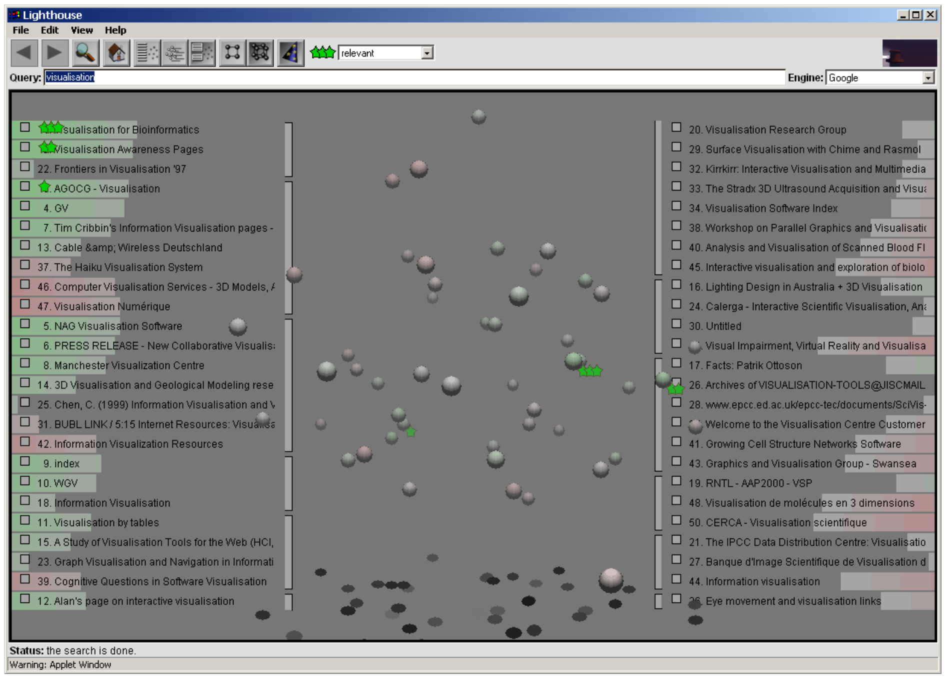
The operation of SQWID is divided in two phases. In the first phase, the processing phase, it collects data from the AltaVista search engine as a result of a standard search query. The system extracts relevant terms from the returned results set and calculates ratings for all pages and sites according to their similarities to the extracted terms. In the second phase the node layout is computed using a simulated annealing algorithm (see Section 4.1.4), where the nodes reach a state of minimum energy and come to rest in fixed positions.



Tip: Attach document titles to the spheres.

Warning: Applet Window

Figure 4.24: Lighthouse applet showing 50 search results with labels attached to spheres. The query was “visualisation”.



- ☐ sualisation for Bioinformatics
- ☐ Visualisation Awareness Pages
- ☐ 22. Frontiers in Visualisation '97
- ☐ . AGOCG - Visualisation
- ☐ 4. GV
- ☐ 7. Tim Cribbin's Information Visualisation pages -
- ☐ 13. Cable &amp; Wireless Deutschland
- ☐ 37. The Haiku Visualisation System
- ☐ 46. Computer Visualisation Services - 3D Models, /
- ☐ 47. Visualisation Numérique
- ☐ 5. NAG Visualisation Software
- ☐ 6. PRESS RELEASE - New Collaborative Visualis:
- ☐ 8. Manchester Visualization Centre
- ☐ 14. 3D Visualisation and Geological Modeling rese
- ☐ 25. Chen, C. (1999) Information Visualisation and \
- ☐ 31. BUBL LINK / 5:15 Internet Resources: Visua
- ☐ 42. Information Visualization Resources
- ☐ 9. index
- ☐ 10. WGV
- ☐ 18. Information Visualisation
- ☐ 11. Visualisation by tables
- ☐ 15. A Study of Visualisation Tools for the Web (HCI,
- ☐ 23. Graph Visualisation and Navigation in Informati
- ☐ 39. Cognitive Questions in Software Visualisation
- ☐ 12. Alan's page on interactive visualisation

- ☐ 20. Visualisation Research Group
- ☐ 29. Surface Visualisation with Chime and Rasmol
- ☐ 32. Kirrkirr: Interactive Visualisation and Multimedia
- ☐ 33. The Stradx 3D Ultrasound Acquisition and Visu
- ☐ 34. Visualisation Software Index
- ☐ 38. Workshop on Parallel Graphics and Visualisati
- ☐ 40. Analysis and Visualisation of Scanned Blood Fl
- ☐ 45. Interactive visualisation and exploration of biolo
- ☐ 16. Lighting Design in Australia + 3D Visualisation
- ☐ 24. Calerga - Interactive Scientific Visualisation, An
- ☐ 30. Untitled
- ☐ Visual Impairment, Virtual Reality and Visualisa
- ☐ 17. Facts: Patrik Ottoson
- ☐ 26. Archives of VISUALISATION-TOOLS@JISCMAIL
- ☐ 28. www.epcc.ed.ac.uk/epcc-tec/documents/SciVis-
- ☐ Welcome to the Visualisation Centre Customer
- ☐ 41. Growing Cell Structure Networks Software
- ☐ 43. Graphics and Visualisation Group - Swansea
- ☐ 19. RNTL - AAP2000 - VSP
- ☐ 48. Visualisation de molécules en 3 dimensions
- ☐ 50. CERCA - Visualisation scientifique
- ☐ 21. The IPCC Data Distribution Centre: Visualisatio
- ☐ 27. Banque d'Image Scientifique de Visualisation d
- ☐ 44. Information visualisation
- ☐ 36. Eye movement and visualisation links

Status: the search is done.

Warning: Applet Window

Figure 4.25: Lighthouse applet showing 50 search results grouped in 11 clusters. The query was “visualisation”.

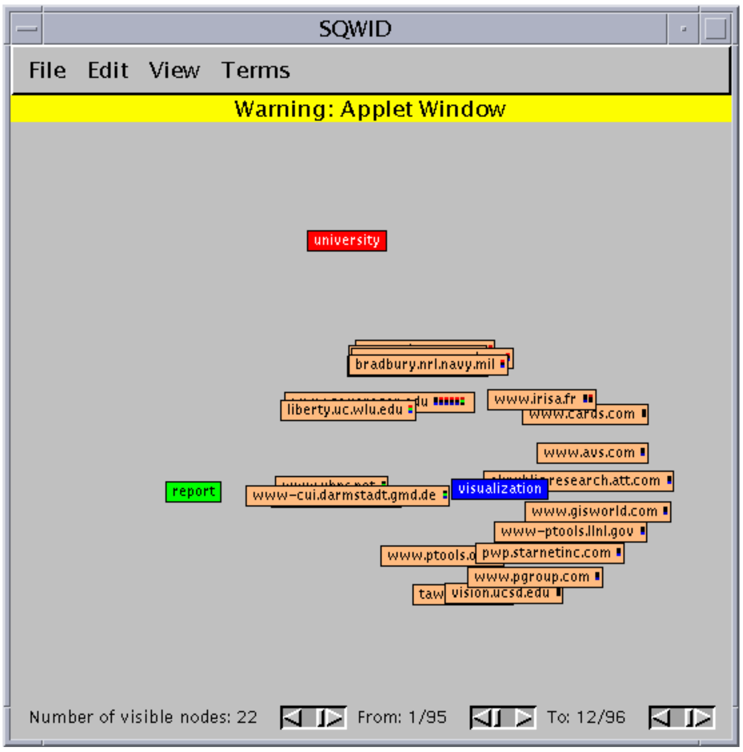


Figure 4.26: SQWID search results visualisation. The query was: "visualisation query results". SQWID chose the three terms for the term nodes. The result pages are grouped to site nodes. Taken from [CKO97].

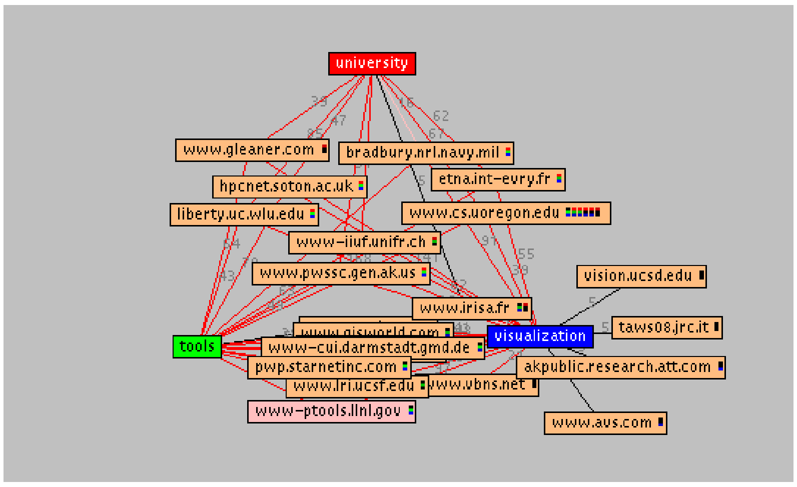


Figure 4.27: SQWID: By increasing the number of visible nodes the repulsive forces between them spread out the nodes, so they do not cover each other too much. The stress indicators are turned on. Taken from [CKO97].

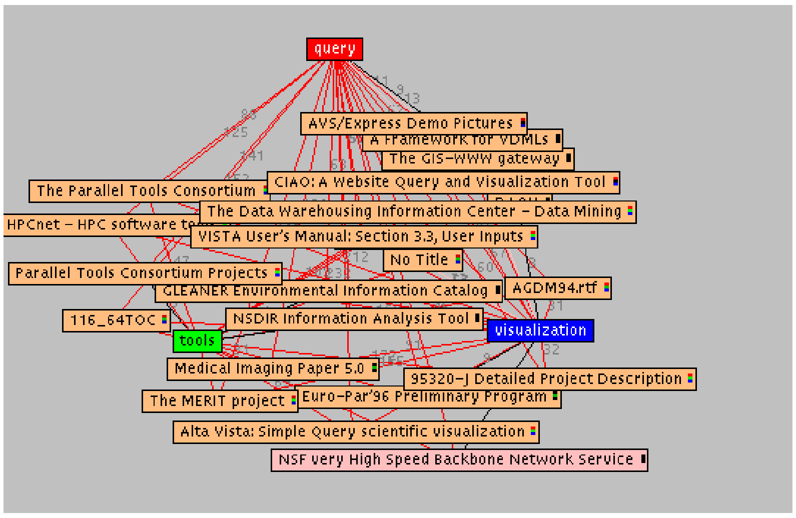

query
AVS/Express Demo Pictures
A Framework for VDMLs
The GIS-WWW gateway
The Parallel Tools Consortium
CIAO: A Website Query and Visualization Tool
The Data Warehousing Information Center - Data Mining
HPCnet - HPC software to
VISTA User's Manual: Section 3.3, User Inputs
No Title
Parallel Tools Consortium Projects
GLEANER Environmental Information Catalog
AGDM94.rtf
116_64TOC
NSDIR Information Analysis Tool
tools
visualization
Medical Imaging Paper 5.0
95320-J Detailed Project Description
The MERIT project
Euro-Par'96 Preliminary Program
Alta Vista: Simple Query scientific visualization
NSF very High Speed Backbone Network Service

Figure 4.28: SQWID: Site nodes are exploded into page nodes. Taken from [CKO97].

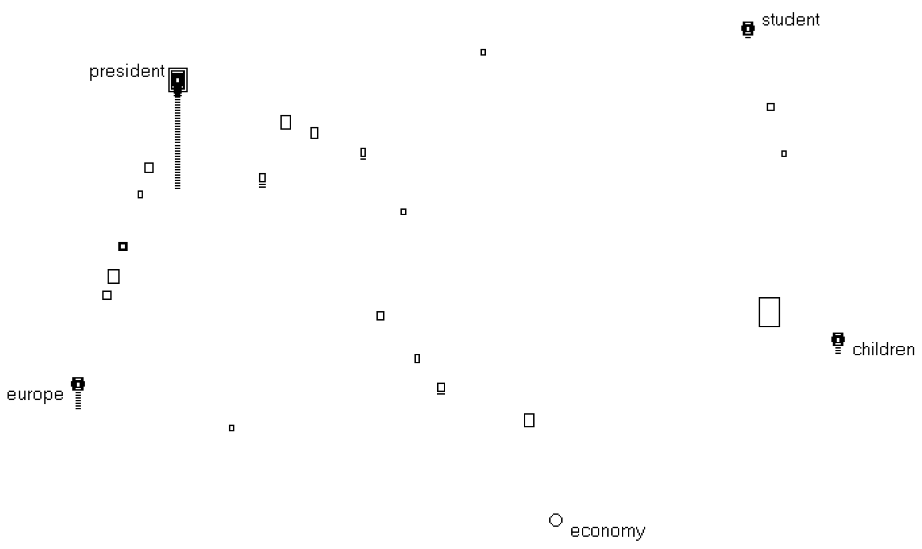


Figure 4.29: The VIBE system visualising the query “president europe student children economy”. Documents are represented by rectangles. Where the concentration of documents is high, clusters are represented by large rectangles. Taken from [OKS93].

4.7.6 VIBE and VR-VIBE

The VIBE (Visual Information Browsing Environment) system [OKS93] is a document collection 2-D visualisation system and one of the first with the capability to group objects together according to their semantic closeness. Points of Interest (POIs) corresponding to query keywords are placed in an abstract space. VIBE arranges documents within this space so that the position of a document represents its relationship to each of the POIs, creating a topical overview of the document collection. By assigning weights to POIs and moving them, the user can visually perceive the strenght of similarity of documents to the terms represented by the POIs.

VR-VIBE (Virtual Reality - VIBE) [BSG95] is a virtual reality version of the original VIBE system for 3-D visualisation of large document collections. The first step in using the system is to define a set of so-called Points of Interest (POIs). POIs contain keywords which are used to perform a query. A text search is performed on the document set and every document in the collection is given a relevance score for each POI. POIs are placed in 3-D space and the documents are attracted to them according to their relevance scores. VR-VIBE offers two layout methods. In the first one, which is similar to the original VIBE layout, the document nodes are placed on a 2-D plane depending on their relevance scores to the POIs. The third, vertical dimension, is used to represent the overall relevance of the object to the search query (see Figure 4.30). The second layout method allows the objects to be positioned freely in 3-D space, anywhere between the POIs (see Figure 4.31). The object's colour is used to represent the relevance to a POI, its dimensions represent a state of the object, for example selection, and its shape helps the user to differentiate between documents, POIs and other users. VR-VIBE supports multiple users, who can see each other in the same visualisation. They can navigate through the 3-D space and perform a number of different operations on the documents, perform queries, apply filtering according to certain criteria, add POIs and switch them on or off, move POIs to see which documents get pulled after them, request additional information, etc. Users may also define a number of different viewpoints and switch between them to compare different views. The following document actions can be performed:

- Selection: To obtain additional information one or more objects can be selected. This causes the icon to change colour and is visible to all users.
- Marking: The user can mark documents of special interest with a label, which is attached to the document.
- Viewing: This allows the user to view the document contents in a browser.
- Filtering: Using a relevance slider a relevance threshold can be defined. Only documents with a relevance above this threshold will be displayed.

4.7.7 Vineta

Vineta [KRO95] is a system which displays both documents and search terms as objects in a 3-D navigation space. The documents are positioned in this space according to their similarities as well as to their relations to the search terms, so that semantically similar objects are placed closer together. Documents which have a higher overall relevance are positioned closer to the user's view point. There are other systems which use a similar approach for visualising documents in 3-D space, like VR-VIBE or BEAD. However, Vineta differs from these systems, because for mapping documents from n-dimensional document space to 3-D navigation space it uses techniques from multivariate analysis and numerical linear algebra. In this way Vineta can visualise much larger term sets or POIs than is possible with similar systems.

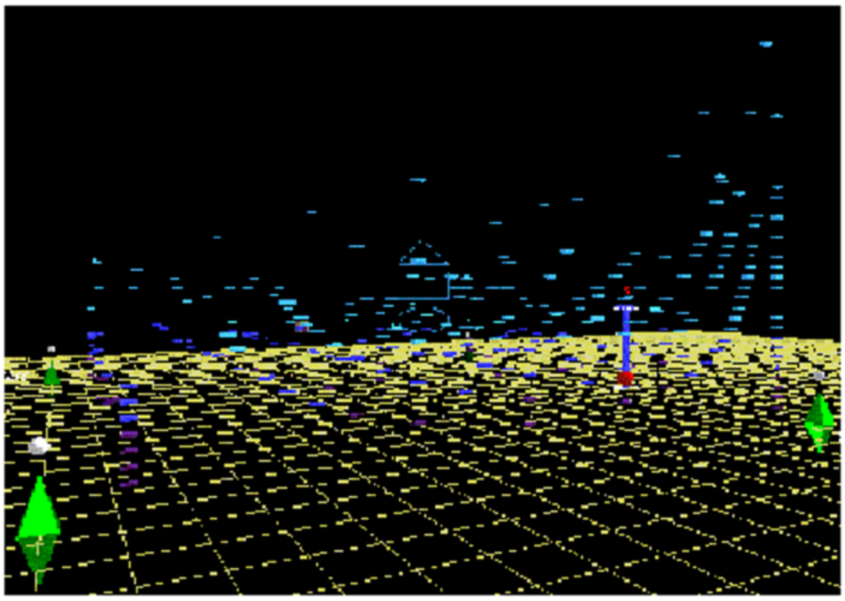


Figure 4.30: Screenshot of VR-VIBE: documents are placed in a 2D plane, height corresponds to relevance. Taken from [VRV01].

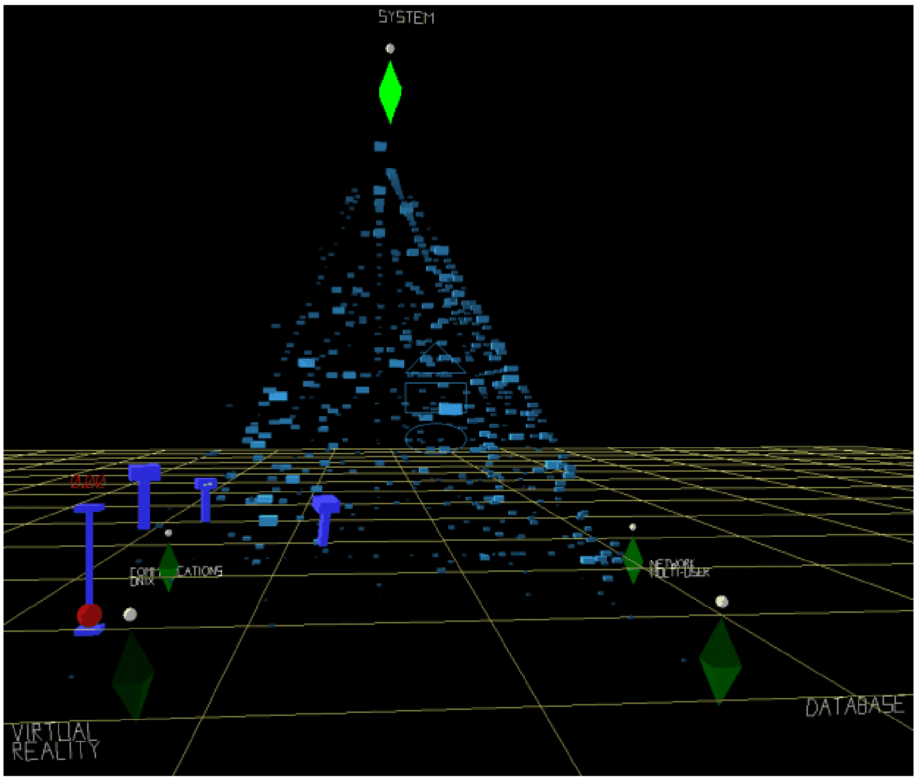


Figure 4.31: Screenshot of VR-VIBE containing 5 POIs and a number of users, full 3D layout. Taken from [VRV01].

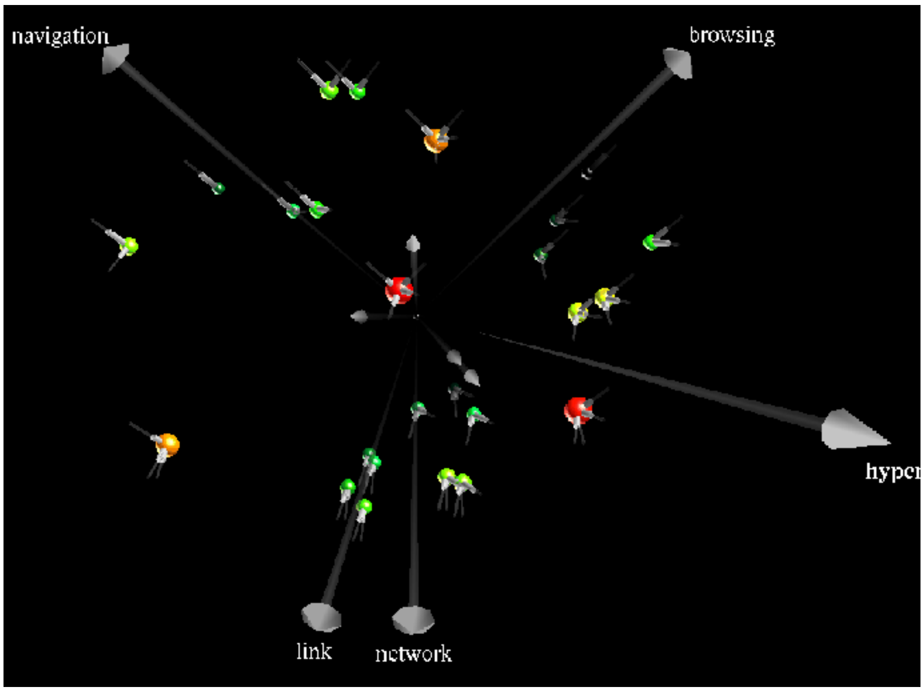


Figure 4.32: Screenshot of Vineta showing the galaxy visualisation. Taken from [YOU96].

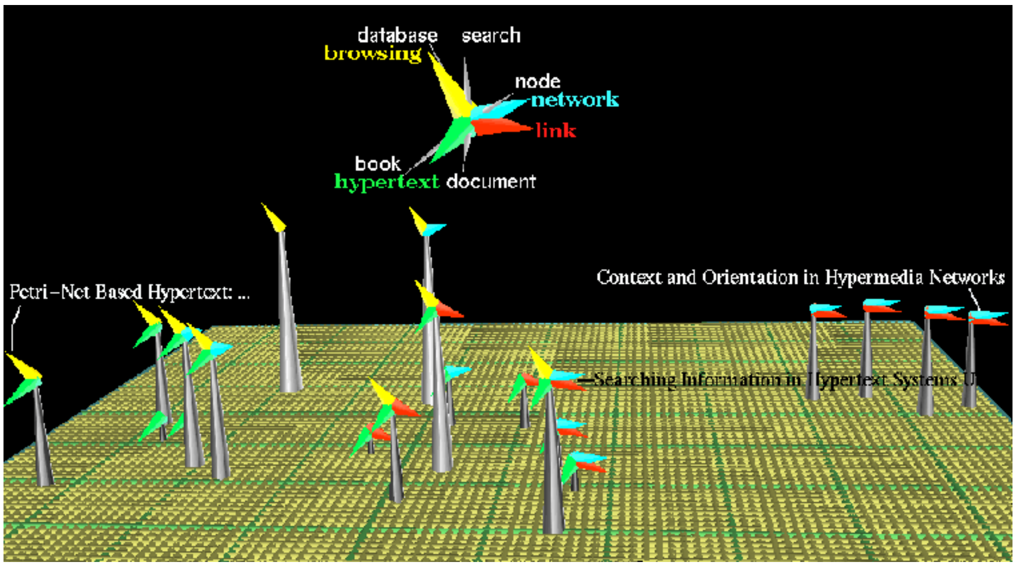

database
search
browsing
node
network
link
book
hypertext
document
Petri-Net Based Hypertext: ...
Context and Orientation in Hypermedia Networks
Searching Information in Hypertext Systems U

Figure 4.33: Screenshot of Vineta showing the landscape visualisation. Taken from [YOU96].

During early development Vineta implemented two different visualisations, the galaxy (see Figure 4.32) and the landscape (see Figure 4.33) visualisation. In the galaxy visualisation documents and terms were freely placed within the 3-D navigation space. Terms are represented as arrows (cones), while documents are displayed as spheres. As with other similar systems that make use of full 3-D geometry for visualising documents sets, this proved to be too complex for the user in terms of orientation and navigating. Therefore the landscape visualisation was chosen for the final implementation of Vineta system. The users found it to be more intuitive and much easier to understand and use. In the landscape visualisation the documents are visualised as flowers, whose stems help the user to exactly determine the 2-D positions of objects as they are touching the ground plane. Determining the overall relevance of the document is therefore much easier than in the galaxy visualisation. Petals represent the search terms and their relevance to each document. In the upper part of the screen there is a legend showing which colours and directions are assigned to different terms. This is the key for understanding the direction and colour of petals on the flowers. A profile of each document can be recognised by determining the colour profile of the flower representing the document.

4.7.8 BEAD

BEAD [CHC92] is a multi-participant document visualisation system which uses the force-directed placement technique to position documents inside a 3-D space (see Figure 4.34). It produces a so-called information terrain where similar documents attracting each other are grouped closer together, and those not similar repel each other, using a linear force-directed placement algorithm (see Section 4.1.4). Documents weakly or not at all related to most other documents are pushed away to the edges of the viewing area. The underlying algorithm relies on document similarities calculated for every two documents in the set from the word co-occurrence in both documents. In the first versions of BEAD, documents could exist freely anywhere in the virtual 3D space, but the complexity of obtained structures was to high for users to orientate themselves. To overcome this problem and make navigation easier for the users, additional forces were used to pull the documents closer to a 2-D plane. Connecting the documents and forming polygons produces the information terrain with its peaks and valleys which help the user in orientating and navigating the terrain. It was discovered that this 2.1-D landscape metaphor decreases the cognitive load on the user as compared to a full 3-D representation. Different users are represented as T-shaped figures constructed of two cuboids, with a further two cuboids representing the eyes. Different users can see each other and can interact with the data. They can select an object to perform operations upon it and can perform searches. Documents matching the query are highlighted in another colour. It is also possible that documents close to it are interesting to the user because they are similar to the highlighted ones, although they did not match the query directly and were not highlighted themselves.

In BEAD the documents are represented by particles with a mass and each document has a vector of n terms assigned to it. Each term in the vector represents a certain property and the value of the term indicates the amount of the property for the considered document. In the case of text documents these properties are words contained within the document, but also some other properties, like size or connectivity, could be taken into consideration. The vector representation makes is easy to calculate the similarity measures, like cosine correlation, between any two documents in the set. These similarity measures represent the relations between documents in this model. In BEAD a spring force model is used to describe the forces between particles in the 3-D space. The strength and the sign of these forces depend on the similarities between documents and on their distances in the 3-D space. Similar documents attract each other, while documents that are not similar but are lying close to each other are pushed apart. This mechanical model reaches a stable local minimum after a certain number of iterations, which is a state of a minimum energy for the system, where related documents are placed close to each other forming clusters of related documents. The structures and patterns formed

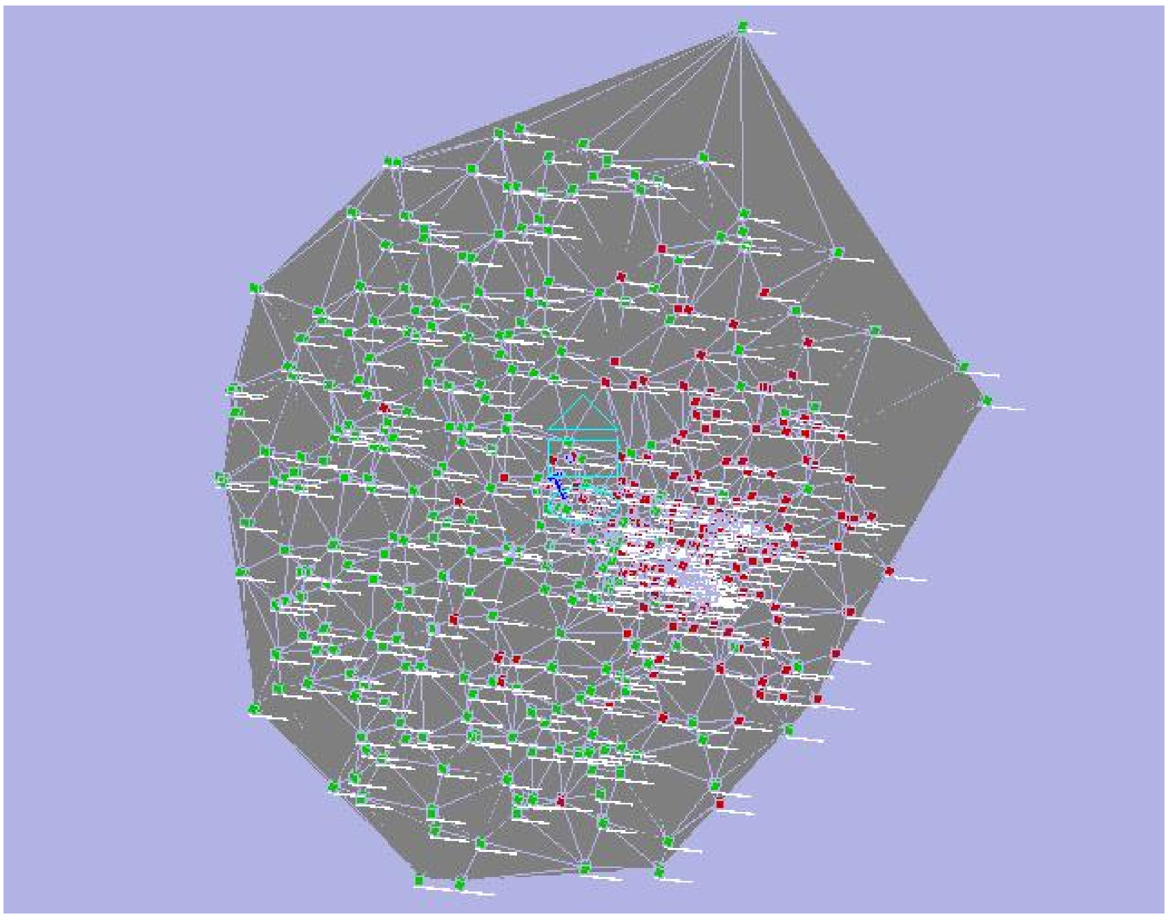


Figure 4.34: Screenshot of BEAD showing a data landscape constructed from over 500 bibliographic references. Taken from [YOU96].

in the layout can be easily identified by the user. However, the spring force model is a part of the well-known N-body problem, that requires, for each of the N particles, the calculation of the force to the remaining N-1 particles. Since this would have quadratic complexity, an approximate solution of linear complexity has been developed. In this approach each particle maintains two lists, each with a fixed number of elements: a list of neighbours and a list of random elements. In each iteration the new position for a document is calculated by considering only those forces associated with the documents in the two lists, instead of considering all N-1 documents in the set. A new random list is generated for every iteration from all documents in the set that are not members of the neighbour list. The neighbour list contains related documents, that means documents that are placed close to the considered document in the high-dimensional term space. This list is improved over time by checking documents that are fed into the random list, and putting them in the neighbour list instead if the random element is found to be better than the worst neighbour element. Good values for the sizes of the lists are found to be 5 for the neighbour set and 10 for the random set of documents. Increasing these values did not bring any significant increase in the layout quality and could not justify the increased calculation time.

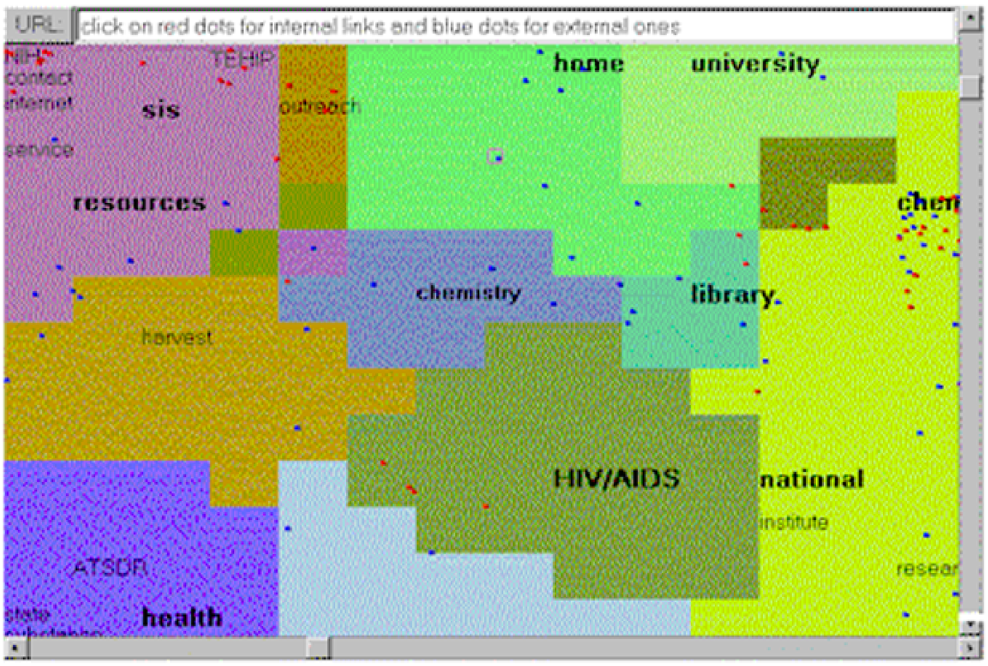
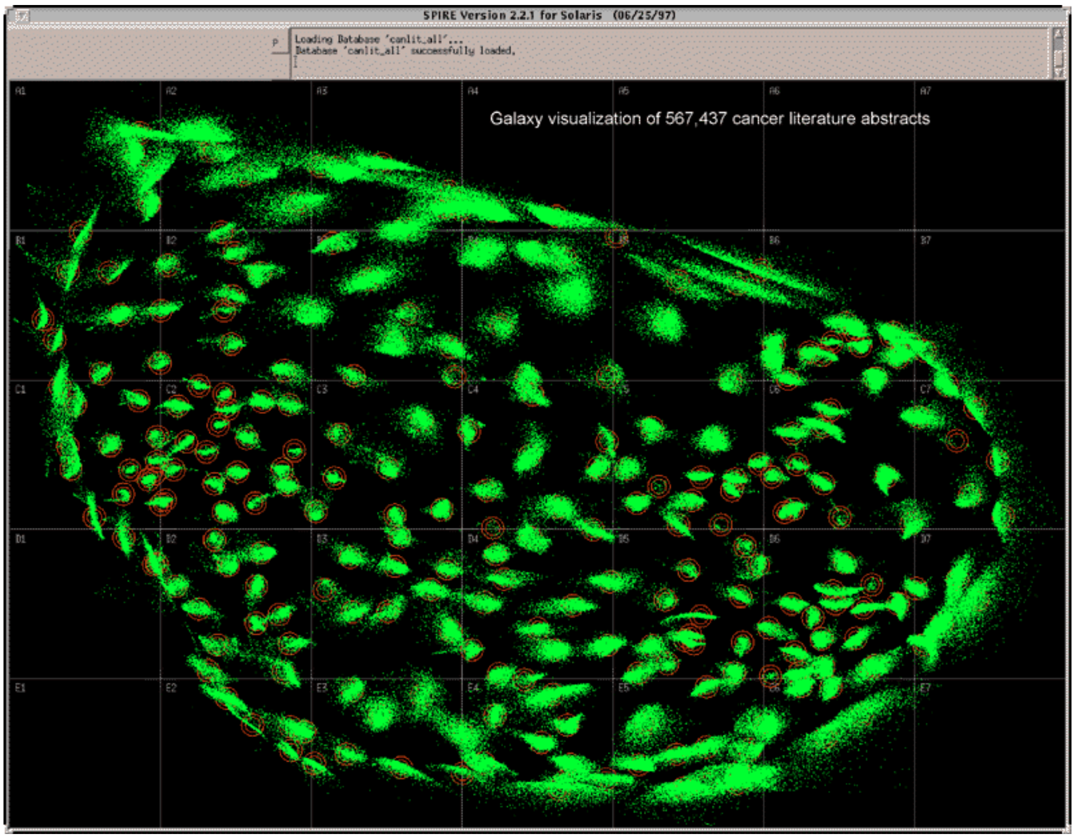


Figure 4.35: Sitemap: A Self-Organising Map. Taken from [VSM01].

4.7.9 Sitemap: A Self-Organizing Map (SOM) Based System

Visual Sitemap [VSM01] is an implementation based on Kohonen's Self Organizing Map (see Section 3.4.2). It is an application for visualisation of a web site and is made up of three main components. The first component is a WebCrawler which indexes the site, the second component is a neural network which organises the data and computes the map, and the third component is an interactive Java applet which presents the computed map of the site to the user (see Figure 4.35). The WebCrawler follows every link and analyses every document on the given web site, collecting data and building an index of all the words and pages of the site. Every document is internally represented by a vector which is then used to train a neural network. The output of this processing is an organised map, which visually represents the structure of the analysed site. In this way the documents are clustered and located inside of their subject areas. The neural network is able to automatically identify subject areas and categories, and extracts key terms that describe them. The sizes of these areas, their locations and relative positions are defined by semantic relationships between the subjects and characteristics of the documents belonging to the subject areas. The maps are labeled with words that represent important or frequently used terms in the documents represented by the area. Label size indicates the importance of the area. Coloured dots represent links to web documents or sites. The user interface, written as a Java-applet, is fully interactive so the user can conduct searches on the element set and perform different operations like selecting, getting additional information or viewing a document in a browser. The vertical scroll bar is used to set the number of visible dots, in order of their overall relevance. The horizontal scroll bar is used to control how detailed the labeled subject area in the map should be. However, web crawling and indexing are slow and time consuming operations. The neural network training is also a very complex and computation-intensive process, so Sitemap can not be used to produce a map on the fly. Visual Sitemap was employed on web pages of some major companies like, for example, Boeing.


SPIRE Version 2.2.1 for Solaris (06/25/97)
Galaxy visualization of 567,437 cancer literature abstracts

```
Loading Database 'canlit_all'...
Database 'canlit_all' successfully loaded.
```

Figure 4.36: SPIRE: Galaxies visualisation. Taken from [SPI01].

4.7.10 SPIRE - Spatial Paradigm for Information Retrieval and Exploration

The SPIRE system [SPI01] is a set of tools for conducting queries and exploring the information. The project was funded by the Department of Energy and U.S. intelligence agencies with the goal to develop means of visualising text. However, it should be mentioned that a variety of other very interesting visualiation projects are currently underway at PNNL [IPN01]. Since a human can not read, classify, and analyse thousands of documents in a short period of time, an electronic aid is needed to reduce information processing load and to improve productivity. An automated procedure had to be found to represent the documents and their contents and to perform retrieval, categorisation, abstraction and comparison. The SPIRE system was designed to do just that, offering two visualisations intuitive to human users. The metaphor for the Galaxies visualisation are stars in the night sky representing documents. The ThemeScapes visualisation is a further development of the concept where document characteristics are represented as sedimentary layers which together create a landscape. SPIRE generates a visualisation in a five step process:

1. The system takes text documents in digitised form that are fed to it in a completely unstructured way. The documents are in their natural forms, what means that they are not pre-processed in any way, there is no use of keywords or a dictionary, no classification, no topics and themes are extracted and there are no structure of any kind.

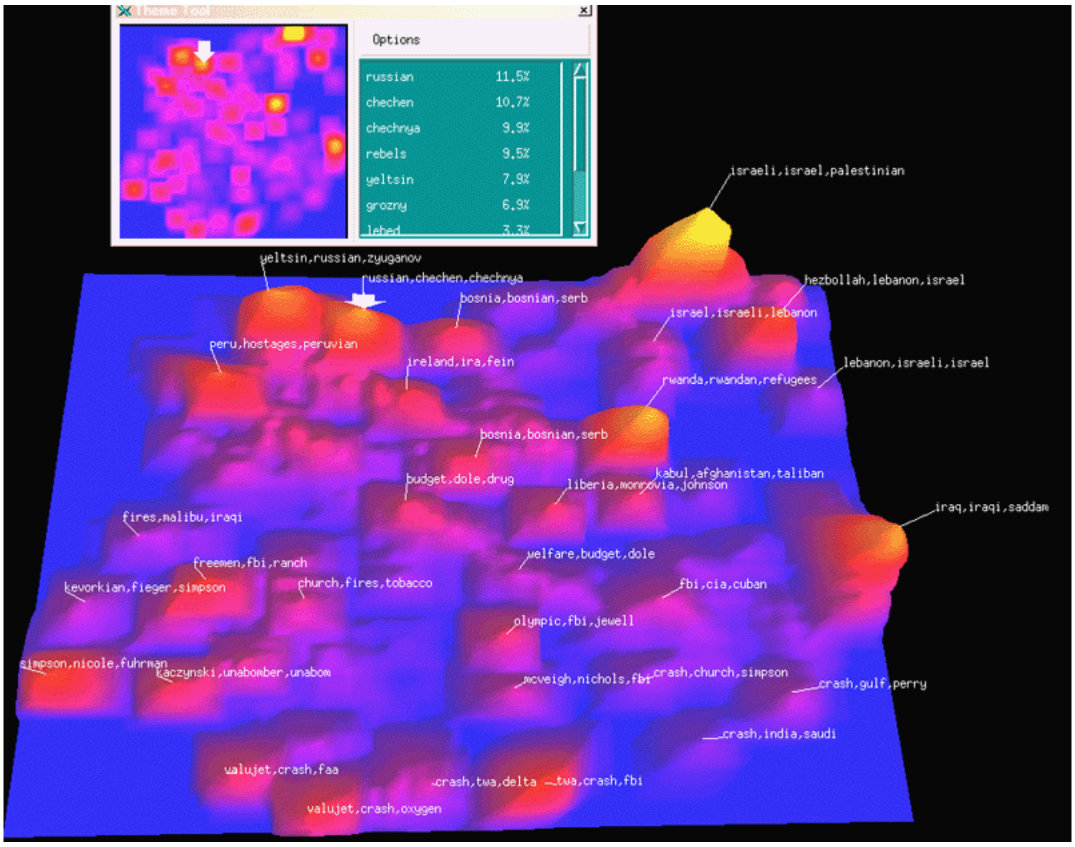
2. As SPIRE uses the vector space model (see Section 2.2) for internal representation of the digitised documents, a text engine analyses the documents to build high dimensional vectors for

each of them. In an early stage of development a commercially available text engine with a dictionary of 200,000 words was used. Depending on whether a word in the dictionary appeared in the document, a 1 or a 0 appeared at the appropriate coordinate of the vector. The problem with this approach was that a document vector was up to 200,000 units long, which was much too large for a high number of documents. To overcome this problem, in a later version of SPIRE, text analysis was performed by the Matchplus text engine created by HNC, Inc., which uses a neural net with 280 output nodes trained on a document set. This resulted in shorter vectors and opened the way to processing more documents. Finally, a special text engine optimised for visualisation systems was developed by the SPIRE team. In the first processing step stemming and stop word removal is performed. After that band pass frequency filtering is performed as too high and too low frequency terms do not generally help in discriminating among document content. In the following step, significant topical terms are extracted by measuring the randomness of the terms in the documents, the so-called Condensation Clustering Value (CCV), as it was found that good terms occur in serial bursts instead of being randomly distributed. The extracted topical terms are also supported by a list of terms that appear together with the topical terms, which characterise the context in which the topical terms are being used. Normalised vectors are constructed with topical terms as dimensions of the vector space.

3. Using the document vectors, the documents are clustered in the high-dimensional space producing groups of topically related documents where each document belongs to exactly one cluster. Standard k-means and complete linkage hierarchical clustering algorithms were taken into consideration and were found to be good for document sets containing a few thousands items. Driven by the important visualisation requirement that Euclidean distances of the documents in the plane should be proportional to distances in the high dimensional representation led to the development of a Fast Divisive Clustering algorithm. The number of clusters to be generated must be set by a person operating the system. Cluster seeds are randomly distributed in the high-dimensional space with some extra analysis ensuring that they did not land too close to each other. Documents are placed into clusters by determining into which of the non-overlapping hyperspheres defined around the seeds they belong to. New cluster centroids are calculated causing a shifting of the hyperspheres, so that some documents have to be assigned to new hyperspheres. This procedure is iteratively repeated until the shifting of the centroids falls under a certain threshold.

4. Projecting the document vectors and their cluster centroids from the high-dimesional space onto a 2-dimensional plane is the basis for creating both Galaxies and ThemeScapes visualisations. Multidimensional Scaling (MDS) was found to be good only for small document sets of arround 1500 elements as it showed an exponential increase in computational complexity. The development of the team's own projection algorithm, the Anchored Least Stress (ALS) algorithm was necessary. The idea behind it is that document placement should be based on its distance to the cluster centroids and not its distances to all other documents, such as is the case with MDS. The document is placed in the 2-D plane so that its position reflects its similarity to the clusters' centroids and not to other documents. Some information is lost with this approach, but considering that by projecting documents from a high dimensional space to a 2-D plane already brings a loss of information, and considering the improvement in computing time, this proved to be a good solution.

5. The last step is the construction of Galaxies and ThemeScapes visualisations based on the positions of the document in the 2-D plane. Galaxies represent the documents directly as stars on a dark background (see Figure 4.36). ThemeScape uses the document positions in Galaxies as a starting point for constructing a landscape (see Figure 4.37). It is built by layering computed

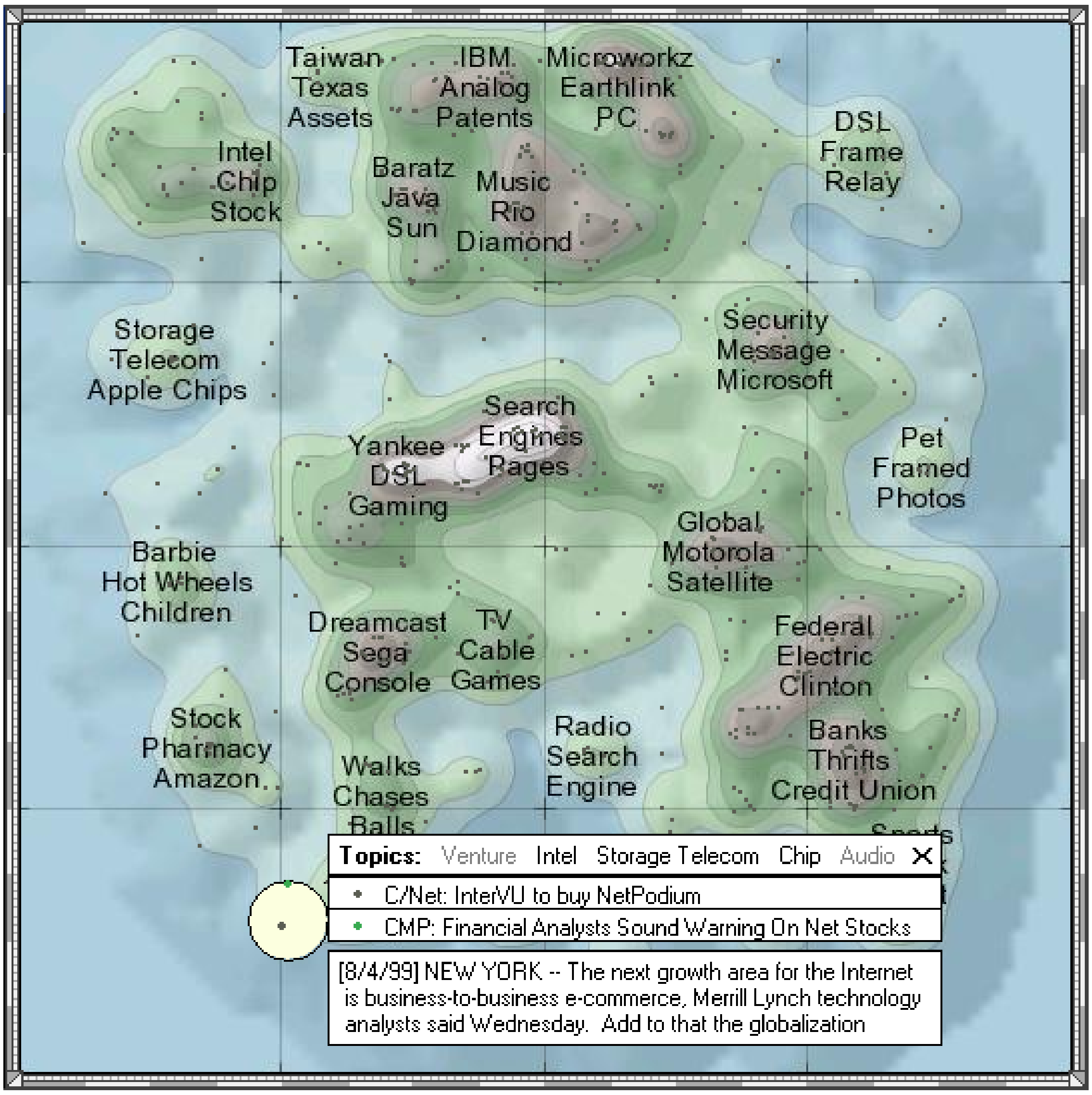


russian	11.5%
chechen	10.7%
chechnya	9.9%
rebels	9.5%
yeltsin	7.9%
grozny	6.9%
lebed	3.3%

Figure 4.37: SPIRE: Themescape visualisation. Taken from [SPI01].

contributions of theme terms over each other. Some 150-300 terms are used, chosen to discriminate clusters from each other as well as possible, which should ensure obtaining a sufficiently detailed landscape. These thematic terms with corresponding document coordinates are used to deposit each term's contribution as sediment layer of certain thickness to obtain the height of the landscape. The layers are summed together and normalised and a smoothing filter is passed over to produce a more natural landscape form. A term layer is thickest where the density of documents with that term is highest and if the clustering and projections of documents from high dimensional space to the 2-D plane are accurate enough to place documents containing with the same terms approximately in the same region of the 2-D plane, then, as term layers accumulate over each other the highest elevations occur where the thickest layers overlay each other, peaks generated in this manner symbolise groups of related documents. In lower regions the number of documents is smaller or the documents are less thematically focused, while a sharp cliff separates areas with strong thematic term content from those with low content.

Galaxies view imposes limitations on screen space for displaying large numbers of documents as each document is represented by at least one dot. With very large numbers of documents the distribution patterns become indistinct. The Themescape view avoids this limitation and is therefore ideal for an overall introduction to the document set. The user can select a desired part of the landscape, and then use the Galaxies view of documents containing topics in the selected part of the landscape. With a slider on each topical term, the user can select their relative weights which results in transformed visualisations providing an additional analysis tool.



C/Net: InterVU to buy NetPodium

CMP: Financial Analysts Sound Warning On Net Stocks

[8/4/99] NEW YORK -- The next growth area for the Internet is business-to-business e-commerce, Merrill Lynch technology analysts said Wednesday. Add to that the globalization

Figure 4.38: Cartia Themescape landscape visualisation, taken from [CAR01].

4.7.11 Cartia Themescape

Themescape [CAR01] is a commercial spin-off to the SPIRE visualisation system. It is a collection of four programs for automatically organizing and visualising large collections of documents. By pointing the software to a collection of documents like one or more web sites, directories, or a set of search query results, the system reads and analyses all the documents and creates an interactive map for exploring them. The software analyses the document contents, identifies important themes and topics, extracts words that describe them, creates abstracts, and visualises the results by creating a map with peaks, valleys and dots representing the documents in the set. Similar documents are placed close to each other on the map creating peaks on the landscape in areas where the concentration of documents with similar contents is large. These peaks are named after topics contained in the corresponding documents and are displayed as labels on the map giving names to certain areas so the user can easily locate the areas of interest. The distance between peaks corresponds to the similarity of the topics that they symbolise. Themescape maps are fully interactive: document title and a short summary is shown when the mouse pointer is placed over a document, the user can select documents or conduct searches, which causes the matching document to be highlighted.

The system is composed of four main components: ThemeServer, Web Manager, Theme Publisher and Map Viewer. The first three make the core of the system and manage the production, distribution and access to the maps. The Map Viewer is a client for viewing and browsing the maps, written in Java which makes it possible to view the maps from virtually any system.

4.7.12 WebMap

WebMap [WMP01] can automatically analyse a data repository and transform it into an interactive, visual map presenting very large amounts of data on a single screen. Data is organised into categories which are presented as distinctly shaped polygons. The size and shape of the polygon depends on the quantity and type of the information stored in the corresponding category. Proximity mapping is performed both for items and categories depending on a content relations metric so that the proximity of objects on the display is a measure for their relatedness. Documents have ratings assigned; documents with higher rating appear higher on the topography. The rating can be based on a parameter of choice. Items are visualised as stars or as icons where the former are used for highly rated items, search results, bookmarks and the similar. The visualised information is personalised dynamically so that icons can appear at the higher levels or sink to the lower levels of the hierarchy. The user can rearrange and customise the map by dragging and dropping items from one part of the map to another.

Animated zooming is WebMap's basis for browsing and retrieving information. Information is provided with a context allowing the user to explore the data and locate the needed information by zooming through the levels of the category hierarchy. Shapes of categories are preserved while zooming to improve orientation. This can be seen in Figure 4.39 where the category "Central Europe" has been selected and zoomed into to produce the representation displayed in Figure 4.40. The amount of map data that must be transferred from the server to the client application to display the visualisation is significant. For this purpose WebMap employs a proprietary communication protocol, load balancing mechanisms, and local caching of loaded data.

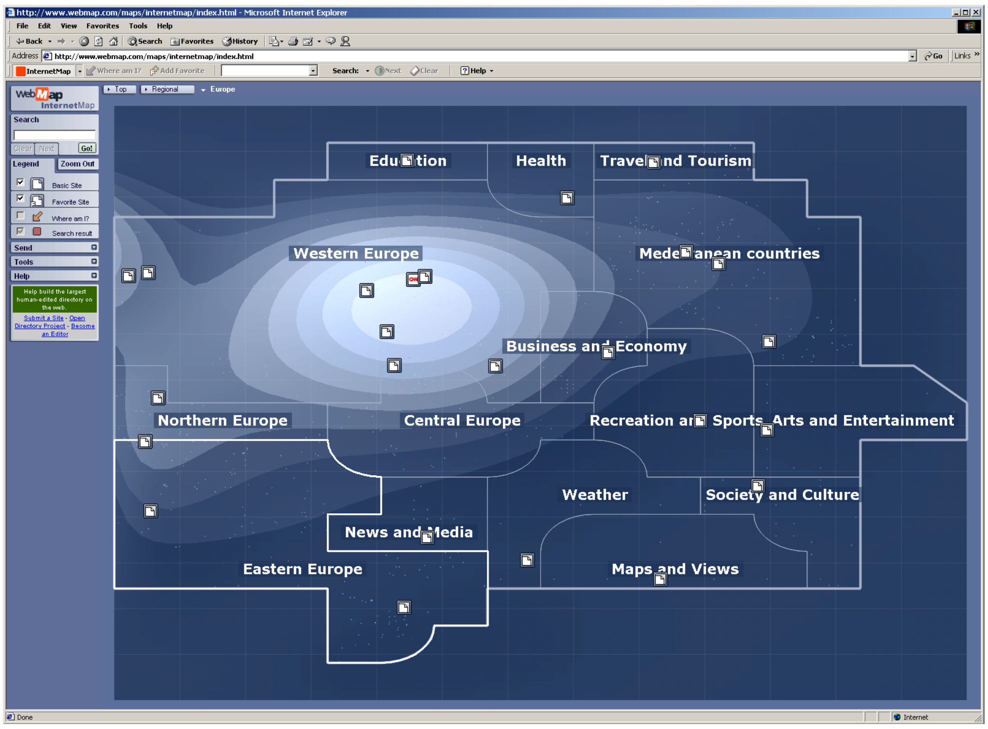


Figure 4.39: WebMap displaying the category “Europe”.

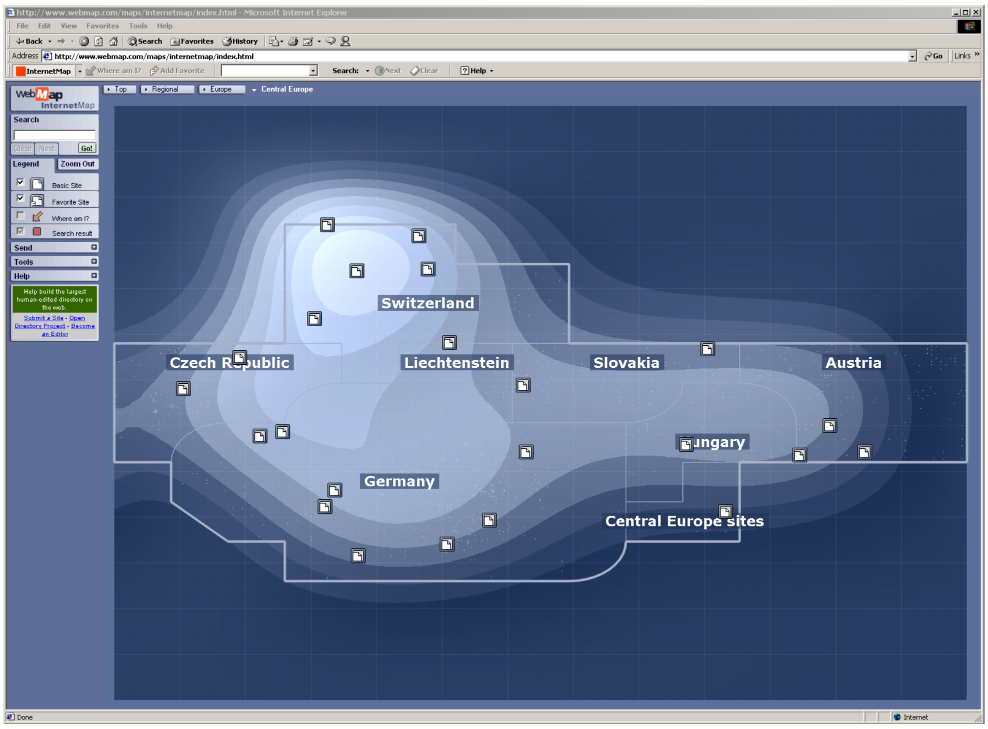


Figure 4.40: WebMap after zooming into the “Central Europe” category.

Chapter 5

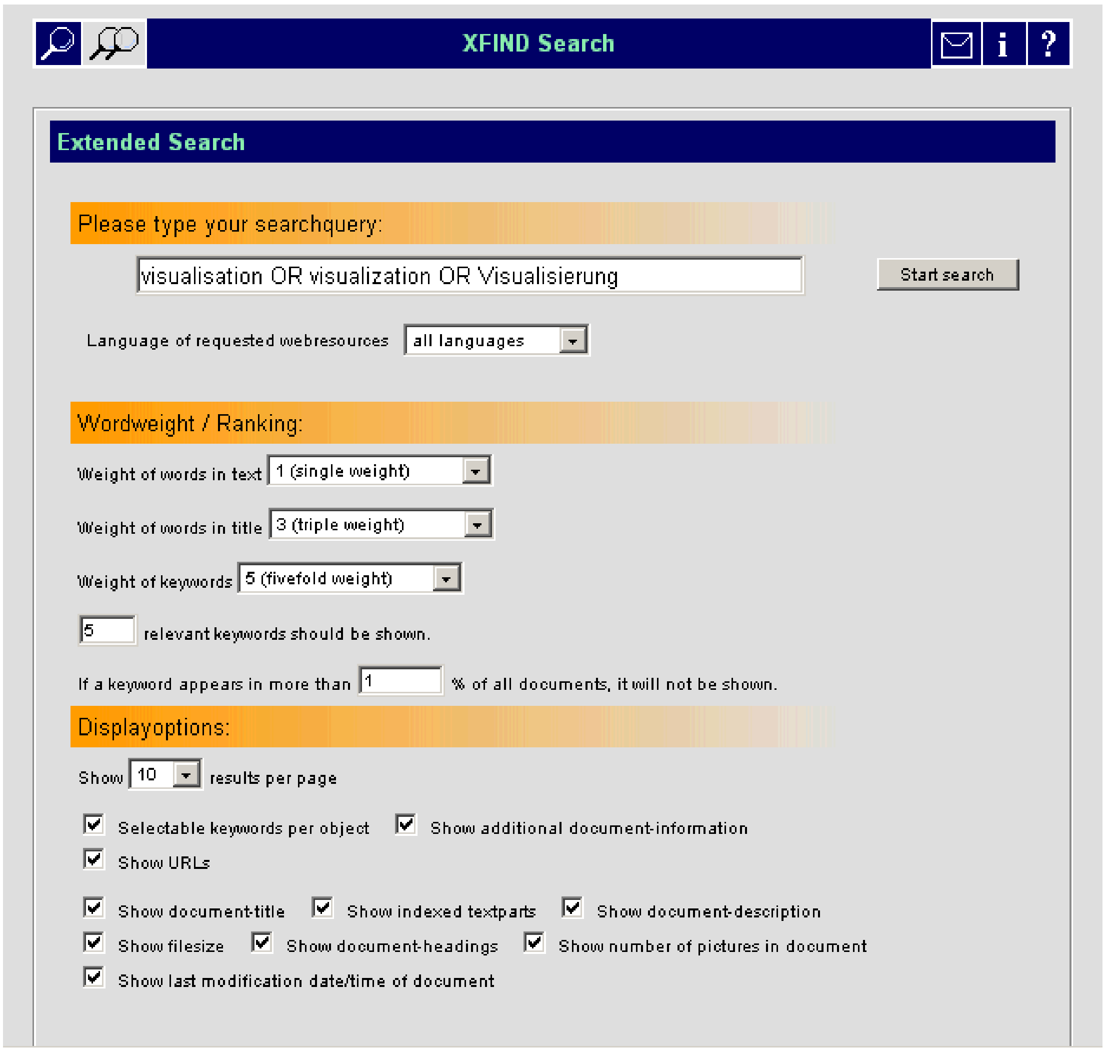
The xFIND Search Engine

5.1 Moving Beyond Harvest

The number of documents stored in the web is exploding. There are billions of accessible web pages in the Internet, and the effort, demanded of search service in finding and indexing them, and keeping indices up to date, is getting ever more difficult. The web has to be searched in regular intervals to meet the requirements, and while the number of pages is increasing so is the number of search services which are searching the web with their robots. Further, entire documents are loaded with all their raw data, although for processing only a fraction of the data is important. Centralised searching systems load documents from the web, analyse them and store the extracted data. Since each search service does this independently, it is obvious that the inefficiency of these systems can cause very high network and server loads. Such systems visit many pages that have not been altered too often, while other, rapidly changing pages, are not updated often enough.

The Harvest research project [HAS94], with its distributed hierarchical search index, goes around the problem of exploding network loads by separating the functionality for location and processing of data from the functionality for information retrieval and presentation. The part for locating and processing data is called the Gatherer. It collects and processes documents from several assigned information servers and examines them in regular intervals for modifications and new documents. Ideally the Gatherer itself runs on one of these servers, reducing the network load to a minimum. The data collected by a Gatherer are passed on, depending upon the configuration of the distributed searching system, to one or more Brokers. The Broker administers and stores the data from one or several Gatherers and it can also load information from other Brokers. It provides a user interface for performing queries to the user.

Distributed systems like Harvest try to address a number of other problems that centralised search engines can not cope with in a satisfying way, like providing many documents not containing relevant information, providing no quality ratings, not supporting the narrowing of the searches to specific domains, inability to deal with humans with different skills and interests etc. Such distributed systems are able, to a certain extent, to offer additional information depending on a situation and even give an advice, making them more a knowledge processing system than a standard search engine. Such problem-specific behaviour can be supported by other knowledge management systems and can be improved through experience by learning.



Extended Search

Please type your searchquery:

visualisation OR visualization OR Visualisierung

Language of requested webresources: all languages

Wordweight / Ranking:

Weight of words in text: 1 (single weight)

Weight of words in title: 3 (triple weight)

Weight of keywords: 5 (fivefold weight)

5 relevant keywords should be shown.

If a keyword appears in more than 1 % of all documents, it will not be shown.

Displayoptions:

Show 10 results per page

[✓] Selectable keywords per object [✓] Show additional document-information

[✓] Show URLs

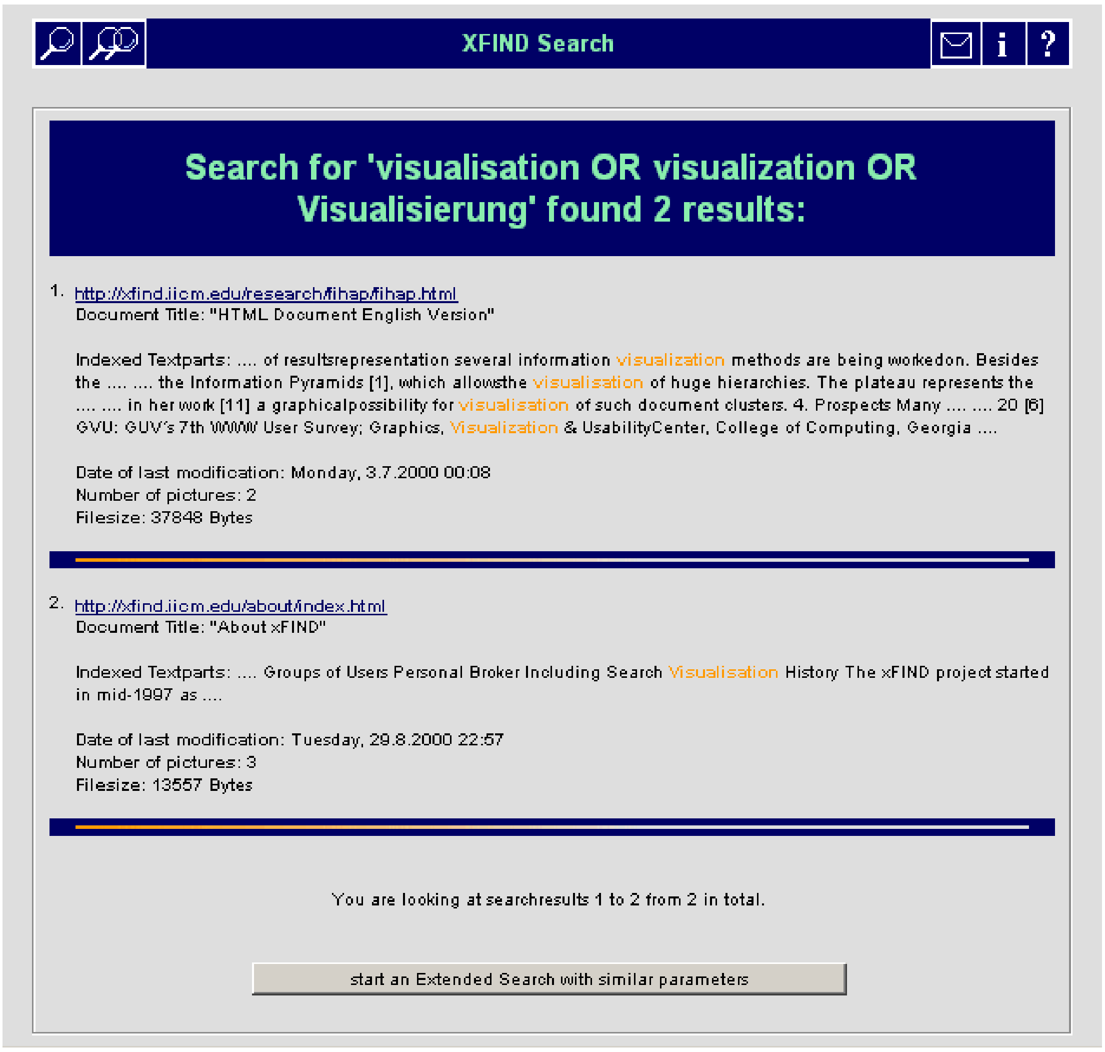
[✓] Show document-title [✓] Show indexed textparts [✓] Show document-description

[✓] Show filesize [✓] Show document-headings [✓] Show number of pictures in document

[✓] Show last modification date/time of document

Figure 5.1: xFIND extended search query input form.

The xFIND (eXtended Framework for INformation Discovery) [XFI01] system was designed as an open system with all these requirements in mind. It is a highly scalable distributed system, able to manage very large amounts of data and deal with highly dynamic content, reducing network and server loads considerably and improving search results using metadata sets, quality rating labels and web site descriptions. An xFIND system is made up of one or more Gatherers, Indexers, and Brokers. These components can be configured in different ways to match the user's interests and improve performance. For example, parts of the system can be configured to search only certain web locations, and sites with content that is updated or changed very often can inform the system and be re-indexed more often than others. The user has more influence on the search process than is the case with most of the standard search engines. It is possible to specify different weights for the occurrence of search terms, as well as which attributes, such as number of links or images, of matching documents should be returned. Statistical information, collected during indexing, can be presented to the user, protected documents, which are found but the user has no access, are reported together with an appropriate message and document metadata. xQMS, a quality metadata scheme for classification of resources



Search for 'visualisation OR visualization OR Visualisierung' found 2 results:

1. http://xfind.iicm.edu/research/fihap/fihap.html
   Document Title: "HTML Document English Version"

   Indexed Textparts: .... of resultsrepresentation several information visualization methods are being workedon. Besides the .... the Information Pyramids [1], which allowsthe visualisation of huge hierarchies. The plateau represents the .... in her work [11] a graphicalpossibility for visualisation of such document clusters. 4. Prospects Many .... 20 [6] GVU: GUV's 7th WWW User Survey; Graphics, Visualization & UsabilityCenter, College of Computing, Georgia ....

   Date of last modification: Monday, 3.7.2000 00:08
   Number of pictures: 2
   Filesize: 37848 Bytes

2. http://xfind.iicm.edu/about/index.html
   Document Title: "About xFIND"

   Indexed Textparts: .... Groups of Users Personal Broker Including Search Visualisation History The xFIND project started in mid-1997 as ....

   Date of last modification: Tuesday, 29.8.2000 22:57
   Number of pictures: 3
   Filesize: 13557 Bytes

You are looking at searchresults 1 to 2 from 2 in total.

Figure 5.2: The standard xFIND search results page.

depending on their richness of information and fitness for use, was defined, and includes fields which can be used to ensure that the quality metadata is trustworthy. Such a distributed system should be able to run on any machine in any available environment, and is therefore implemented in Java. For detailed information of how the xFIND system works please consult [GUT00].

5.2 xFIND Architecture

The xFIND System is a framework for distributed information discovery and knowledge management. The architecture of the system is based on three components: the Gatherer, the Indexer and the Broker.

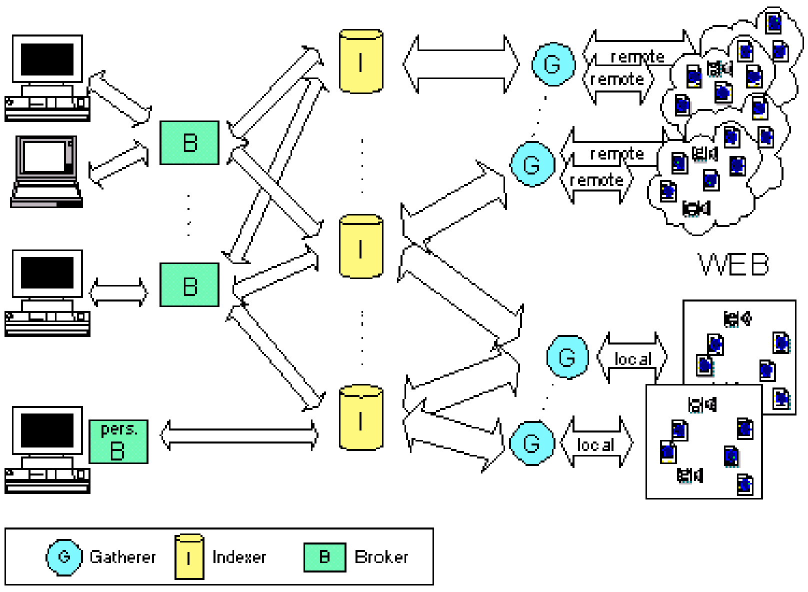


Figure 5.3: The xFIND architecture.

5.2.1 The xFIND Gatherer

The Gatherer visits servers and gathers information about documents and resources from local and remote sources. The pre-processing of the document data is also done at this point. This includes identifying the title, keywords, type, language, creation time, modification time, in case of HTML files the links, images that can be represented by thumbnails and other embedded objects like Java applets and metadata fields. An electronic fingerprint of each document is created, which makes it possible to detect the origin of every piece of information and to determine the trustworthiness of information in case of replication. Metadata embedded in the document as well as external metadata are processed. The system allows a wide range of configuration for pre-processing and handling of metadata sets, and supports the use and conversion between the following:

- Dublin Core Metadata (DC)
- Learning Object Metadata (LOM)
- xFIND Quality Metadata Scheme (xQMS)

The pre-processing and the reduction of information together with the usage of additional metadata improves the retrieval process, what is especially useful for learning environments. The enrichment of documents with metadata is made easy for authors as xFIND supports them to define metadata for a whole document structure, a directory or a particular document by inserting additional metadata files, where more specific metadata overrides more general. This help to improve the quality of information received by users greatly, but unfortunately authors of documents rarely use such possibilities to enrich their documents with such additional description. The xFIND Gatherer is able to process HTTP and plain text documents and information stored in local file systems and generates a set of metadata for each document. The document descriptors are passed to one or more Indexers, depending on the configuration of the system. The gathering process is configurable, so entire servers, particular sub-sites, or individual documents can be gathered at certain intervals. For best performance in terms of reduction of server and network load a local Gatherer should be used. It can also be configured to search for read-protected documents, in this case only a configurable predefined set of meta information is forwarded to the xFIND system, and the document remains protected. xFIND was designed to serve active information systems so an API and a communication layer had to be designed for this purpose. In this way active information systems with highly dynamic contents can contact the xFIND system and inform it about new or modified documents that should be processed. This improves greatly the actuality of the information and reduces the network and server loads as processing is done only when necessary. External systems for information enrichment (like expert rating systems, announcement systems, collaboration systems, etc.) may be processed similarly. Gatherers also process meta descriptions about information sources and summarise statistical data about information sources. The latter may be used for detecting highly dynamic information or may give a summary about activities of areas of interest.

5.2.2 The xFIND Indexer

Indexers are acting as a connection between Gatherers and Brokers. The document descriptors collected by gatherers can be fetched by or sent compressed to one or more authorised indexers. An Indexer may be specialised on a particular topic or can be dedicated to a project group, department or a geographical location. Indexer's job is to enable the Knowledge Broker to search for words, phrases and metadata and to provide statistical data about documents, like term frequencies. Indexers store information about web resources from documents to information sources, for example the number of changed or new documents for web areas. The indexer manages the communication with external systems (ranking systems, archiving systems, etc.) that, if trusted, can inform the xFIND system about new or modified documents, or send some additional information and metadata. Descriptions and external additional information improve the information structure for the learning process with respect to the information lifecycle. xFIND allows either the whole contents of a particular index or some parts of it, depending on topics or information sources, to be replicated from one Indexer to the others, if granted by the administrator of the Indexer. In this way network traffic network loads are reduced. Fingerprints for all documents and the relation to their origin help to detect and prevent changes of contents supporting a global information platform, which can be crucial when sharing information. Search capabilities of an external systems can also be used with the help of a xFIND API or a wrapper.

5.2.3 The xFIND Broker

The xFIND Broker is the component where user interactions take place. Following the distributed design concept of xFIND and because of the fact that Indexers manage only parts of the whole knowledge base, the Knowledge Brokers must distribute their search queries to a set of Indexers corresponding to the current search query. Using a thesaurus a broker can expand and transform queries, and the results of such a distributed search are put together to form a uniform search result set. Due to the open concept using the xFIND API external information sources can also be searched. For example, Hyperwave knowledge management systems [HYP01], a large-scale, multi-user, distributed, structured hypermedia information system [MAU96], developed at the Institute for Information Processing and Computer Supported New Media (IICM) at Graz University of Technology, can be searched directly and the results are merged with xFIND results. To improve search queries and the quality of returned information Knowledge Brokers can be specialised for a particular topic or can search only some problem specific Indexers or parts of them. Brokers are able to consider past search results and user ratings, as well as feedback from users that can influence the decision which Indexers should be queried. Thanks to the distributed architecture, Brokers can be configured individually to provide best results for a division or a department, for a single user or a group of users with similar interests, or to support particular topics. Similar to an agent system, the Personal Knowledge Broker is able to adapt to user habits and current problems and it will inform the user in case of new relevant information. When using an ordinary web browser, the xFIND broker provides standard search functions through HTML forms, like simple, extended, and expert search. Such a personalised Broker can provide additional information and find relevant keywords with respect to the users current interests. These keywords can be used to provide dynamically generated links, which start a new query when clicked. Relations to similar documents and links to expert or user opinions can be added to the original document. While in a standard centralised system, a high request rate can lead to a very high response time, such a distributed concept will not cause excessive network loads for any user. The search results are returned as a linear list ranked by relevance.

5.3 QCF - Query Communication Format

The internal communication format of the xFIND system is a text based QCF format. It is a query language standard for xFIND systems in which an application, the client, sends its request to the server and receives an answer from the it. The query sent to the xFIND system is an array of strings, with variable length which depends on the number of terms in the query, the number of query attributes specified by the user, and the number of attributes the user wants to receive. The array is made up of QCF keys and values, where the keys describe the values. The structure of the keys is hierarchical, different areas are separated by periods. The areas get more specific from left to right, so that the first word is specifies most global area. The query array is divided into three parts:

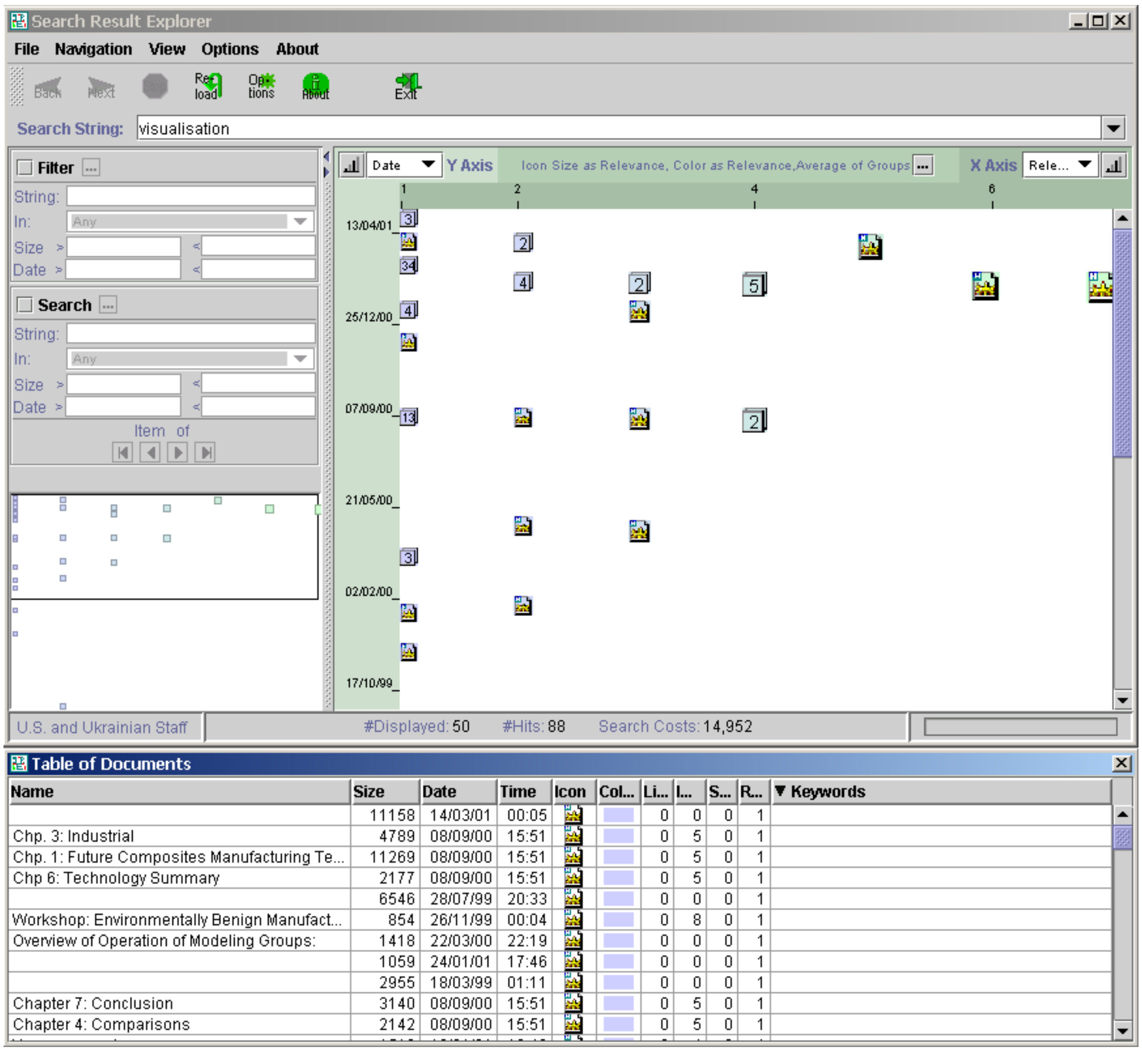
- Query: The first part contains the query, which is composed of search words and boolean operators combining the search words. Additionally xFIND offers the possibility to specify where the search words should occur enabling the user to influence the results. It is possible to search, selectively for each term of the query, for a string in the keywords, title and the body of the document assigning to each of these a different weight and influencing the estimated relevance of matching documents in this way. The ability to provide prefix, infix and postfix notation in the search terms with the use of wildcards such as *, ? is also offered.

- Query Attributes: The second part of the string array allows the user to specify additional settings that influence the search. To mention a few examples: the number of reported results and the number of keywords returned with each document, can be specified, it is possible to vary the weights of words that occur in different parts of the document, etc.

- Return Attributes: The third section specifies which attributes of the found documents should be returned giving the user full control of the result formats. The user can choose if keywords or parts of the body surrounding the keywords should be returned, which metadata should be returned, etc.

5.4 The Search Result Explorer (SRE)

The Search Result Explorer was developed to address one of the main problems with the currently available information discovery systems: they generally do not present explorable results. Standard search engines produce a list of matching documents, ranked by estimated relevance, with little or no additional information. The user has no possibility to compare the retrieved documents and can



Search String: visualisation

U.S. and Ukrainian Staff #Displayed: 50 #Hits: 88 Search Costs: 14,952

Name	Size	Date	Time	Icon	Col...	Li...	I...	S...	R...	▼ Keywords
	11158	14/03/01	00:05			0	0	0	1	
Chp. 3: Industrial	4789	08/09/00	15:51			0	5	0	1	
Chp. 1: Future Composites Manufacturing Te...	11269	08/09/00	15:51			0	5	0	1	
Chp 6: Technology Summary	2177	08/09/00	15:51			0	5	0	1	
	6546	28/07/99	20:33			0	0	0	1	
Workshop: Environmentally Benign Manufact...	854	26/11/99	00:04			0	8	0	1	
Overview of Operation of Modeling Groups:	1418	22/03/00	22:19			0	0	0	1	
	1059	24/01/01	17:46			0	0	0	1	
	2955	18/03/99	01:11			0	0	0	1	
Chapter 7: Conclusion	3140	08/09/00	15:51			0	5	0	1	
Chapter 4: Comparisons	2142	08/09/00	15:51			0	5	0	1	

Figure 5.4: The Search Result Explorer.

hardly decide which documents are up to date and contain wanted information. SRE sends queries to the xFIND system and visualises search result sets using the rich metadata returned by the search engine. SRE uses metadata attributes such as document size, modification date, relevance, and number of links and maps them to the display axes to present retrieved documents by plotting them on a two-dimensional display, with further dimensions mapped to colour, size and shape of the displayed graphic primitives, providing an interactive and intuitive interface with features like flexibility in scaling, functions for details on demand, etc. SRE benefits from the capabilities of the xFIND search system and is a further development of the similar File Attribute Explorer (FAE), which gave an overview of the files in a file system's directory hierarchy. The variety of the returned attributes make an ideal basis for creating multidimensional visualisation of the returned data.

SRE is designed to improve the searching process by visually presenting all attributes, metadata, and statistical information of the returned documents, enabling the user to compare age, size, last modification time, estimated relevance, authors, and other meta-information, with the goal to enhance and speed up the search process. A visual presentation of the retrieved documents is created. As already mentioned, the user is allowed to define the semantics of the two axes assigning any attribute to any of the two axes, as well as mapping some of the attributes to size, colour and shape of the icons representing the documents. The axes can be mapped to the following values:

- Size: The size of the document.
- Date: Last modification time of the document.
- Name: Title of the document.
- Relevance: The estimated relevance of the document for the given query as computed by xFIND.
- Host: Specifies the server where the document is located.
- Number of links: Describes the “connectivity” of the document. It can be useful to find documents that are themselves tables of contents.
- Number of images: The number of all images in the document.
- Number of scripts: This value can be seen as a degree of interactivity of the document.

SRE supports and manages some additional descriptive information such as keywords, description and author, which are not be mapped to axes, but that can be displayed in the Table of Files and in the Document Detail Window, with both tables offering sorting possibilities for every displayed column. Furthermore, keywords, size and date can be used for searching and filtering the retrieved documents. Additional attributes supported by SRE are:

- Keywords: This string contains the keywords and their number of occurrences in the document. They are extracted from metadata and content by the xFIND search engine.
- Author: The person who created, and is responsible for the content of the document.
- Publisher: The entity responsible for making the resource available in its present form.
- Organisation: The organisation where the document was created.
- Description

Chapter 6

Visualisation Islands Software Architecture

Search engines usually return search query results in the form of a linear list of hits sorted by their estimated relevance to the query. It is hard to get an overview when the number of results is large and the relationships between different hits are not made explicit. Visualisation Islands software package organizes search query results depending on their content and visualises them using an intuitive topographic map. This chapter discusses design decisions, the class hierarchy and tasks of different classes, interfaces and packages, as well as data and control flow.

6.1 What is Visualisation Islands?

Visualisation Islands is an xFIND search engine client which processes documents returned in a response to a search query, organising them according to their similarity, and visualising the results in a form of an interactive, intuitive 3-D topographic map. Document data returned include the title, URL, the size and the creation date as well as a number of keywords (and their frequencies), describing the documents' content. Visualisation Islands uses these keywords to build a vector representation of the documents in a high dimensional space (see Section 2.2), allowing the definition of a similarity measure (see Section 2.2.1) for every two vectors and thus giving a mechanism for comparing returned documents. In the first processing step, clusters of similar documents are formed using a standard clustering algorithms such as hierarchical agglomerative clustering (see Section 3.3.1) or the k-means algorithm (see Section 3.2.2). The obtained structure is used to display the documents in a table of contents style. In the second processing step a force-directed placement algorithm (see Section 4.1.4) based on stochastic sampling [CHA96a] maps the documents from the high-dimensional vector space to a 2-D plane whilst trying to preserve their relative distances as far as possible. This is achieved by assigning a force between every pair of documents according to their similarity: an attractive force for similar and a repulsive force for dissimilar documents. After a certain number of iterations a 2-D layout is obtained where similar documents are grouped together forming clusters, while less similar documents are placed apart from each other. Instead of comparing every two documents in every iteration, stochastic sampling is used to improve performance. In the third processing step, a 3-D (or 2-D) topographic background is generated based on the computed 2-D document coordinates, forming mountains at areas where document density is high. Mountains are divided by areas with smaller document density which are represented as lower areas or water. As a result islands of topically similar documents are formed. By labeling the mountains with the keywords of the underlaying documents and displaying the documents' icons atop the generated background, an intuitive landscape (or seascape) is created enabling the user to instantaneously locate areas of interest and discard documents covering less interesting topics. The user interface provides several means of exploring the map including zooming, searching, filtering and selecting and viewing the details and contents of documents of interest. The system can handle and process multiple maps simultaneously by using several execution threads and allows the reprocessing of any subset of the returned documents for more detailed insight.

Visualisation Islands is written in Java (see next section) and therefore follows the object-oriented paradigm. Java was chosen for the reasons of inter-platform portability and because of the fact that the xFIND system is also implemented in Java. The program itself is modularly designed. It is a Java package consisting of three main parts, or sub-packages, with clearly defined interfaces, which makes it possible to easily expand or interchange any of the parts without the need to change the others. The parent package visislands provides the main program control. The three sub-packages are: visislands.searchenginecommunication which handles the communication with the xFIND search engine, sending the query and loading results; visislands.dataprocessing provides data structures and processing algorithms for organising the loaded documents and creating the visualisation; visislands.userinterface displays the computed results and interacts with the user.

6.2 Java

Java is an object-oriented language, developed by Sun Microsystems in the early 1990's, similar to C++, with certain restrictions such as elimination of multiple inheritance and pointer arithmetic, which do not affect the power of the language, but eliminate some common sources of errors. Another important feature is the garbage collector so that the programmer does not have to care about releasing memory: a chunk of memory which is not referenced any more by a pointer will be collected and released automatically. Java's simplicity and similarity to C and C++ make it possible for most programers to become familiar with it in a fast and straightforward manner. The language was designed for use in Internet and distributed environments and it is therefore secure and robust. Java executable code can move through the network and be executed anywhere, even remote calls are possible through the RMI (Remote Method Invocation). In addition to standalone applications, Java has another program form: an applet. An applet is executed in a browser and is dynamically loaded together with the rest of the page. The use of multi-threading (and multi-processing if multiple processor are available) is also well supported, but the most important feature of Java is its portability and platform independence. The language and the accompanying libraries are designed in such a way that a Java program will execute and perform in the same way on any hardware and under any operating system, provided a Java Virtual Machine (JVM) is available and installed. Today Java is widely supported and available for almost any system, making software development in Java a write once - run everywhere solution.

6.2.1 Platform Independence

Portability and platform-independence are probably the most important features of Java. A significant effort was needed to achieve this. First of all a number of standardised Application Programing Interfaces (API) had to be developed covering a large number of programing areas, for example user interface functionality, 2-D graphics, sound, security, naming, input and output, networking, remote method invocation, just to mention a few covered by the Java2 platform. Additional APIs are making their way into the platform or are available as an add-on. These APIs are, for example, Java 3-D, advanced imaging, media framework, Java speech API, and many others. The language, APIs, and the accompanying libraries are designed in such a way that a Java program will execute and perform in the

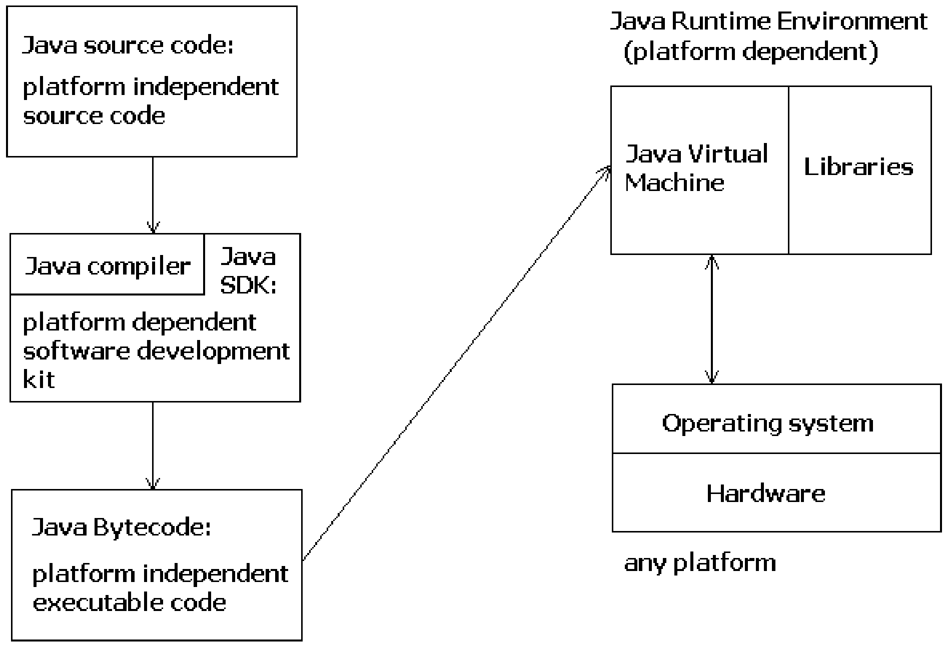

Java source code:
platform independent source code
Java compiler
Java SDK:
platform dependent software development kit
Java Bytecode:
platform independent executable code
Java Runtime Environment (platform dependent)
Java Virtual Machine
Libraries
Operating system
Hardware
any platform

Figure 6.1: Java: platform-independent software development and execution.

same way on any platform, independent of the hardware and the operating system. To understand how this works, take a look at Figure 6.1. After a Java program is written it is compiled and Java byte code is generated. This is a machine code which is not generally executed directly by a microprocessor (although such processors were also designed), but by the Java Virtual Machine (JVM). Therefore, to execute a Java program a Java Virtual Machine must be available for a particular system. As Java is a widely supported industry standard this is the case for almost any hardware platform and operating system (ever more systems are supporting Java2 as a standard feature). Early Virtual Machines simply interpreted the Java bytecode which brought a serious performance penalty and that is probably the most serious drawback of the Java concept. To overcome this problem just-in-time (JIT) compilers were developed. These compilers translate byte code into native code just before execution. This results in a noticeable increase in performance but has some disadvantages: the size of the code explodes, the time to start the program is longer and the compiled code is not optimised as this would bring a much greater delay. The latest generation of Java Virtual Machines (summer 2001), so-called Hot Spot VM, address these problems in a completely different way. The byte code is initially interpreted, but at the same time the Virtual Machine watches which part of the code is executed often. These parts of the code are dynamically converted from byte code to the native code and optimised. As a part of the code is executed more and more often it is more aggressively optimised. Much of the code is executed only once, and often most of the execution time is spent in small, localised pieces of the code, the hot spots. This approach improves performance further significantly, reducing long start delays and code size explosion. However, Java performance, although very much improved, is still far from the performance provided by a good C++ compiler. There are Java native code compilers too, but producing native code for improved performance sacrifices one of the Java's most important features, the platform-independence.

6.2.2 Java Foundation Classes

Java Foundation Classes (JFC) is a powerful set of graphical user interface class libraries, giving programers the possibility to develop modern, user friendly, full featured, platform independent user interfaces. At present, the JFC consists of six libraries each covering an important area of design and development of graphical user interfaces. These libraries are:

- The Abstract Window Toolkit (AWT) supports Graphical User Interface (GUI) programming featuring a set of user interface components, layout managers, an event handling model, graphics and imaging tools, shape, color, font and data transfer classes for cut-and-paste operations. AWT user interface components like buttons or menus rely on native platform components. For this reason only a relatively small set of components, that exists on all platforms, is supported. Being native components they do not look and work exactly the same on every platform.
- Swing is a set of GUI components implemented in 100% pure Java. Many Swing components inherit on existing AWT components, like buttons, menus or scrollbars. A large number of new components with a many new features, such as tree view, internal frames and tabbed panel, was added. Swing components do not need to exist as native components as they are implemented in pure Java. As a consequence, components with powerfull features which look and work the same on every system are available to the programmer. Also, additional event handlers and layout managers were introduced. Another interesting feature of Swing is the pluggable look-and-feel: a single set of GUI components can automatically change the look and feel to Microsoft Windows, Solaris, Macintosh, or Metal - the native Swing look-and-feel. Although Swing components are called lightweight components (AWT components are called heavy weight), these, being quite complex and being written in pure Java, need significantly more hardware resources than AWT components.
- The Accessibility API supports people with disabilities, letting Java programs interact with assistive technologies such as screen readers, speech recognition systems and refreshable braille displays.
- The Java 2D Graphics and Imaging API is made of classes added to java.awt and java.awt.image packages. They provide tools for managing line art, text, and images, a rich set of imaging operators, accurate colour space definition and conversion and support for image compositing and alpha channel images.
- Drag-and-Drop enables simple and user friendly data transfer between Java applications as well as between Java and native applications.
- Input Method Framework is a link between text editing components and input methods in entering text. Input methods are software components that let a user enter text in other ways then typing it on the keyboard, for example handwriting and speech recognition. These methods are also used to enter symbols such as Chinese which might have thousands of different symbols.

6.3 Classes and Interfaces

Visualisation Islands is organised as a single visislands package, together with three sub-packages: searchenginecommunication, dataprocessing and userinterface. The role of each package, class and interface is explained here, together with most important interactions between classes. A hierarchy of classes defined in Visualisation Islands is presented in the Figure 6.2.

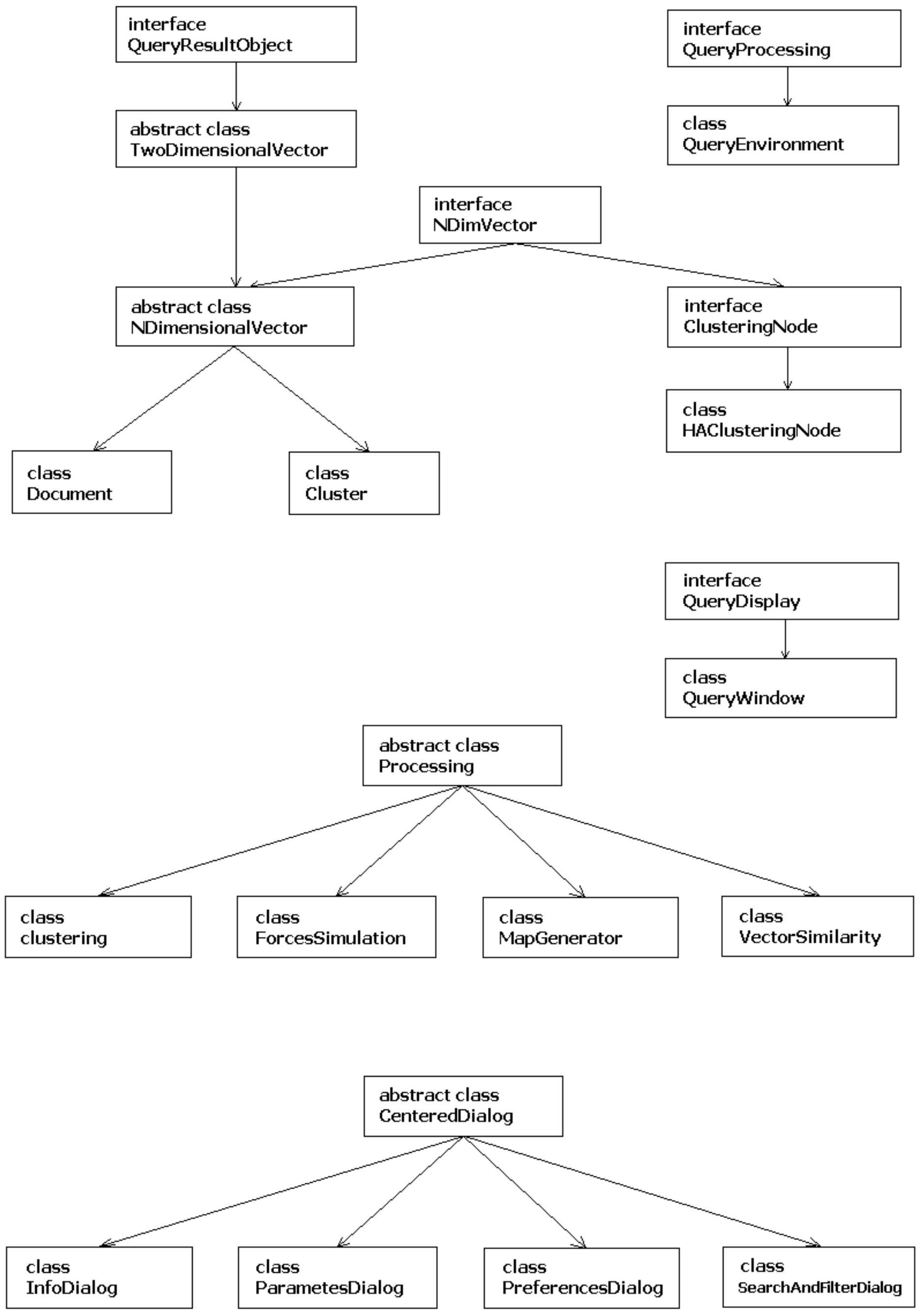


Figure 6.2: Visualisation Islands class hierarchy.

- Package visislands: Provides main program control and service classes as well as classes containing different parameters which influence the processing and the look of the program.

  - Parameters class: This class contains all parameters important for loading, processing and displaying the processing results. Visualisation Islands can handle more than one query at a time whereby each query is represented by two special objects, one at the data processing side, a QueryEnvironment instance, and one on the user interface side, a QueryWindow instance. An instance of Parameters is created by the ProgramRoot class and assigned to each query. As parameters for each query can be modified, a Parameters instance is able to check if the entered parameters are within allowed intervals and if they are consistent with each other. If this is not the case an error message is issued, inconsistencies are automatically corrected and the user is informed of the changes.
  - Preferences class: Settings that are not directly connected to a query, but apply to the program as a whole, for example which browser is used for viewing the documents, are contained in this class.
  - ProgramRoot class: There exists only one instance of this main program class. Its job is to maintain a list of existing queries, destroy queries which are not needed any more and to create new queries by creating instances of the QueryEnvironment and QueryWindow classes. When a request to create a new query is received the two classes are instantiated as a pair and connected to each other, representing a query on the data-processing and user-interface side respectively. The class also provides services to the existing queries, like starting a document viewer process through the ViewerLauncher class or creating an instance of QueryLoader which sends the query to xFIND and receives results. It also maintains one instance of Parameters class which is cloned and passed over to each new query at its creation and one instance of Preferences class which holds main program settings. One instance of MainWindow class, which contains and manages user interface components, is owned by the ProgramRoot instance.
  - ViewerLauncher class: When a QueryWindow instance sends a request to the ProgramRoot to display a document in a browser (or any viewer application) the ViewerLauncher class is used to display a document in the selected viewer. The class tries to start a standard browsing application like Netscape Navigator or Microsoft Internet Explorer, and if it does not succeed it asks the user to select an application.
  - VisIslands class: This class contains the main() method and is only used as a program name. It creates an instance of the ProgramRoot class which actually initiates the program.

- Package visislands.searchenginecommunication: Provides classes for parsing the query string, taking care of the communication with the xFIND search engine, sending the query in the desired format (QCF) and receiving and storing the returned document data in data structures provided by the visislands.dataprocessing package.

  - QueryLoader interface: This is an interface used by QueryEnvironment class to access the instance which handles the communication with the xFIND search engine. Methods are provided for setting the query string, initiating and controlling of the loading of document data, and handling of errors.
  - XFindCommunication class: XFindCommunication is an implementation of the QueryLoader interface. Communication with the xFIND server, which means sending the query and query parameters and loading the document data, is handled by an instance of this

class, using the Query Communication Format (QCF, see Section 5.3). After QueryEnvironment sends a request to the ProgramRoot instance for loading, ProgramRoot responds by delivering an instance of XFindCommunication class. This object, for each document received from xFIND, requests its parent QueryEnvironment instance to create a new Document object, and then writes the loaded data using methods of the Document class.

– XFindTokenizer class: This class is used by the XFindCommunication object to transform the query string typed by the user into the array format required by the QCF (no syntax checking is performed). The code contained in the XFindTokenizer was developed by Erwin Weitlaner [WEI99] for his Master's Thesis Metadata Visualisation Tool (SRE, see Chapter 5.4) and is used with kind permission of the author.

- Package visislands.dataprocessing: Provides data structures and algorithms for processing the data loaded from the xFIND search engine. Implemented algorithms organise the returned documents into clusters, compute the 2-D layout of the documents by positioning similar documents close to each other and pushing dissimilar documents apart, and generate the 3-D topographic map background. Also, Java interfaces are provided, used by the visislands.userinterface package, which allow manipulation and access of processing results as well as full control of the processing.

  – Cluster class: A child of the NDimensionalVector abstract class. A Cluster object can contain any number of other NDimensionalVector instances (Document or Cluster instances) which can be added or removed at will and can be checked for membership. The high-dimensional vector representation, inherited from the NDimensionalVector class, can be updated automatically when adding or removing members, if stated so in the Parameters instance of the QueryEnvironment owning the Cluster object.

  – Clustering class: This is a child of the Processing abstract class. An instance of this class is contained by each QueryEnvironment class. The code for three implemented clustering algorithms (for details on hierarchical agglomerative clustering see Section 3.3.1, for K-Means clustering see Section 3.2.2 and for single pass clustering see Section 3.2.1) and their initialisation is contained in this class. An array of Document instances (or more generally NDimensionalVector instances) is passed to the chosen algorithm and an array of Cluster instances is produced as a result. Certain important operations are delegated to other objects, like cluster centroid updating which is performed by classes representing a cluster (Cluster, HAClusteringNode, or any implementation of the ClusteringNode interface). A Clustering object makes heavy use of similarity computation functions implemented in VectorSimilarity class. For implementation details on clustering algorithms please consult Section 7.2.

  – ClusteringNode interface: This is a child of the NDimVector interface. A Cluster class is a good representation for a cluster, but different clustering algorithms need different data structures for effective operation. Classes implementing such data structures should implement the ClusteringNode interface to provide methods that allow the Cluster class to initialise itself from such objects.

  – Document class: This is a child of the NDimensionalVector abstract class. It contains document data such as, the title, URL, size and creation time as well as document keywords. When adding a keyword to a Document object this class interacts with its parent QueryEnvironment instance which maintains a dictionary of all keywords. This is necessary as a Document only holds IDs of the keywords it contains, while all existing keywords for one query and the their IDs are managed by the parent QueryEnvironment object.

- ForcesSimulation class: This class is a child of the Processing abstract class. One instance is contained by each QueryEnvironment class. The code for the force-directed placement algorithm based on stochastic sampling (see Section 4.1.4) is contained in that class, however it should be noted that some operations, like initiating and handling the neighbour and random set and the data associated with these two sets, is handled by the TwoDimensionalVector class. An array of TwoDimensionalVector instances is passed to the algorithm, which then calculates their 2-D layout positions by performing a number of simulation steps in which the forces between the objects are calculated based on their similarity in the high-dimensional space and the 2-D layout distance. During the simulation the objects are pulled together or moved apart depending on the force between them. A ForcesSimulation object makes heavy use of similarity and distance computation functions implemented in VectorSimilarity class. For implementation details of the force-directed placement algorithms please consult Section 7.3.
- HAClusteringNode class: An implementation of the ClusteringNode interface. It provides a tree-like data structure suitable for hierarchical clustering algorithm merging operation. The class maintains and automatically updates a centroid representing all objects in this object's sub-tree. Also, methods for initialising an instance of Cluster with the objects contained in the sub-tree of this HAClusteringNode instance are implemented here.
- MapGenerator class: This is a child of the Processing abstract class. One instance of this class is contained by each QueryEnvironment class. It contains the code for generating a 3-D (or 2-D) map background image based on the density of documents and clusters in the 2-D layout created by algorithms of the ForcesSimulation class. The arguments for the map background generation algorithm are an array of documents, an array of clusters, and an array and colours which are assigned to different heights. Using the 2-D positions of the documents and clusters a 3-D style topographic map background image is computed. For implementation details please consult Section 7.4.
- NDimensionalVector class: This abstract class is a child of the TwoDimensionalVector class and implements the nDimVector interface. It models an object in a high-dimesional vector space, where the dimensions of the vector space correspond to different keywords. Each entry in the high-dimensional vector is proportional to the product of frequency and weight of the corresponding keyword. It is a central class both for clustering and force-directed placement algorithms. The implementation of the NDimVector interface, which defines the internal representation (implementation) of the high-dimensional vector, is needed for the computation of the similarity values for a pair of Document or Cluster instances (classes that inherit on NDimensionalVector), which is performed by a special VectorSimilarity object. The class also provides access to data fields like title, size, URL (only in case of a document) and keywords. NDimensionalVector stores frequencies and weights of object's keywords and provides means for checking if a particular keyword is contained by the object. For this purpose NDimensionalVector class defines an internal class Keyword:
    - Keyword class: This internal class implements the java.lang.Comparable interface used for sorting the keywords in descending order of the product of their frequency and weight. The class contains a pointer to the keyword's string kept in the dictionary managed by the QueryEnvironment parent instance, a unique identifier (as defined in the dictionary) and keyword's weight and frequency.
- NDimVector interface: This interface defines an internal representation of a high-dimensional vector. Instances of classes which implement and interfaces which inherit this interface (NDimensionalVector, ClusteringNode) can be compared by calculating their similarity,

which is accomplished by VectorSimilarity class.

– Processing class: This abstract class is a parent of all processing classes (Clustering, ForcesSimulation, MapGenerator and VectorSimilarity). As processing of a query is always executed in a separate thread a mechanism for stopping the execution by setting a stop-flag to true has been implemented. This class offers methods for setting this flag's values as well as methods for informing the owner of the processing object, which is a QueryEnvironment instance, of the progress of the execution.

– QueryEnvironment class: This very important class represents a query on the data processing side of Visualisation islands. It is also a unit of execution, a thread, providing an environment in which the document data returned by the xFIND search engine are processed. Each QueryEnvironment is connected to a corresponding QueryWindow instance which is the user interface side representation of a query. The class implements the QueryProcessing interface which allows a QueryWindow object to access certain methods and data managed by an QueryEnvironment object. This object can in turn access QueryWindow methods through the QueryDisplay interface to display some info during processing and to notify the QueryWindow when the processing is finished or display an error message in case of some problem (network connection, out of memory, etc.). An instance of Parameters class, containing all relevant parameters for this query concerning the loading, processing and display, is kept by an QueryEnvironment object. It should be noted that the same Parameters instance is shared with the corresponding QueryWindow object.

  QueryEnvironment is a central class for the data processing side of Visualisation Islands as all data related to a query and objects performing the processing are contained by this class: one instance of each processing class (Clustering, ForcesSimulation, MapGenerator and VectorSimilarity), an array of Document instances, an array of Cluster instances created by some clustering algorithm, document to document, cluster to cluster and document to cluster similarity matrices, and a dictionary of all keywords belonging to a query. This dictionary consists of a String array, representing integer to string mapping, and a hash table, representing string to integer mapping. It can translate each keyword string to a unique integer identifier and the other way round which is used by NDimensionalVector instances that keep keyword integer IDs instead of whole strings, as well as for searching for and filtering objects containing a specific string(s). The QueryEnvironment class offers a number of powerful and flexible methods for finding documents and clusters with any selected set of features and combination of features (keywords, size, creation date, etc.).

  As already mentioned this class is a unit of execution. For this purpose it implements java.lang.Runnable interface and can be started and executed as a separate thread. Depending on the parameter values stored in its Parameters instance QueryEnvironment controls the loading and processing of the data and dispatches tasks and passes data to the four processing objects it owns. Error handling and notification of the user interface part of the program is also the task of this class.

– QueryProcessing interface: This interface is implemented by QueryEnvironment class. It is used by the user interface side representation of a query, the QueryWindow class, to access certain methods of the QueryEnvironment class. Through the QueryProcessing interface the user interface gains control of the data loading and processing. A QueryWindow object can start or interrupt the data processing, as well as search for and get access to loaded document data and processing results. Through this interface it is possible to access sets and arrays containing data loaded from the xFIND search engine as well as data created and by the processing procedure, but objects in these arrays can not be modified as they are accessed through the QueryResultObject interface. QueryProcessing also offers a means to extract only those objects that have certain features or satisfy some conditions concerning their contents, size, modification date, and some other attributes.

  – QueryResultObject interface: This interface is implemented by the TwoDimensionalVector class and is designed for the visislands.userinterface package, and the QueryWindow class in particular, to gain read access to data stored in every Document instance loaded or Cluster instance created by the processing performed in QueryEnvironment. This includes reading the object's attributes, like the title or the URL of the document, getting objects contained by a cluster, reading 2-D layout coordinates of an object, checking for some property, etc.
  – TwoDimensionalVector class: This class meets the needs of the force-directed placement algorithm implemented in the ForcesSimulation class. As its name suggests it models an object having a position in 2-D space. The implemented force-directed placement algorithm makes use of stochastic sampling, taking into consideration only a constant size subset of the complete object set when computing the position of each object. This reduces the execution time significantly compared to standard method of comparing each object with all other objects in the set. Each processed object maintains this subset divided into two different sets: a neighbour set containing only objects similar to this object and a random set containing a new random sample of objects for each iteration of the algorithm. TwoDimensionalVector class includes methods for initialising the neighbour set, called before staring the algorithm, and methods for filling the random set and updating the neighbour set, which are called anew for each iteration of the algorithm. TwoDimensionalVector class also implements the QueryResultObject interface which is used by classes of the visislands.userinterface package to access the processing results.
  – VectorSimilarity class: This class is a child of the Processing abstract class. One instance of this class is contained by each QueryEnvironment class. It contains the code for calculating the similarity of each two objects implementing the NDimVector interface (NDimensionalVector class, ClusteringNode interface). Three different types of similarity coefficients can be computed by this class: cosine, which is the most commonly used coefficient, dice, and jaccard. However the main task of the VectorSimilarity class is to calculate a similarity matrix for one array of objects, giving a triangular matrix, or for two arrays of objects, returning a quadratic matrix. Force-directed placement and clustering algorithms compare every pair of objects many times. Matrices containing object-to-object similarity values are therefore computed before running other algorithms, and then stored, to reduce the computation time. VectorSimilarity can also be used to calculate 2-D Euclidean distances of TwoDimensionalVector instances.

- Package visislands.userinterface: Provides classes to display the processing results, handle interaction with the user and allow the manipulation of the processing itself as well as of processing results. Interfaces are provided, used by the visislands.dataprocessing package, allowing the processing part of the program to display processing progress, notify the user in case of an error and initiate the displaying of the computed results.

  – CenteredDialog class: This abstract class extends the javax.swing.JDialog class. It is a superclass of four different dialogs: InfoDialog, ParametersDialog, PreferencesDialog and SearchAndFilterDialog. Its task is to display itself centered with respect to its owner window, however ensuring that the whole dialog is displayed inside of the screen borders.

– DetailsWindow class: Extends the javax.swing.JFrame class. When a number of objects, documents or clusters, is passed to this class, it displays a list of strings describing each of these objects. The user can select an object to display detailed information about it. This class contains five inner classes that have following tasks:
  - QueryObjectList class: Extends the javax.swing.JList class. It displays the titles of the given objects in a list and lets the user chose a document or cluster with a mouse click.
  - DetailsPanel class: Extends the javax.swing.JPanel class. It displays metadata of the document: title, URL, size, creation date, and the cluster to ehich ot belongs. For a cluster its number and the number of contained objects is shown.
  - ContentsPanel class: Extends the javax.swing.JPanel class. It displays up to four lines of the content of the selected document for the user to gain more knowledge of the document's content. The content shown are words surrounding the query terms found in that particular document[1].
  - KeywordTable class: Extends the javax.swing.JTable class. It displays a table with all keywords of an object, together with their weights and frequencies, sorted in descending order of the product of their frequency and weight.
  - ButtonPanel class: Extends the javax.swing.JPanel class. It enables the user to view the selected document in a browser (or some other viewer application) and to close the dialog.

– InfoDialog class: This is a child of the CenteredDialog abstract class which displays info about the program, the author, the author's advisors and license and copyright information.

– MainWindow class: This important class builds an environment in which all other user interface objects exist. One instance of this class is created, owned and controlled by the ProgramRoot instance, which also offers certain services, like starting a viewer application, to the MainWindow. The class offers a menu bar with pull-down menus containing most program functions, a toolbar with buttons for most important functions, and an area where any number of QueryWindows can be displayed. MainWindow owns and controls one instance of PreferencesDialog, ParametersDialog, SearchAndFilterDialog and InfoDialog and can launch a number of DetailsWindow instances. MainWindow listens to all events generated by the pull-down menus and the toolbar buttons and dispatches them to the corresponding component.

– ParametersDialog class: This class is a child of the CenteredDialog abstract class and its only task is to display a ParametersPanel instance which is used to edit data fields of the Parameters class.

– ParametersPanel class: A subclass of javax.swing.JPanel class. Its job is to edit data fields of a Parameters instance. Two modes exist: a simple mode that changes only the number of documents and clusters with the help of two sliders, and an advanced mode which displays a tabbed panel and gives access to around 70 parameters managed by the Parameters object.

– PreferencesDialog class: A subclass of the CenteredDialog abstract class used to edit the fields of a Preferences instance.

– QueryDisplay interface: This interface is implemented by QueryWindow class. It is used by the data processing side representation of a query, the QueryEnvironment class in par-

[1]For technical reasons this function is disabled in the Version 1.0 of Visualisation Islands.

ticular, to access methods of the QueryWindow class. Through these methods a QueryWindow can be driven to display a form for entering the query and its parameters, to display information about the task currently being performed by the processing, show error, warning and info messages, display processing results once they are computed, highlight a number of objects as a result of search or select operation and hide (filter) objects which do not satisfy some criteria.

– QueryInputPanel class: This class extends the javax.swing.JPanel class. It is used to enter the query string and to adjust a number of parameters which influence the loading, processing and visualisation of the data. Once the query has been sent to the xFIND search engine this panel displays a description of the current processing task and its progress. To achieve this two inner classes, both extending the javax.swing.JPanel class, are employed:

  ∗ QueryPanel class: This panel is divided in two parts. The upper area is used for typing in the query, or choosing it from the query history list, and sending it to the xFIND search engine by clicking on the Go! button. It should be noted that all event processing is done in the QueryWindow instance that owns the enclosing QueryInputPanel instance. The lower part of the panel is used to display a ParametersPanel instance which is used to adjust and edit the query's Parameters instance. By default QueryPanel is displayed and ProgressPanel is hidden.

  ∗ ProgressPanel class: When the user has typed the query, adjusted the parameters, and pressed the Go! button, this panel is shown and QueryPanel is hidden. A ProgressPanel instance gives the user detailed information on how the loading and processing is proceeding. A textual description of the currently performed task together with its progress is displayed and refreshed continuously.

– QueryResultsPanel class: This class is, together with QueryWindow, the most important class of the userinterface package. Each QueryWindow instance owns one instance of QueryResultsPanel which displays the documents returned by xFIND, after they have been processed by the clustering, force-directed placement and map generation algorithms. QueryResultsPanel visualizes the document partition created by some of the clustering algorithms in a table-of-contents manner and presents the topographic map of topically organised documents. The class also handles zooming, displays selected objects highlighted, hides objects that do not satisfy the filter condition, displays info for the object the mouse is pointing to and scrolls automatically the table-of-contents representation to display the object(s) selected in the map. Events generated by user interactions are not handled by the class itself, but by the QueryWindow instance owning this QueryResultsPanel object. While the QueryWindow takes actions to handle the user interactions and subsequently changes the state of the QueryResultsPanel, the QueryResultsPanel instance takes care that the current state is properly visualised. To achieve this the class delegates some responsibilities to four internal classes described below. QueryResultsPanel is visually divided in three parts. The bottom part is a StatusPanel displaying object details, while the upper part is divided by a javax.swing.JSplitPane into the left half, occupied by a ClusterDocumentList, and a right half which visualises the map using a javax.swing.JLayeredPane. JLayeredPane is a special panel for handling panels which are placed over each other each having a different depth. It was chosen to conveniently brake up the map into a BackgroundPanel and a ForegroundPanel.

  ∗ ClusterDocumentList class: Extends javax.swing.JList class. This class displays the partition of the document set, produced by clustering methods implemented in the Clustering class, in a table-of-contents style. The javax.swing.JList class allows the user to select and deselect its elements and maintains an internal selection model.

This selection model is used by the enclosing QueryResultsPanel instance to manage the selecting and deselecting of the displayed objects.

- BackgroundPanel class: Extends the javax.swing.JPanel class. This task of this class is to display the computed topographic map of topically organised documents. The map consists of several components, each computed by different algorithms, which are displayed all together forming the main visualisation offered by the program. The first component, documents and clusters icons, are placed at positions computed by the force-directed placement algorithm implemented in ForcesSimulation class, according to their similarity. The icon shape is different for document and cluster and their sizes vary depending on the width and height of the BackgroundPanel instance. The second map component is the map background image (3-D or 2-D style) computed by MapGenerator class, depending on 2-D documents and clusters positions and their local density in the 2-D space. The background image is displayed under the icons to give clue where large amounts of similar objects can be found. It is dynamically resized to fill the current BackgroundPanel area. The third component are the labels placed close to cluster icons giving the user the possibility to immediately find areas of interest. Labels are those keywords with the highest imporance extracted from the high-dimensional term vectors of clusters, where importance of a keyword is defined as the product of its frequency and weight. The number of currently displayed labels and their sizes depend on the size of the BackgroundPanel instance. If the number of clusters currently displayed is larger than the maximum allowed number of icons, which can vary depending on the zoom degree, labels of large clusters have priority. The last map component is a grid displayed over all other components to improve the user's orientation. Together these four components create a metaphor of a topographic map populated by documents positioned into topically similar groups.
- ForegroundPanel class: This class extends the javax.swing.JList class. Its task is to display the user interactions of zooming and selecting. When the user drags the mouse a rectangle enclosing the area that will be zoomed is displayed, if in zoom mode, or, if in select mode, a circle is drawn and all objects inside of it will be selected when the mouse button is released.
- StatusPanel class: Extends the javax.swing.JPanel class. This class displays the title, URL and the keywords of the object that the mouse pointer is positioned on. The keywords are sorted and displayed by descending importance.

– QueryWindow class: This class extends the javax.swing.InternalFrame class. It represents a query on the user interface part of the program. The class implements the QueryDisplay interface used by the QueryEnvironment class, which is the data-processing-side representation of a query, to access methods of the QueryWindow. Each QueryWindow is connected to a corresponding QueryEnvironment instance, with which it shares a Parameters instance. On the other side a QueryWindow object can access QueryEnvironment methods through the QueryProcessing interface to control the loading and processing of the data as well as to get access to processing results. A QueryWindow owns an instance of the QueryInputPanel class for entering the query, setting the query parameters and displaying the processing progress. When query results are loaded and processed QueryInputPanel is hidden and QueryResultsPanel is shown to displays the results. A very important aspect of the QueryWindow class is that it is a listener for all user events fired by its QueryInputPanel and QueryResultsPanel instances. The class processes every user interaction with the query represented by this QueryWindow instance and dispatches actions to other classes like MainWindow, ProgramRoot and QueryEnvironment. To achieve this the class implements following event listeners: InternalFrameListener, ActionListener, ComponentListener, ChangeListener, ListSelectionListener, MouseInputListener, KeyListener.

- SearchAndFilterDialog class: This child of the CenteredDialog abstract class is used to enter the upper and lower limit for size and date and some keyword string(s) with the goal of searching for and filtering the documents and clusters currently displayed.
- StartupWindow class: This class is a child of javax.swing.JWindow class. While the program is loading it displays a startup image in a window without a title bar and window management buttons. The StartupWindow will try to display itself atop all other windows on the screen and will remain visible for a specified number of milliseconds or until the program is completely loaded. By clicking on it the window will close.

Chapter 7

Selected Details of the Implementation

Visualisation Islands is composed of over 50 classes organised in four packages. As already explained in Chapter 6 the visislands package takes care of the program control and servicing other packages, the visislands.searchenginecommunication package parses the user's query, sends it to the xFIND search engine, receives results and passes them to the visislands.queryprocessing package which processes the loaded document data, creates the visualisation and maintains the processing results, which are used by visislands.userinterface package to display the results and handle user interactions like zooming, selecting, viewing and searching. Each of these packages constitutes an important part of Visualisation Islands software, however, the most important classes and algorithms, those that blow life into the program, are contained in the visislands.queryprocessing, while other packages are concerned with providing data to the algorithms, displaying the processing results and providing other services.

When the document data is loaded as a response to a query the documents are transformed into high-dimensional vectors, according to the keywords returned as their representation. In this vector space model (see Section 2.2) every keyword represents a dimension in the high-dimensional space. A vector representing an object has an entry for each keyword in the object set, and each entry is proportional to the frequency of the keyword in that particular object multiplied by its weight (its weight is for example higher if the keyword is in the document's title). The product of frequency and weight of a keyword is the keyword's importance. This vector form gives the possibility to compute object-to-object similarity values and thus to compare objects having a vector representation. The algorithms used to compute and create the visualisation rely heavily on this similarity function to perform their task. The similarity coefficient for two objects is a value between 0 and 1, where a value of 1 means complete similarity and a value of 0 indicates no similarity at all. As each pair of objects may be compared more than once during the execution of different algorithms, it is a good idea to pre-compute and store all inter-object similarities in a matrix. Such a similarity matrix serves as a similarity values cache and greatly improves the execution speed of the implemented algorithms. The visualisation offered by Visualisation Islands software package is based on three different algorithms, or classes of algorithms, which are performed successively, each providing data used by the following steps. These are:

- Clustering algorithms: Clustering is the first processing step which places all documents in the given set in a predefined number of clusters in such a way that each cluster contains only documents which are similar to each other. A cluster is represented by its centroid which is a high-dimensional vector composed of terms of all documents contained in the cluster. A centroid of a cluster is the centre of gravity of its documents. Thus, a cluster can be seen as a pseudo-document and can be compared to other clusters and documents or to any other object that has a high-dimensional vector representation. Three different clustering algorithms are implemented: the single pass clustering, k-means clustering with continuous centroid updating, and hierarchical agglomerative clustering. The first two, being less precise but faster can be applied to a larger number of documents, while the third algorithm gives the best results at the cost of high execution time.

- Force-directed placement algorithm: The force-directed placement algorithm performs a simulation in which 2-D positions of objects with high-dimensional vector representation are calculated depending on their similarities and their distances in the 2-D space. Attractive or repulsive forces between objects are calculated depending on these two values. Similar objects in high-dimensional space are pulled together in the 2-D layout to form visual clusters of similar documents, while dissimilar objects are pushed apart. As a result, objects from the high-dimensional space are mapped to 2-D space preserving their relationships (distances) as well as possible. Clustering results can be used to improve the results and reduce computation time by first processing cluster centroid vectors, then placing the clusters' documents around the computed positions and running the algorithm again. To reduce the complexity of the algorithm, not all vectors are compared in every iteration, but a stochastic sampling technique with a neighbour and a random set is used.

- Map background generator: Based on the calculated 2-D layout positions of documents and clusters the algorithm generates a 3-D landscape bitmap image reflecting document and cluster density in the 2-D plane. The principle of the algorithm is to imagine each element in the layout to represent a small peak with a base of circular form. When more elements are placed close to each other on a small area, these peaks overleap and are added together to build larger and higher peaks - the mountains. As similar documents are gathered together by the force-directed placement algorithm to build visual clusters, the result of generating an image from such layout are islands of similar documents divided by some empty or scarcely populated areas - the sea. By choosing colours proportional to the height and doing some shading a 3-D landscape image is generated, which together with documents and clusters displayed as icons on corresponding 2-D positions, and labels extracted from clusters' vectors and displayed close to corresponding cluster icons, results in an interactive 3-D style topographic map of the original document set.

The particular implementations of different algorithms used in Visualisation Islands software package will presented in this section, using natural language, mathematical expressions and pseudo code. The goal is to emphasize important parts of the algorithm and to avoid getting lost in less important details. Therefore, exact implementations of objects such as clusters or high-dimensional vectors is not presented here, but some of their properties are explained if necessary for better understanding of the algorithms.

7.1 Similarity Computation

The most common similarity measure is the cosine of the angle between two vectors, but other similarity coefficients, like dice or jaccard, can also be used in Visualisation Islands by simply setting a corresponding parameter. Please see section 2.2.1 for exact coefficient definitions. As the same pairs of objects can be compared many times during processing, inter-object similarity measures are stored in a similarity matrix where they can be accessed in a fast and straightforward manner. If a similarity matrix for a single array of objects is needed then a triangular matrix, with its diagonal set to 1, is used, as a quadratic matrix would be symmetric and waste space. Of course, if similarity values for objects from two different arrays are needed a full matrix is computed. For the rest of this chapter the following conventions are used:

- Method $similarity(O_1, O_2)$ returns the similarity between the high-dimensional vectors of objects O_1 and O_2, by computing it explicitly or by retrieving it from a pre-computed similarity matrix, depending on the need.
- Method $similarityMatrix($Array $AO)$ computes and returns a $n \times n$ triangular similarity matrix with object-to-object similarity values for objects from the array AO. n is the number of objects in the AO array.
- Method $similarityMatrix($Array $AO1,$ Array $AO2)$ computes and returns a $m \times n$ similarity matrix with similarities values for objects from the array $AO1$ compared to objects from the array $AO2$. The values m and n are the numbers of objects in the arrays $AO1$ and $AO2$ respectively.
- Method $distance(O_1, O_2)$ computes the 2-D Euclidean distance between two 2-D vectors.

7.2 Clustering Algorithms

Three standard clustering algorithms have been implemented: the single pass clustering (see Section 3.2.1), the k-means algorithm (see Section 3.2.2) and the hierarchical agglomerative clustering algorithm (see Section 3.3.1). Each of these algorithms was already discussed in Chapter 3 in their generic form. They place all documents from the given set into a predefined number of clusters, each containing similar documents. All documents are normalised using the Euclidean norm prior to clustering, although a maximum norm, or no normalisation at all, can also be employed. Like the documents, every cluster is represented by a high-dimensional vector, its centroid. Visualisation Islands computes the centroid as the center of mass (gravity) of all documents contained in the cluster. Such a cluster can be updated in $O(dim)$ time, where dim is the dimensionality of the high-dimensional space. As this value is constant for a given document set, removing and adding document from and to the cluster is a simple task performed in constant time. With $d_i[h]$ representing the h-th coordinate of the i-th document, the h-th coordinate of the centroid of a cluster with n documents, $c[h]$, is defined as:

$$c[h] = \frac{\sum_{i=1}^{n} d_i[h]}{n} \text{ where } h = 1...dim$$

Both partitional algorithms, k-means and single pass, make use of the $createZeroDimensionCluster$ method. This method is used to extract documents which have no keywords and are placed at the origin of the high-dimensional space. Before the actual algorithms are run, all such documents are put in a separate cluster as they would have a negative influence on the clustering algorithms.

```
Method $createZeroDimensionCluster($array of documents $AD)$:
• Create an empty cluster $C_{zero}$.
• For all documents $D$ in the array $AD$
    – Get the document's high-dimensional vector $V$.
    – If all components of $V$ are zero put document $D$ into cluster $C_{zero}$.
• If cluster $C_{zero}$ is not empty return it, otherwise return $NULL$.
```

7.2.1 Single Pass Clustering

The main idea of this algorithm is to consider every document exactly once (hence the name single-pass), compare it to all existing clusters and put it in the best fitting one, but only if the similarity between the document and the best fitting cluster is higher than a predefined threshold, otherwise a new cluster with this document as its only member is created.

```
Method $singlePassClustering($array of documents $AD,$ empty array of clusters $AC)$:

• Cluster $C = createZeroDimensionCluster(AD)$.
• if $C$ is not $NULL$ put $C$ in $AC$.
• For all documents $D$ in the array $AD$.
    – For all clusters $C$ in the array $AC$.
        ∗ Calculate $similarity(D, C)$, remember the best fitting cluster $C_{best}$
    – if $(similarity(D, C_{best}) >=$ similarity-threshold$)$
        ∗ Put document $D$ in the cluster $C_{best}$.
        ∗ Update the centroid of $C_{best}$.
    – otherwise
        ∗ Create new cluster $C_{new}$.
        ∗ Put document $D$ in the cluster $C_{new}$.
        ∗ Put cluster $C_{new}$ in the cluster array $AC$.
        ∗ Update the centroid of $C_{new}$.
• run $kMeansReallocationalClustering(AC)$ to refine the partition.
```

This algorithm is quite simple and fast because it passes through the document set only once. The algorithm tends to build larger clusters at the beginning, with the size of those clusters created when most documents are already assigned getting ever smaller. The partition depends strongly on the order of the processed documents. The consequence of this is that the quality of the obtained partition is not very good, which is the reason why a k-means reallocational algorithm is subsequently launched to refine the partition. Due to two nested loops the execution time of the algorithm is $O(mn)$ where m is the number of created clusters and n is the number of documents. The number of created clusters depends on the similarity threshold value, where a small value gives a smaller number of clusters.

7.2.2 K-means Clustering

The k-means clustering is a reallocational iterative algorithms that requires an initial partition to be created in advanced. The algorithm refines the given partition by passing through the document set inspecting each document and replacing it, if necessary, to the best fitting cluster. As this procedure is repeated a number of times the partition converges towards a local minimum with each performed iteration. The algorithm can be implemented with continuous or non-continuous centroid updating. Continuous centroid updating means that whenever a document is moved from one cluster to another both cluster centroids are immediately updated. With non-continuous centroid updating cluster centroids are updated at the end of an iteration when all documents have been inspected. Visualisation Islands uses continuous centroid updating.

The initial partition passed to the k-means algorithm for refining can be generated in two different ways. The first is to run a single pass clustering algorithm as described in the previous subsection. The second way is to randomly choose a predefined number of documents as initial cluster seeds and add all other documents to the best fitting seed, which actually is a k-means algorithm performing only one iteration after creating the initial seeds. The procedure to build a random initial partition is implemented by the following method:

```
Method $buildInitialPartition($array of documents $AD,$ empty array of clusters $AC)$:

• Cluster $C = createZeroDimensionCluster(AD)$.
• if $C$ is not $NULL$ put $C$ in $AC$.
• For all documents $D$ in the array $AD$
    – if $D$ is not in cluster $C$ put $D$ in the list of free documents $FDL$
• While ($size(AC) <$ required number of clusters) do:
    – Choose a random document $D$ from $FDL$ and remove it from $FDL$
    – Create a cluster $C$ with document $D$ as its only member
    – Add cluster C to cluster array AC
• While ($FDL$ is not empty) do:
    – Take a document $D$ from $FDL$ and remove it from $FDL$
    – Pass all clusters in $AC$ and find cluster $C$ that is most similar to document $D$
    – Add $D$ to $C$ and update the centroid of $C$
```

When an initial partition is created each document from the document array AD is a member of exactly one cluster from the cluster array AC. However not every document will belong to a cluster most similar to it, a result of centroid movement as more and more documents were added to each cluster. To improve the partition an iterative refinement procedure, the k-means algorithm, is applied by passing every document in the set and reallocating it, if necessary, from the current cluster to the cluster with the most similar centroid. The procedure is repeated several times until a stop condition is reached. There are a number of different stop conditions or combination of these which can be found in the literature. The one implemented in Visualisation Islands is relatively complex. Another iteration is executed if following three conditions are satisfied: the maximum number of iterations has not yet been reached, the number of replaced documents compared to the number of the documents in the complete set is still higher than a predefined threshold, the relative improvement of quadratic error, or distortion in Visualisation Islands terminology, over the previous two iterations is higher than a predefined treshold. The k-means reallocational clustering procedure is implemented as follows:

```
Method $kMeansReallocationalClustering($array of documents $AD,$ array of clusters $AC)$:

• new distortion $= distortion($array of documents $AD)$
• Let iteration $= 0$
• Repeat
```

```
    – iteration = iteration + 1
    – reallocated documents = 0
    – Let old distortion = new distortion
    – For all documents $D$ in the array $AD$
        * Let $C_{old} =$ parent cluster($D$)
        * Consider all clusters in $AC$ and find cluster $C_{new}$ most similar to document $D$
        * if $C_{old} <> C_{new}$
            · replace $D$ from $C_{old}$ to $C_{new}$
            · Update centroids of clusters $C_{old}$ and $C_{new}$
            · reallocated documents = reallocated documents + 1
    – new distortion $= distortion($array of documents $AD)$
• Until (reallocated documents / size of($AD$) <= reallocation threshold) OR
  (new distortion / old distortion >= distortion threshold) OR
  (iteration >= maximum iteration)
```

The distortion for a particular partition is defined as the sum of the squared high-dimensional distances between each document and the centroid of the cluster it belongs to. However, it is not in the sense of Visualisation Islands to use high-dimensional distances, but rather to operate with similarity values based on high-dimensional vectors, a value $(1 - similarity)$ is used instead of a high-dimensional distance. Distortion, or the squared error is calculated by the following method:

```
Method $distortion($array of documents $AD)$:

• $distortion = 0$
• For all documents $D$ in the array $AD$
    – $C =$ parent cluster($D$)
    – $distortion = distortion + (1 - similarity(C, D))^2$
• return $distortion$
```

The k-means algorithm converges towards a local minimum. If the maximum number of iterations is limited, which is the case in Visualisation Islands, the execution time is $O(mn)$, where m is the number of clusters and n the number of clustered objects. Putting a limit on the number of performed iterations is usually not a problem since convergence occurs rapidly in the large majority of cases. The quality of the reached partition depends very strongly on the choice of the initial partition which is probably the largest drawback of this algorithm, as there is no sure way of guessing the best initial partition of an object set. Despite this k-means is one of the most popular clustering algorithms, because it offers a very good trade off between speed and quality. In many cases the initial partition problem is solved by running the algorithm several times using different initial partitions and taking the result with the smallest squared error. This is a very time consuming procedure, so one may want to consider heuristic procedures for generating an initial partition as described in Section 3.2.2. The same section discusses the influence of continuous versus non-continuous centroid updating. Continuous centroid updating leads to faster convergence and is much less sensitive to the choice of initial partition. Non-continuous centroid updating can produce empty clusters which must be eliminated and at the same time some other clusters must be split into two if the desired number of clusters should be obtained. Continuous centroid updating avoids this problem. For these reasons this strategy was implemented as standard for k-means algorithm providing reasonably good clustering results with relatively small processing and implementation overhead. Note that non-continuous centroid updating is also supported by the implemented algorithms. The chosen strategy has one negative aspect: a similarity matrix containing pre-computed document to cluster similarity values can not be used. The reason for this is that each time a document is moved form one cluster to another their centroids are recomputed and two rows of the similarity matrix (containing similarity values of these two clusters to all other documents) would have to be recomputed. If the number of replaced documents in one iteration is larger than the number of clusters divided by two (which is especially true at the beginning) it is less time consuming to compare documents and clusters directly by explicitly computing the similarity coefficient. However, it has been found that this does not lead to a significant increase in processing time as the high-dimensional vectors are not excessively large for a typical document set processed by Visualisation Islands.

7.2.3 Hierachical Agglomerative Clustering

Hierarchical agglomerative clustering follows a simple and effective idea. At the beginning each document is treated as a cluster with one element. All existing clusters are compared and the two most similar in the set are merged together forming a new cluster. A number of different methods for comparing clusters of objects are described in the literature (single link, complete link, etc., see Section 3.3.1). Visualisation Islands uses a centroid-based approach: cluster centroids are computed and used to compare clusters with other high-dimensional vectors. For improved performance this implementation uses a pre-computed triangular similarity matrix storing similarity coefficients for the whole document set. Each time a new cluster is formed by merging two others the row and the column of one cluster are marked as not used, whilst the row and column containing similarity values of the other cluster are updated with values for the newly formed, merged cluster. The merging of clusters is repeated until the number of clusters is reduced to the number defined by the user, or until all existing clusters have similarity coefficients of 0 meaning that there no relation between any of them and further merging would make no sense.

```
Method $hierarchicalAgglomerativeClustering$(array of documents $AD$, array of clusters $AC$):

• For each document $D_i$ create a cluster $C_i$ containing only this document and put $C_i$ in $AC$
• $matrix = similarityMatrix$(array of clusters $AC$)
• Mark all $matrix$ rows and columns as USED
• Repeat
    – Inspect all USED $matrix$ values and find $C_i$ and $C_j$ with the largest similarity $s_{ij}$
    – if $s_{ij} > 0$
        ∗ Merge $C_i$ and $C_j$ to $C_i$ and update its centroid
        ∗ Remove $C_j$ form $AC$
        ∗ Mark $j$-th row and column of $matrix$ as NOT USED
        ∗ Update $i$-th row and column of $matrix$ with similarity values for $C_i$
```

- Until $(length(AC)$ <= desired number of clusters) OR $(s_{ij} == 0)$

The similarity matrix must be searched for the largest similarity coefficient for every merge that is performed. As the similarity matrix is of quadratic size and the number of merges is maximally $n - 1$ the execution time of this algorithm is $O(n^3)$ in worst case, where n is the number of documents.

7.3 Force-Directed Placement

The force-directed placement algorithm computes a 2-D layout for a set of high-dimensional vectors preserving their high-dimensional relations as well as possible. This is achieved by placing the objects in 2-D space and computing forces between them depending on the similarities of their high-dimensional vectors and their 2-D distances. The implementation of force-directed placement is based on an algorithm described in [CHA96a]. Similar objects are pulled together by an attractive force whilst dissimilar are pushed apart by a repulsive force. Instead of comparing each object to all other objects in the set to compute its new position, each object is compared to a constant size sample of the complete set.

Every object maintains its own sample in the form of two special sets: the neighbour and the random set. The idea is to keep a few similar objects in the neighbour set and to take a new random sample of the complete object set for each processing iteration and store it in the random set. Before the algorithm is started, the object's neighbour set is filled with random elements and its least similar element is identified. In Visualisation islands terminology this neighbour element is called the worst neighbour. For every iteration of the algorithm the random set is filled with new elements, but before a new element is inserted into the random set it is compared to the worst neighbour. If the randomly chosen element is more similar to the object owning the sets than the worst neighbour, the worst neighbour is removed from the neighbour set and replaced by the randomly chosen element.

In this way the neighbour set is continuously updated to contain objects having a high similarity to the object owning the sets. In each algorithm iteration each object is compared to all objects in its sets and a force between each pair of compared objects is computed depending on their similarity and distance in the 2-D space. If objects are similar the force is attractive and the object owning the sets is moved towards the objects it is compared to, otherwise the force is repulsive and the object is pushed away. Averaging these movements over all elements in the sets gives the new position of the object. This procedure is applied on all objects and then repeated a number of times proportional to the number of processed objects. The result is a layout in which similar objects are placed close together in the 2-D space.

Before the force-directed placement algorithm is executed three similarity matrices are computed: a rectengular document-to-clusters matrix and triangular document-to-document and cluster-to-cluster matrices. The $similarity(O1, O2)$ function used in methods of this section accesses these matrices to retrieve similarity coefficients for objects $O1$ and $O2$.

doc-to-cluster matrix $= similarityMatrix($Array of documents $AD,$ Array of clusters $AC)$
doc-to-doc matrix $= similarityMatrix($Array of documents $AD)$
cluster-to-cluster matrix $= similarityMatrix($Array of clusters $AC)$

Each object stores its 2-D coordinates and maintains its neighbour and random sets. Java offers excellent data structures for implementing the sets and set operations such as inserting and removing of elements and checking membership. These structures are based on hash tables (constant amortised time set operations), red-black balanced sorted trees (amortised logarithmic time set operations, constant time smallest and largest member retrieval) and resizeable arrays. Only the functionality of the implemented set data structures is described here, detailed implementation is not given. The initialisation of the neighbour set NS and updating of the neighbour set and the random set RS for an object O is performed by following two methods. Note that for improved performance the neighbour set of a document is initialised with documents from the same cluster as the original document (as far as it is possible), because these are expected to be similar to the object owning the sets.

```
Method $initNeighbourSet($integer $NSsize,$ array of objects $AO)$:

• Clear neighbour set $NS$
• Worst neighbour $WN = null$
• Repeat
    – if $size(parentCluster(O)) > (size(NS) + 1)$
        ∗ Object $N =$ random element of $parentCluster(O)$
    – else
        ∗ Object $N =$ random element of $AO$
    – if $((N <> O)\ AND\ NOT(N \in NS))$
        ∗ Insert $N$ into $NS$
        ∗ if $(similarity(N, O) < similarity(WN, O))$ then $WN = N$
• Until $(size(NS) = NSsize)$
```

```
Method $updateSets($integer $RSsize,$ array of objects $AO)$:

• Clear random set $RS$
• Repeat
    – Object $R =$ random element of $AO$
    – if $((R <> O)\ AND\ NOT(R \in NS)\ AND\ NOT(R \in RS))$
        ∗ if $(similarity(R, O) > similarity(WN, O))$ then
            · Remove $WN$ from $NS$
            · Insert $R$ into $NS$
            · $WN = N$ with $similarity(N, O) = min_{N_i \in NS}(similarity(N_i, O))$
        ∗ else
            · Insert $R$ into $RS$
• Until $(size(RS) = RSsize)$
```

The implemented force-directed placement algorithm with stochastic sampling makes use of a $force$ function to compute the force between two objects as a function of their similarity and their distance in the 2-D space. The similarity between the two objects, a value between 0 and 1 is raised to the power of $discriminator$, which, if greater than 1, increases the power of differentiation between similar and less similar objects. Multiplied by a constant $sfactor$ this term represents the attractive force component dependent upon the similarity between the pair of objects. The second term, a product of the $dfactor$ constant and the 2-D distance between objects, adds another, much weaker attractive component. Being proportional to the distance this component prevents the objects from flying apart into infinity. The third term, $rterm$, is a constant repulsive term that pushes non-similar objects apart. The attractive distance proportional term and the constant repulsive term should be looked upon together. When the similarity between a pair of objects is low there is a distance at which the force is 0. If the distance becomes smaller, a repulsive force appears keeping dissimilar objects apart. If the distance increases, an attractive force prevents the object set from breaking apart completely. The higher the similarity between a pair of objects the higher the attractive similarity term of the force and consequently the distance for which the force is 0 becomes smaller allowing similar objects to come closer together.

```
Method $force$(Object $O1,$ Object $O2$):

• return $sfactor * similarity(O1, O2)^{discriminator} + dfactor * distance(O1, O2) - rterm$
```

Every object processed by the force-directed placement algorithm has two pairs of coordinates: $(oldX, oldY)$ and $(newX, nexY)$. Old values store coordinates computed in the previous iteration, the new values store object's new coordinates computed on the base of old coordinates of objects being elements of the object's neighbour and random sets. Before the actual processing by the force-directed placement algorithm begins, clusters are placed at random positions of the 2-D space inside the interval $[(0, 0), (1, 1)]$ and their neighbour sets are initialised.

To improve the quality of the layout the algorithm is first applied to clusters and once their 2-D positions are computed the documents are placed very close to their parent clusters instead of being placed at completely random positions. Documents' neighbour sets are initialised and the algorithm is then applied again on documents and clusters together. This two step procedure has two positive effects: it reduces the possibility for a document to get stuck somewhere in a local minimum before reaching objects similar to it and reduces the number of iterations the algorithm needs to reach a good layout. In every iteration all objects being processed are considered and each of them is compared to all objects contained in its neighbour and random sets. The similarity of a pair of objects is read from one of the three pre-computed similarity matrices and the 2-D euclidean distance between them is computed. These values are used to calculate the force between the processed object and the influencing object. Proportional to the value of the force and depending on positions of both objects, a movement along the line connecting the objects is computed for the processed object, separately for the X and Y coordinate. A positive force moves it towards and a negative force moves it away from the influencing object. A force of 0 causes no movement at all. Every movement results in a special new position of the processed object and all these positions caused by all objects in its sets are added together and averaged giving a new 2-D position. At the end of the iteration, when all objects have been moved to new positions, new coordinates are copied to the old coordinates and the whole process is repeated. The number of iterations needed to reach a good, stabilised layout is roughly proportional to the number of processed objects. This fact is used as a rule of thumb for the termination condition.

```
Method $calculate2DPositions$(array of documents $AD,$ array of clusters $AC$):

• For each $C \in AC$ do
    – $C.oldX = random(0, 1)$
    – $C.oldY = random(0,1)$
• For each $C \in AC$ do $C.initNeighbourSet($integer $NSsize,$ array of clusters $AC)$
• Let $iteration = 0$
• Repeat
    – Let $iteration = iteration + 1$
    – For each $C \in AC$
        * $C.updateSets($integer $RSsize,$ array of clusters $AC)$
        * Let $tx = ty = 0$
        * For each $N \in C.NS$
            · $tx = tx + force(C,N) * N.oldX + (1 - force(C,N)) * C.oldX$
            · $ty = ty + force(C,N) * N.oldY + (1 - force(C,N)) * C.oldY$
        * For each $R \in C.RS$
            · $tx = tx + force(C,R) * R.oldX + (1 - force(C,R)) * C.oldX$
            · $ty = ty + force(C,R) * R.oldY + (1 - force(C,R)) * C.oldY$
        * $C.newX = tx/(size(C.NS) + size(C.RS))$
        * $C.newY = ty/(size(C.NS) + size(C.RS))$
    – For each $C \in AC$ do
        * $C.oldX = C.newX$
        * $C.oldY = C.newY$
• Until (iteration == size(AC))
• For each $D \in AD$ do
    – $D.oldX = parentCluster(D).oldX + (random(0,1)/1000)$
    – $D.oldY = parentCluster(D).oldY + (random(0,1)/1000)$
• For each $D \in AD$ do $D.initNeighbourSet($integer $NSsize,$ array of documents $AD)$
• Let $AO = concatenate(AD, AC)$
• Let $iteration = 0$
• Repeat
    – Let $iteration = iteration + 1$
    – For each $O \in AO$
        * $O.updateSets($integer $RSsize,$ array of objects $AO)$
        * Let $tx = ty = 0$
        * For each $N \in O.NS$
            · $tx = tx + force(O,N) * N.oldX + (1 - force(O,N)) * O.oldX$
            · $ty = ty + force(O,N) * N.oldY + (1 - force(O,N)) * O.oldY$
        * For each $R \in O.RS$
```

```
            · $tx = tx + force(O,R) * R.oldX + (1 - force(O,R)) * O.oldX$
            · $ty = ty + force(O,R) * R.oldY + (1 - force(O,R)) * O.oldY$
        * $O.newX = tx/(size(O.NS) + size(O.RS))$
        * $O.newY = ty/(size(O.NS) + size(O.RS))$
    – For each $O \in AO$ do
        * $O.oldX = O.newX$
        * $O.oldY = O.newY$
• Until $(iteration == size(AO))$
```

Standard force-directed placement algorithms compare each object to all other objects in the collection, therefore the number of operations during one iteration is $O(n^2)$ with n being the number of processed objects. The stochastic sampling version implemented in Visualisation Islands compares each objects with an object sample of constant size so that one iteration has a time complexity of $O(n)$. Although some extra overhead is needed to manage the sets this is a huge improvement over the standard, non-sampling version. The number of iterations needed to reach a stable layout is roughly proportional to the number of processed objects although this depends very much on the properties of the object set and, to a lesser degree, on objects' initial positions. Therefore the time complexity of the sampling algorithm is roughly $O(n^2)$ compared to $O(n^3)$ of the standard version. It should also be noted that $O(n^2)$ is a lower boundary for all algorithms that need all object-to-object similarity values to be computed as the similarity matrix computation is of that time and space complexity. The sampling algorithm has also another advantage over the conventional version. The fact that each object is influenced by similar objects from the neighbour set which is continuously updated and improved during each iteration, and that the number of less similar objects, stored in the random set, it is influenced by is small compared to n, ensures that the object is quickly attracted to objects similar to it and is not distracted by a larger number of dissimilar object present in the collection. This characteristic leads to faster building of visual clusters of similar objects. The algorithm is also less sensitive to becoming stuck in a local minimum then the non-sampling version, because of the certain amount of jitter which is brought in by taking a new sample for the random set for every iteration.

7.4 Map Background Generation

Map background generator is an algorithm to compute a 3-D style topographic map background image based on the 2-D document and cluster coordinates computed by the force-directed algorithm. Each element in the layout represents a small peak with a shape of a spherical cap thus having a base of circular form. The height of a peak representing a cluster and the radius of its base are larger than document peaks. The objects are placed on a matrix where each element of the matrix represents one pixel, as shown in Figure 7.1. If more objects are placed close to each other on a small area the peaks overleap and their heights are accumulated to build larger and higher structures - the mountains. Now every matrix element contains the height of the mountain above it. A number of colours can be passed to the algorithm (otherwise a default set of colours is used) and each is assigned to a different height interval. To obtain a 2-D style background the algorithm creates an image and writes the color values depending on the height of the corresponding element of the matrix.

If a 3-D style background is desired some additional operations are performed. A colour palette is computed from the given colours so that continuous shades exist for every color, from darkest to lightest. Furthermore, a slope is assigned to each pixel by calculating dh/dp as shown in Figure 7.2. The colour shade taken is proportional to the slope, a lighter shade is taken if the slope is positive

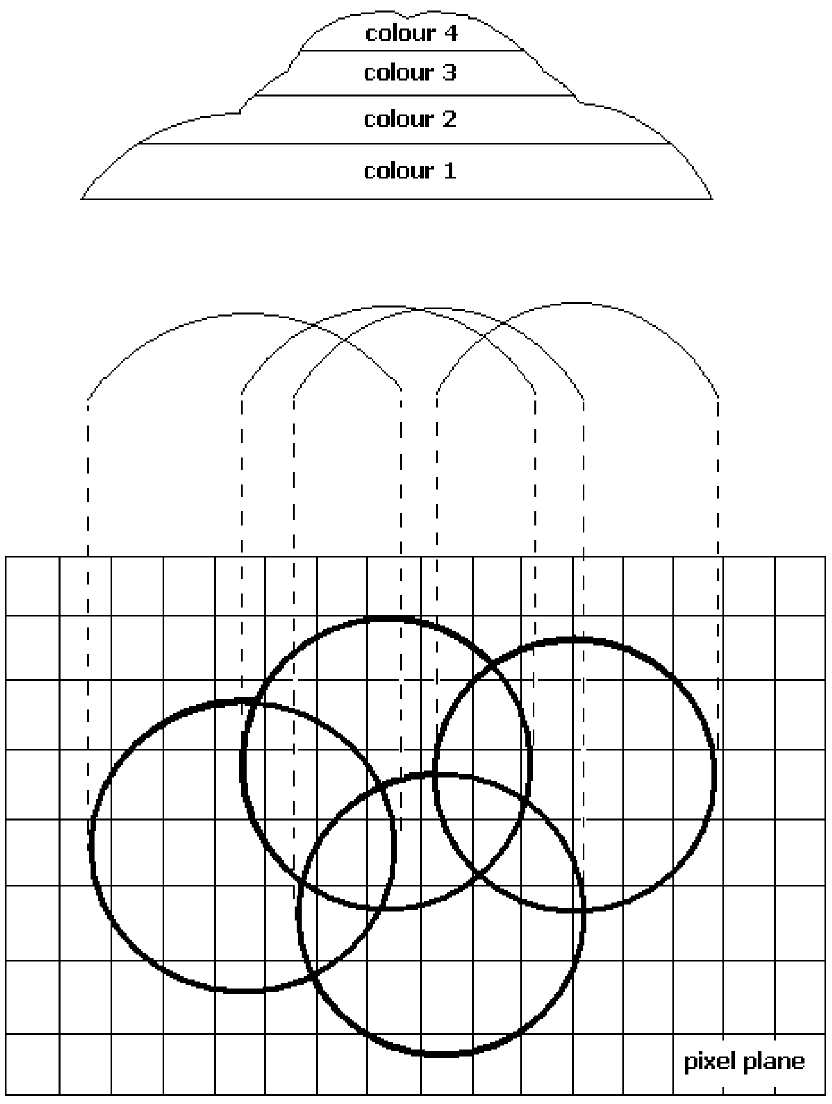


Figure 7.1: Four objects with overlapping peaks form a mountain.

and a darker one if the slope is negative. It is very important to note that the slope should never be measured with the help of neighbour pixels. Using a larger pixel window makes smaller details disappear, but at the same time the surface is smoothed giving a realistic landscape image reflecting document and cluster density in the 2-D plane.

The execution time of this algorithm depends on a number of factors. The most important is the resolution of the created image which linearly affects the execution time. The radius of the base of the peak also has a strong impact as for a larger radius more pixels are influenced. As the area of the base is proportional to the squared radius this affects the execution time strongly. The last factor is the number of processed objects n. The execution time is a linear function of n.

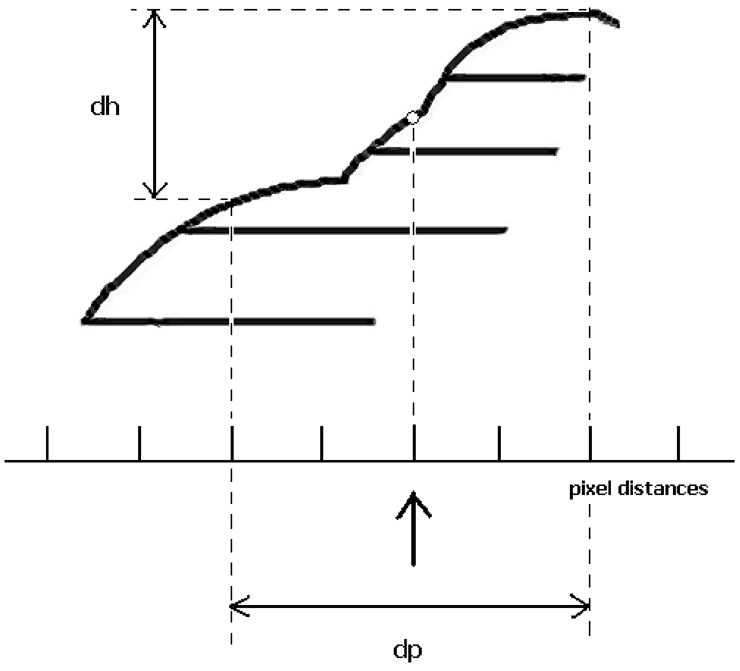


Figure 7.2: Computing the slope for the selected pixel with a pixel window of 4.

Chapter 8

Visualisation Islands Case Study

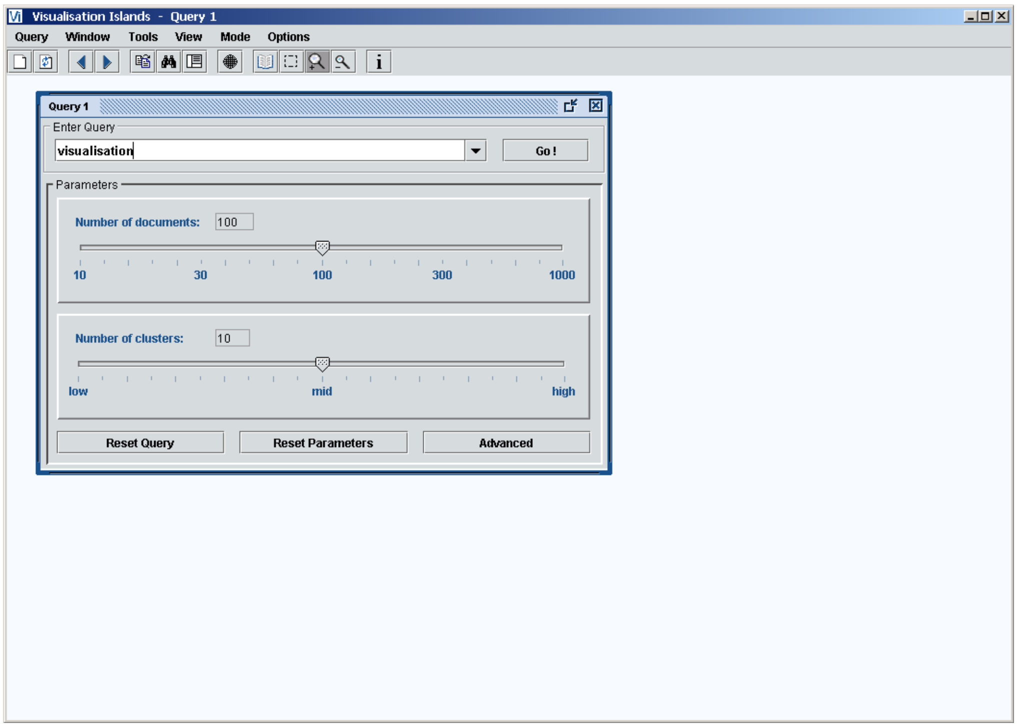
This Chapter demonstrates the use of Visualisation Islands with real world data. Typical user interactions and their results are described and discussed while exploring the retrieved document set. The user guide in Appendix A gives details on how to perform the various operations.

8.1 Typing the Query and Retrieving Results

The search query “visualisation” is entered, see Figure 8.1, and sent to an xFIND search engine. The maximum number of documents is set to 100 and the number of clusters of similar document that should be built is set to 10. The xFIND search engine responds by sending data for 53 documents. These are processed by Visualisation Islands which generates the visualisation shown in the Figure 8.2. The left part of the window shows a list containing similar documents grouped in clusters, displayed in a table of contents style. The right part of the window contains a topographic map where similar documents are placed closed together forming visual clusters which are labeled with keywords of underlaying documents. Areas of the map where the density of documents is high are displayed as islands or mountains. These areas are separated by water denoting areas with little or no documents. Documents are displayed as filled icons while cluster icons are hollow and slightly larger.

By taking a look at the map no area of the map is labeled with “visualization” although this was the query term. There is an island labeled “multimedia”, which could contain documents on visualisation, however it is not particularly compact, what means that these objects are not very similar to each other. This, and the fact that only 53 documents were returned, could be a hint to extend our query with another term. Adding “visualization” to the search query, which is the American spelling of the word, brings the desired improvement. 100 matching documents with the highest estimated relevance are returned by the xFIND search engine. They are organised in 10 clusters and we can get a clue about the content of each cluster by looking at its keywords displayed in the list. However, the topographic map offers a better overview than the table of contents representation of the document set. Looking at Figure 8.3, a high island labeled “visualization” can immediately be located. Its height tells us that there is a large number of documents located in that area. The documents are placed quite tightly to each other, meaning that they are very similar in content. Clearly, these are the documents we are looking for and that is where we will continue our exploration of the search results.

The list representation in Figure 8.3 tells us that the majority of returned objects, 62 of them, are topically similar and that the emphasis of their contents is on visualisation (Cluster 1 keywords: visualization, 3d, environments, visual, ...). This large cluster and the “visualization” peak represent the same documents, which could be confirmed by selecting the objects.



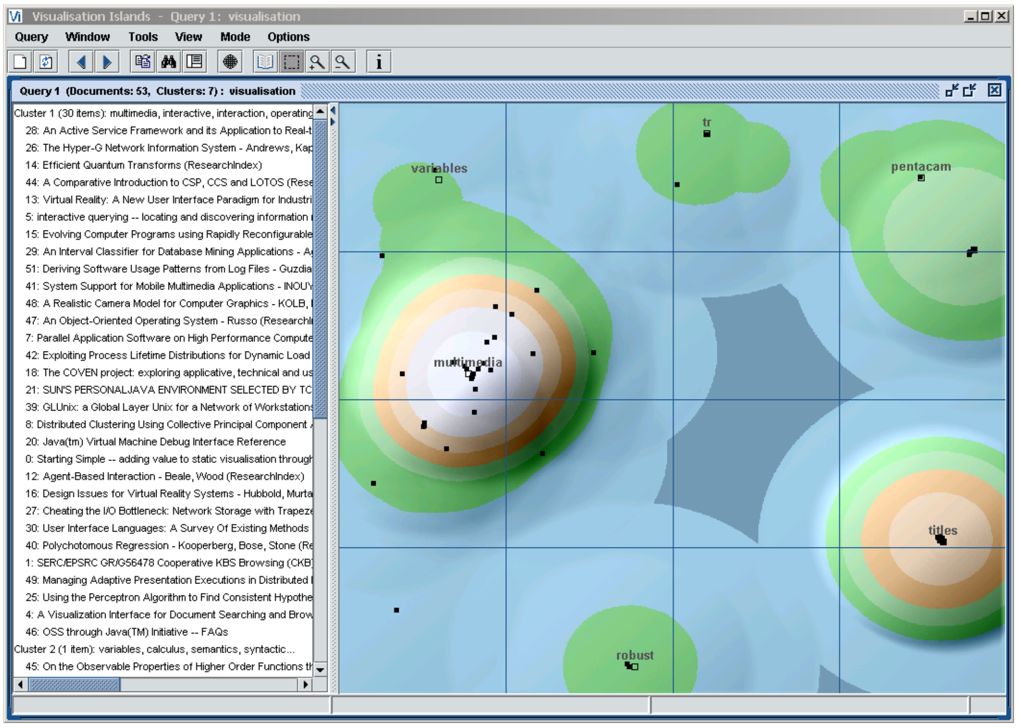
Enter Query: visualisation

Parameters

Number of documents: 100

Number of clusters: 10

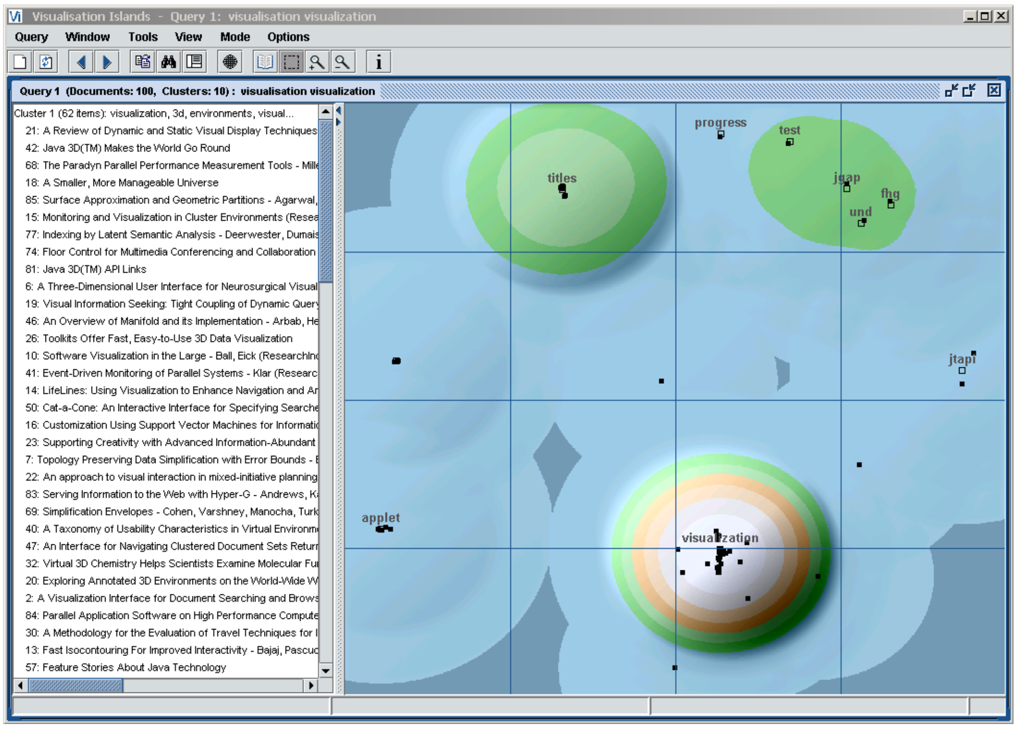
Figure 8.1: Typing in the query: “visualisation”.



Query 1 (Documents: 53, Clusters: 7) : visualisation

Cluster 1 (30 items): multimedia, interactive, interaction, operating
28: An Active Service Framework and its Application to Real-t
26: The Hyper-G Network Information System - Andrews, Kap
14: Efficient Quantum Transforms (ResearchIndex)
44: A Comparative Introduction to CSP, CCS and LOTOS (Rese
13: Virtual Reality: A New User Interface Paradigm for Industri
5: interactive querying -- locating and discovering information
15: Evolving Computer Programs using Rapidly Reconfigurable
29: An Interval Classifier for Database Mining Applications - A
51: Deriving Software Usage Patterns from Log Files - Guzdia
41: System Support for Mobile Multimedia Applications - INOUY
48: A Realistic Camera Model for Computer Graphics - KOLB,
47: An Object-Oriented Operating System - Russo (Researchl
7: Parallel Application Software on High Performance Compute
42: Exploiting Process Lifetime Distributions for Dynamic Load
18: The COVEN project: exploring applicative, technical and us
21: SUN'S PERSONALJAVA ENVIRONMENT SELECTED BY TC
39: GLUnix: a Global Layer Unix for a Network of Workstations
8: Distributed Clustering Using Collective Principal Component
20: Java(tm) Virtual Machine Debug Interface Reference
0: Starting Simple -- adding value to static visualisation through
12: Agent-Based Interaction - Beale, Wood (ResearchIndex)
16: Design Issues for Virtual Reality Systems - Hubbold, Murta
27: Cheating the I/O Bottleneck: Network Storage with Trapeze
30: User Interface Languages: A Survey Of Existing Methods
40: Polychotomous Regression - Kooperberg, Bose, Stone (Re
1: SERC/EPSRC GR/G56478 Cooperative KBS Browsing (CKB)
49: Managing Adaptive Presentation Executions in Distributed
25: Using the Perceptron Algorithm to Find Consistent Hypothe
4: A Visualization Interface for Document Searching and Brow
46: OSS through Java(TM) Initiative -- FAQs
Cluster 2 (1 item): variables, calculus, semantics, syntactic...
45: On the Observable Properties of Higher Order Functions th

Figure 8.2: Results of processing the query: “visualisation”.



Query 1 (Documents: 100, Clusters: 10) : visualisation visualization

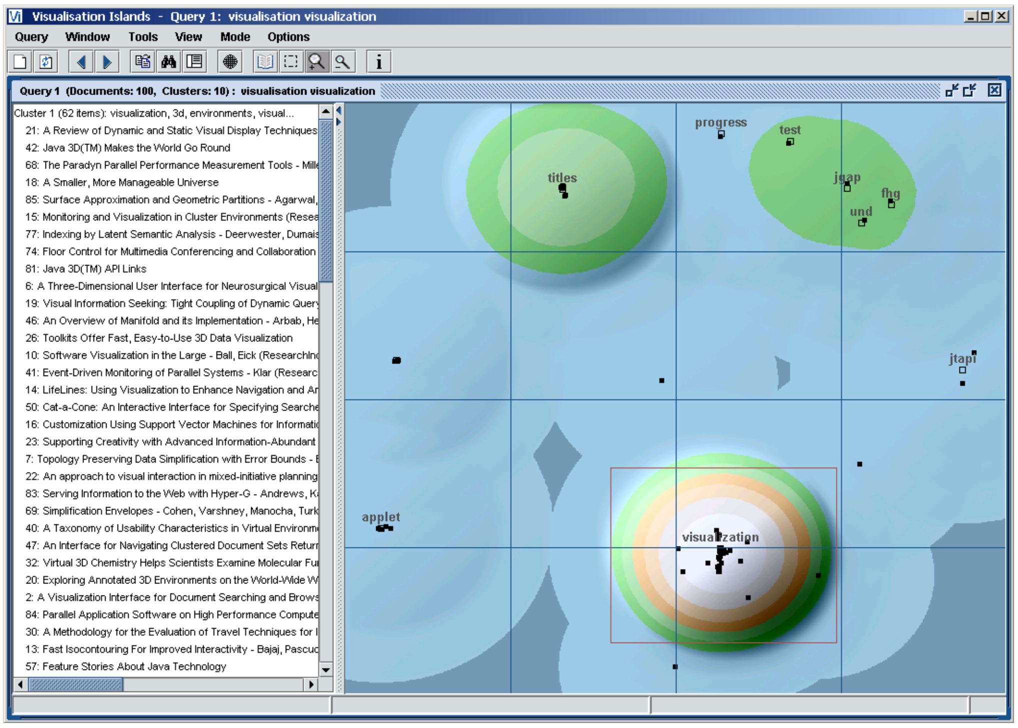
Cluster 1 (62 items): visualization, 3d, environments, visual...
21: A Review of Dynamic and Static Visual Display Techniques
42: Java 3D(TM) Makes the World Go Round
68: The Paradyn Parallel Performance Measurement Tools - Mille
18: A Smaller, More Manageable Universe
85: Surface Approximation and Geometric Partitions - Agarwal,
15: Monitoring and Visualization in Cluster Environments (Resea
77: Indexing by Latent Semantic Analysis - Deerwester, Dumais
74: Floor Control for Multimedia Conferencing and Collaboration
81: Java 3D(TM) API Links
6: A Three-Dimensional User Interface for Neurosurgical Visual
19: Visual Information Seeking: Tight Coupling of Dynamic Query
46: An Overview of Manifold and its Implementation - Arbab, He
26: Toolkits Offer Fast, Easy-to-Use 3D Data Visualization
10: Software Visualization in the Large - Ball, Eick (ResearchInd
41: Event-Driven Monitoring of Parallel Systems - Klar (Researc
14: LifeLines: Using Visualization to Enhance Navigation and Ar
50: Cat-a-Cone: An Interactive Interface for Specifying Searche
16: Customization Using Support Vector Machines for Informatic
23: Supporting Creativity with Advanced Information-Abundant
7: Topology Preserving Data Simplification with Error Bounds - E
22: An approach to visual interaction in mixed-initiative planning
83: Serving Information to the Web with Hyper-G - Andrews, Ka
69: Simplification Envelopes - Cohen, Varshney, Manocha, Turk
40: A Taxonomy of Usability Characteristics in Virtual Environme
47: An Interface for Navigating Clustered Document Sets Return
32: Virtual 3D Chemistry Helps Scientists Examine Molecular Fu
20: Exploring Annotated 3D Environments on the World-Wide W
2: A Visualization Interface for Document Searching and Brows
84: Parallel Application Software on High Performance Compute
30: A Methodology for the Evaluation of Travel Techniques for I
13: Fast Isocontouring For Improved Interactivity - Bajaj, Pascuc
57: Feature Stories About Java Technology

Figure 8.3: Results of the query: “visualisation visualization”.

8.2 Exploring Search Results

Now we wish to focus on the “visualization” island and therefore the part of the map where the island is located is magnified. While in zoom mode the mouse is dragged and a red rectangle is drawn around the area that will be zoomed in (see Figure 8.4). When the mouse button is released the part of the map within the borders of the rectangle is magnified as seen in Figure 8.5. By moving the mouse pointer over document or cluster icons (or over entries in the list) object’s title, URL, and keywords are displayed in the bar at the bottom of the window.

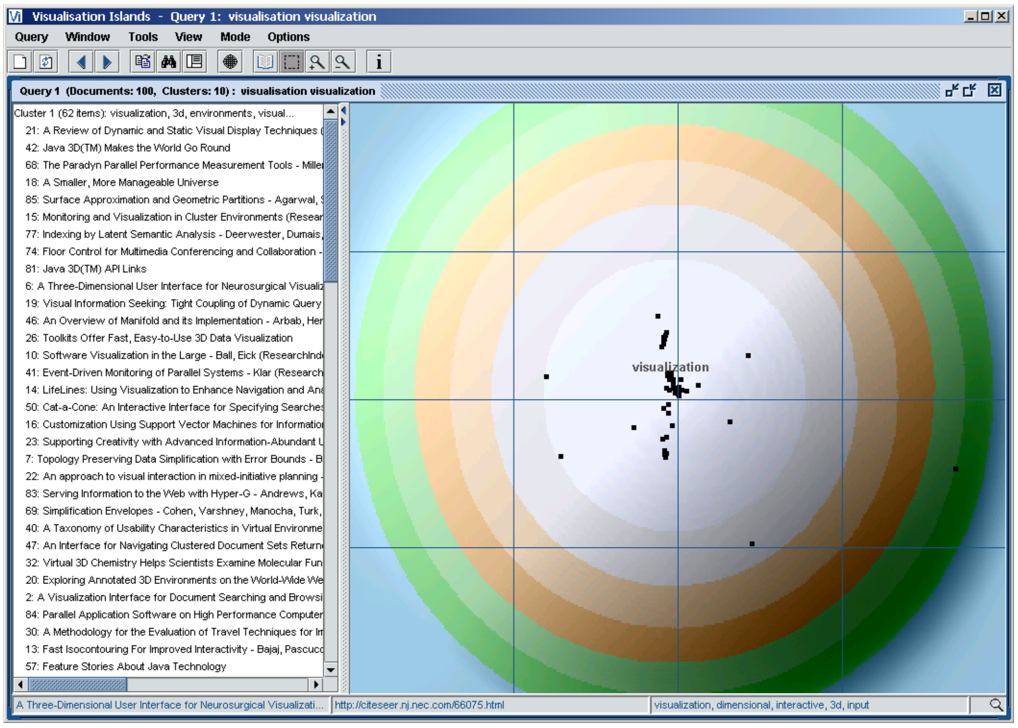
However, we discover that the island is populated with a large number of documents which lie too closely to each other to be separately analysed. This document subset should be reanalysed and organised in a new topical map to give us an overview of the topics covered only in this subset. This can be achieved by entering select mode and dragging the mouse over the documents. A red circle is drawn (see Figure 8.6) and when the mouse button is released all objects inside the circle are selected. Selected objects are highlighted, red in the map and blue-gray in the list. Note that objects, in the map or in the list, can be selected by clicking on them, by using the popup menu activated with the right mouse button, or by double-clicking them to select the whole cluster. Once the documents of interest have been selected the spawn query tool is applied on them. This opens a new pseudo-query window (pseudo because no real search query is sent to xFIND), the documents are processed, and the resulting visualisation is displayed (see Figure 8.8). The new window is named after keywords contained by the processed documents and these keywords are also placed in the query history list as a suggestion for a potential search query on the topic.



Query 1 (Documents: 100, Clusters: 10) : visualisation visualization

Cluster 1 (62 items): visualization, 3d, environments, visual...
21: A Review of Dynamic and Static Visual Display Techniques
42: Java 3D(TM) Makes the World Go Round
68: The Paradyn Parallel Performance Measurement Tools - Mill
18: A Smaller, More Manageable Universe
85: Surface Approximation and Geometric Partitions - Agarwal,
15: Monitoring and Visualization in Cluster Environments (Resea
77: Indexing by Latent Semantic Analysis - Deerwester, Dumais
74: Floor Control for Multimedia Conferencing and Collaboration
81: Java 3D(TM) API Links
6: A Three-Dimensional User Interface for Neurosurgical Visual
19: Visual Information Seeking: Tight Coupling of Dynamic Query
46: An Overview of Manifold and its Implementation - Arbab, He
26: Toolkits Offer Fast, Easy-to-Use 3D Data Visualization
10: Software Visualization in the Large - Ball, Eick (ResearchInd
41: Event-Driven Monitoring of Parallel Systems - Klar (Researc
14: LifeLines: Using Visualization to Enhance Navigation and An
50: Cat-a-Cone: An Interactive Interface for Specifying Searche
16: Customization Using Support Vector Machines for Informatic
23: Supporting Creativity with Advanced Information-Abundant
7: Topology Preserving Data Simplification with Error Bounds - E
22: An approach to visual interaction in mixed-initiative planning
83: Serving Information to the Web with Hyper-G - Andrews, Ka
69: Simplification Envelopes - Cohen, Varshney, Manocha, Turk
40: A Taxonomy of Usability Characteristics in Virtual Environm
47: An Interface for Navigating Clustered Document Sets Return
32: Virtual 3D Chemistry Helps Scientists Examine Molecular Fu
20: Exploring Annotated 3D Environments on the World-Wide W
2: A Visualization Interface for Document Searching and Brows
84: Parallel Application Software on High Performance Compute
30: A Methodology for the Evaluation of Travel Techniques for I
13: Fast Isocontouring For Improved Interactivity - Bajaj, Pascuc
57: Feature Stories About Java Technology

Figure 8.4: Marking a rectengular area around the “visualization” label which shall be zoomed in.

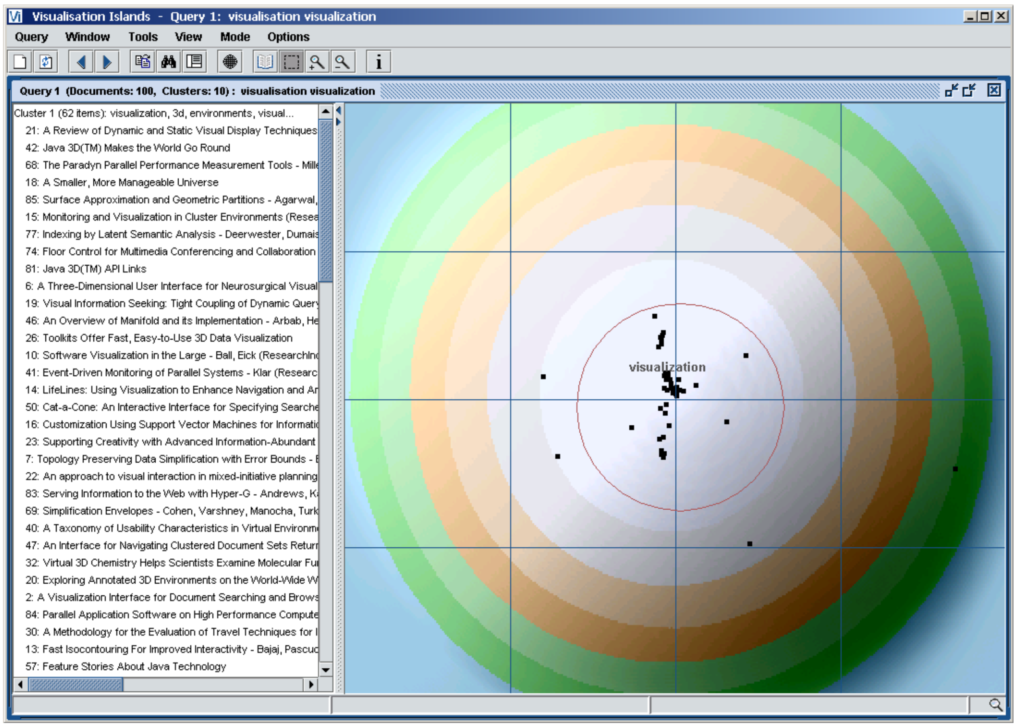

Visualisation Islands - Query 1: visualisation visualization
Query
Window
Tools
View
Mode
Options
visualization

Query 1 (Documents: 100, Clusters: 10) : visualisation visualization

Cluster 1 (62 items): visualization, 3d, environments, visual...
21: A Review of Dynamic and Static Visual Display Techniques (
42: Java 3D(TM) Makes the World Go Round
68: The Paradyn Parallel Performance Measurement Tools - Miller
18: A Smaller, More Manageable Universe
85: Surface Approximation and Geometric Partitions - Agarwal, S
15: Monitoring and Visualization in Cluster Environments (Resear
77: Indexing by Latent Semantic Analysis - Deerwester, Dumais,
74: Floor Control for Multimedia Conferencing and Collaboration -
81: Java 3D(TM) API Links
6: A Three-Dimensional User Interface for Neurosurgical Visualiz
19: Visual Information Seeking: Tight Coupling of Dynamic Query
46: An Overview of Manifold and its Implementation - Arbab, Her
26: Toolkits Offer Fast, Easy-to-Use 3D Data Visualization
10: Software Visualization in the Large - Ball, Eick (ResearchInd
41: Event-Driven Monitoring of Parallel Systems - Klar (Research
14: LifeLines: Using Visualization to Enhance Navigation and Ana
50: Cat-a-Cone: An Interactive Interface for Specifying Searches
16: Customization Using Support Vector Machines for Informatio
23: Supporting Creativity with Advanced Information-Abundant U
7: Topology Preserving Data Simplification with Error Bounds - B
22: An approach to visual interaction in mixed-initiative planning -
83: Serving Information to the Web with Hyper-G - Andrews, Ka
69: Simplification Envelopes - Cohen, Varshney, Manocha, Turk,
40: A Taxonomy of Usability Characteristics in Virtual Environme
47: An Interface for Navigating Clustered Document Sets Return
32: Virtual 3D Chemistry Helps Scientists Examine Molecular Fun
20: Exploring Annotated 3D Environments on the World-Wide We
2: A Visualization Interface for Document Searching and Browsi
84: Parallel Application Software on High Performance Computer
30: A Methodology for the Evaluation of Travel Techniques for Im
13: Fast Isocontouring For Improved Interactivity - Bajaj, Pascucc
57: Feature Stories About Java Technology

A Three-Dimensional User Interface for Neurosurgical Visualizati... | http://citeseer.nj.nec.com/66075.html | visualization, dimensional, interactive, 3d, input

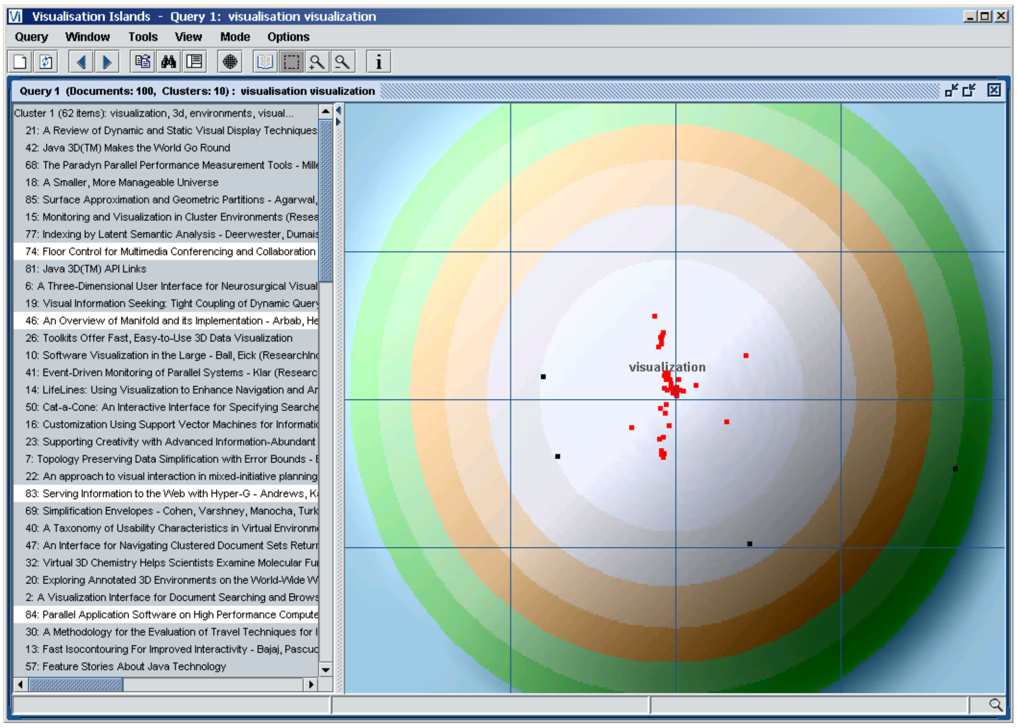
Figure 8.5: The island labeled “visualization’ is zoomed in.


Visualisation Islands - Query 1: visualisation visualization
Query
Window
Tools
View
Mode
Options
visualization

Query 1 (Documents: 100, Clusters: 10) : visualisation visualization

Cluster 1 (62 items): visualization, 3d, environments, visual...
21: A Review of Dynamic and Static Visual Display Techniques
42: Java 3D(TM) Makes the World Go Round
68: The Paradyn Parallel Performance Measurement Tools - Mille
18: A Smaller, More Manageable Universe
85: Surface Approximation and Geometric Partitions - Agarwal,
15: Monitoring and Visualization in Cluster Environments (Resea
77: Indexing by Latent Semantic Analysis - Deerwester, Dumais
74: Floor Control for Multimedia Conferencing and Collaboration
81: Java 3D(TM) API Links
6: A Three-Dimensional User Interface for Neurosurgical Visual
19: Visual Information Seeking: Tight Coupling of Dynamic Query
46: An Overview of Manifold and its Implementation - Arbab, He
26: Toolkits Offer Fast, Easy-to-Use 3D Data Visualization
10: Software Visualization in the Large - Ball, Eick (Researching
41: Event-Driven Monitoring of Parallel Systems - Klar (Researc
14: LifeLines: Using Visualization to Enhance Navigation and Ar
50: Cat-a-Cone: An Interactive Interface for Specifying Searche
16: Customization Using Support Vector Machines for Informatic
23: Supporting Creativity with Advanced Information-Abundant
7: Topology Preserving Data Simplification with Error Bounds - E
22: An approach to visual interaction in mixed-initiative planning
83: Serving Information to the Web with Hyper-G - Andrews, Ka
69: Simplification Envelopes - Cohen, Varshney, Manocha, Turk
40: A Taxonomy of Usability Characteristics in Virtual Environme
47: An Interface for Navigating Clustered Document Sets Return
32: Virtual 3D Chemistry Helps Scientists Examine Molecular Fu
20: Exploring Annotated 3D Environments on the World-Wide W
2: A Visualization Interface for Document Searching and Brows
84: Parallel Application Software on High Performance Compute
30: A Methodology for the Evaluation of Travel Techniques for I
13: Fast Isocontouring For Improved Interactivity - Bajaj, Pascuc
57: Feature Stories About Java Technology

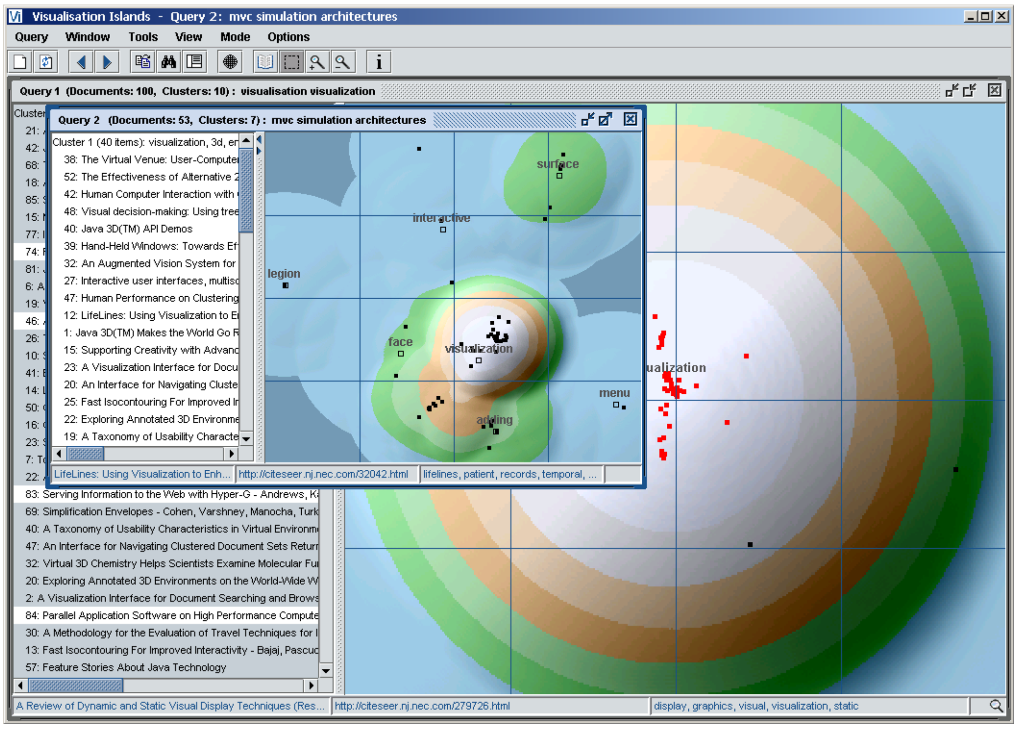
Figure 8.6: Marking a circular area with similar documents which shall be selected.


Visualisation Islands - Query 1: visualisation visualization
Query
Window
Tools
View
Mode
Options
visualization

Query 1 (Documents: 100, Clusters: 10) : visualisation visualization

Cluster 1 (62 items): visualization, 3d, environments, visual...
21: A Review of Dynamic and Static Visual Display Techniques
42: Java 3D(TM) Makes the World Go Round
68: The Paradyn Parallel Performance Measurement Tools - Mille
18: A Smaller, More Manageable Universe
85: Surface Approximation and Geometric Partitions - Agarwal,
15: Monitoring and Visualization in Cluster Environments (Resea
77: Indexing by Latent Semantic Analysis - Deerwester, Dumais
74: Floor Control for Multimedia Conferencing and Collaboration
81: Java 3D(TM) API Links
6: A Three-Dimensional User Interface for Neurosurgical Visual
19: Visual Information Seeking: Tight Coupling of Dynamic Query
46: An Overview of Manifold and its Implementation - Arbab, He
26: Toolkits Offer Fast, Easy-to-Use 3D Data Visualization
10: Software Visualization in the Large - Ball, Eick (Researching
41: Event-Driven Monitoring of Parallel Systems - Klar (Researc
14: LifeLines: Using Visualization to Enhance Navigation and Ar
50: Cat-a-Cone: An Interactive Interface for Specifying Searche
16: Customization Using Support Vector Machines for Informatic
23: Supporting Creativity with Advanced Information-Abundant
7: Topology Preserving Data Simplification with Error Bounds - E
22: An approach to visual interaction in mixed-initiative planning
83: Serving Information to the Web with Hyper-G - Andrews, Ka
69: Simplification Envelopes - Cohen, Varshney, Manocha, Turk
40: A Taxonomy of Usability Characteristics in Virtual Environme
47: An Interface for Navigating Clustered Document Sets Return
32: Virtual 3D Chemistry Helps Scientists Examine Molecular Fu
20: Exploring Annotated 3D Environments on the World-Wide W
2: A Visualization Interface for Document Searching and Brows
84: Parallel Application Software on High Performance Compute
30: A Methodology for the Evaluation of Travel Techniques for I
13: Fast Isocontouring For Improved Interactivity - Bajaj, Pascuc
57: Feature Stories About Java Technology

Figure 8.7: Marked document are selected.



Query 1 (Documents: 100, Clusters: 10) : visualisation visualization

Query 2 (Documents: 53, Clusters: 7) : mvc simulation architectures

Cluster 1 (40 items): visualization, 3d, en
38: The Virtual Venue: User-Compute
52: The Effectiveness of Alternative 2
42: Human Computer Interaction with
48: Visual decision-making: Using tree
40: Java 3D(TM) API Demos
39: Hand-Held Windows: Towards Ef
32: An Augmented Vision System for
27: Interactive user interfaces, multis
47: Human Performance on Clustering
12: LifeLines: Using Visualization to E
1: Java 3D(TM) Makes the World Go R
15: Supporting Creativity with Advanc
23: A Visualization Interface for Docu
20: An Interface for Navigating Cluste
25: Fast Isocontouring For Improved In
22: Exploring Annotated 3D Environme
19: A Taxonomy of Usability Characte

LifeLines: Using Visualization to Enh... | http://citeseer.nj.nec.com/32042.html | lifelines, patient, records, temporal, ...

83: Serving Information to the Web with Hyper-G - Andrews, K
69: Simplification Envelopes - Cohen, Varshney, Manocha, Turk
40: A Taxonomy of Usability Characteristics in Virtual Environm
47: An Interface for Navigating Clustered Document Sets Retur
32: Virtual 3D Chemistry Helps Scientists Examine Molecular Fu
20: Exploring Annotated 3D Environments on the World-Wide W
2: A Visualization Interface for Document Searching and Brows
84: Parallel Application Software on High Performance Compute
30: A Methodology for the Evaluation of Travel Techniques for I
13: Fast Isocontouring For Improved Interactivity - Bajaj, Pascuc
57: Feature Stories About Java Technology

A Review of Dynamic and Static Visual Display Techniques (Res... | http://citeseer.nj.nec.com/279726.html | display, graphics, visual, visualization, static

Figure 8.8: Selected documents are reprocessed allowing a more detailed analysis of their content.

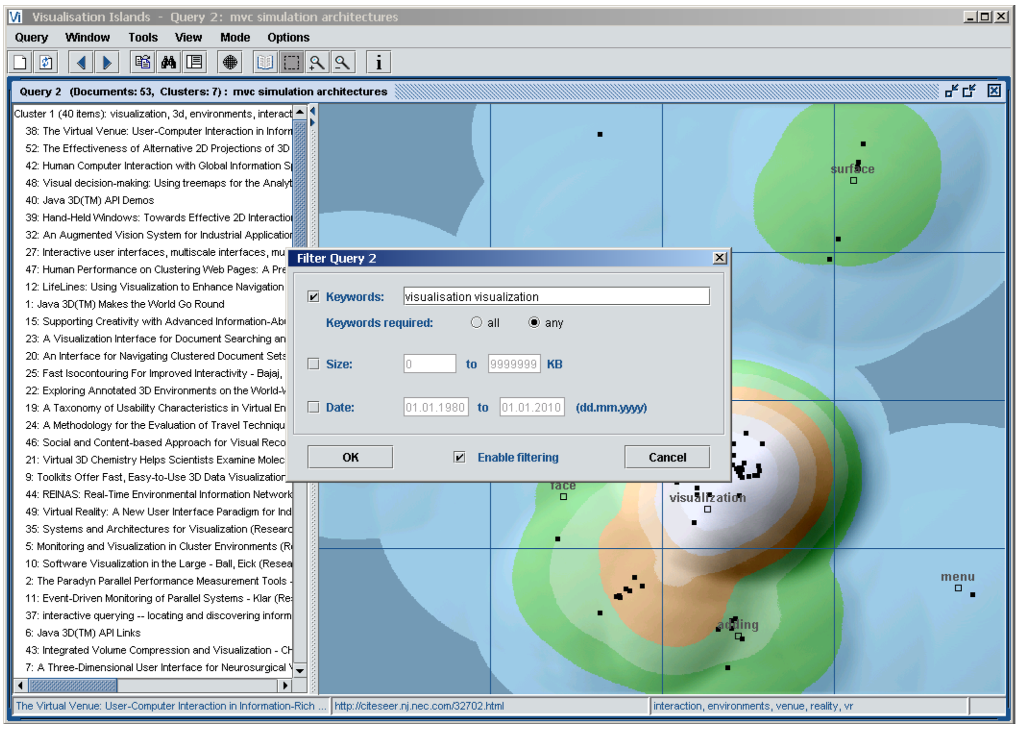
As can be seen in Figure 8.8 the reprocessed documents are well layed out forming distinct structures in the map which can be explored (and reprocessed) further until a document or a reasonably small group of documents is identified which are reliable candidates for relevant documents.

8.3 Identifying Relevant Documents

We can now assume that the subset of documents we are concentrating on is thematically connected to visualisation. To narrow the current set of documents further we can try to filter the set, for example displaying only those objects having “visualisation” or “visualization” as keywords. It is likely that documents which do not have one of these words as keywords are not as relevant as those which have. Typing the words in the filter dialog (see Figure 8.9) and applying the filter brings us to Figure 8.10.

Only a smaller part of the current subset was eliminated strengthening our assumption that we have tracked down and isolated relevant documents. At this point the number of displayed objects is reduced enough that we could start exploring them one by one with the mouse pointer and viewing their title, URL and keywords and perhaps viewing their contents. In view mode every document can be opened in an external viewer application, typically a HTML browser such as Netscape or Internet Explorer, by clicking on it or by invoking the popup menu with the right mouse button.

Another possibility would be to identify documents with some characteristics which are of special interest to us. Visualisation Islands is itself a tool for browsing document collections and searching for relevant documents in search results sets. Let us take a look if there are any documents on browsing and searching document collections in our visualisation document subset.



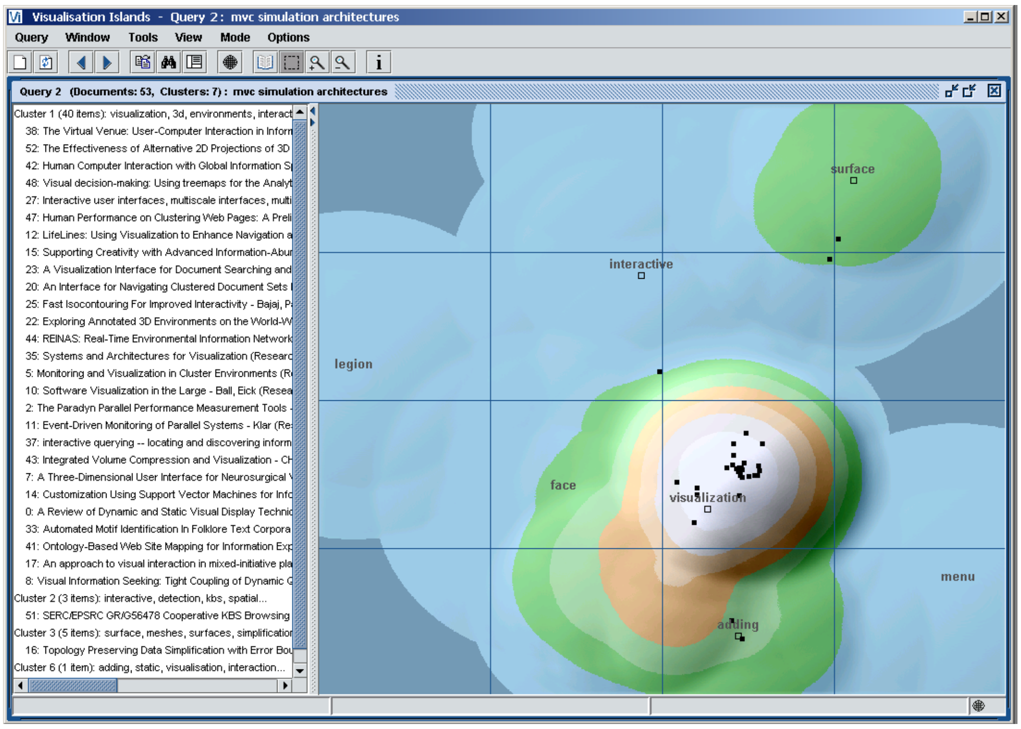
Query 2 (Documents: 53, Clusters: 7) : mvc simulation architectures

Cluster 1 (40 items): visualization, 3d, environments, interact
38: The Virtual Venue: User-Computer Interaction in Inforn
52: The Effectiveness of Alternative 2D Projections of 3D
42: Human Computer Interaction with Global Information S
48: Visual decision-making: Using treemaps for the Analyt
40: Java 3D(TM) API Demos
39: Hand-Held Windows: Towards Effective 2D Interactio
32: An Augmented Vision System for Industrial Applicatior
27: Interactive user interfaces, multiscale interfaces, mu
47: Human Performance on Clustering Web Pages: A Pre
12: LifeLines: Using Visualization to Enhance Navigation
1: Java 3D(TM) Makes the World Go Round
15: Supporting Creativity with Advanced Information-Abu
23: A Visualization Interface for Document Searching an
20: An Interface for Navigating Clustered Document Sets
25: Fast Isocontouring For Improved Interactivity - Bajaj,
22: Exploring Annotated 3D Environments on the World-W
19: A Taxonomy of Usability Characteristics in Virtual En
24: A Methodology for the Evaluation of Travel Techniqu
46: Social and Content-based Approach for Visual Reco
21: Virtual 3D Chemistry Helps Scientists Examine Molec
9: Toolkits Offer Fast, Easy-to-Use 3D Data Visualizatior
44: REINAS: Real-Time Environmental Information Network
49: Virtual Reality: A New User Interface Paradigm for Ind
35: Systems and Architectures for Visualization (Researc
5: Monitoring and Visualization in Cluster Environments (R
10: Software Visualization in the Large - Ball, Eick (Resea
2: The Paradyn Parallel Performance Measurement Tools -
11: Event-Driven Monitoring of Parallel Systems - Klar (Re
37: interactive querying -- locating and discovering inform
6: Java 3D(TM) API Links
43: Integrated Volume Compression and Visualization - CH
7: A Three-Dimensional User Interface for Neurosurgical V

Keywords: [✓] visualisation visualization
Keywords required: all [✓ any]
Size: [ ] 0 to 9999999 KB
Date: [ ] 01.01.1980 to 01.01.2010 (dd.mm.yyyy)
[✓] Enable filtering

The Virtual Venue: User-Computer Interaction in Information-Rich ... | http://citeseer.nj.nec.com/32702.html | interaction, environments, venue, reality, vr

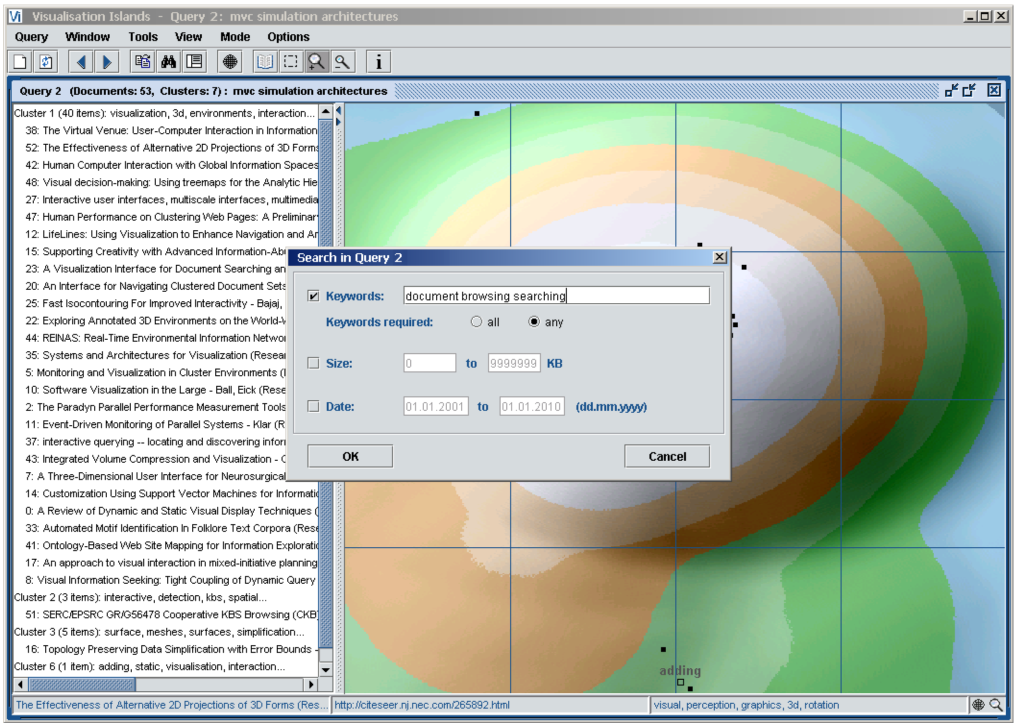
Figure 8.9: Filter dialog over the window with reprocessed documents.



Query 2 (Documents: 53, Clusters: 7) : mvc simulation architectures

Cluster 1 (40 items): visualization, 3d, environments, interact
38: The Virtual Venue: User-Computer Interaction in Inforn
52: The Effectiveness of Alternative 2D Projections of 3D
42: Human Computer Interaction with Global Information S
48: Visual decision-making: Using treemaps for the Analyt
27: Interactive user interfaces, multiscale interfaces, multi
47: Human Performance on Clustering Web Pages: A Preli
12: LifeLines: Using Visualization to Enhance Navigation a
15: Supporting Creativity with Advanced Information-Abur
23: A Visualization Interface for Document Searching and
20: An Interface for Navigating Clustered Document Sets
25: Fast Isocontouring For Improved Interactivity - Bajaj, P
22: Exploring Annotated 3D Environments on the World-W
44: REINAS: Real-Time Environmental Information Network
35: Systems and Architectures for Visualization (Researc
5: Monitoring and Visualization in Cluster Environments (R
10: Software Visualization in the Large - Ball, Eick (Resea
2: The Paradyn Parallel Performance Measurement Tools -
11: Event-Driven Monitoring of Parallel Systems - Klar (Re
37: interactive querying -- locating and discovering inform
43: Integrated Volume Compression and Visualization - CH
7: A Three-Dimensional User Interface for Neurosurgical V
14: Customization Using Support Vector Machines for Info
0: A Review of Dynamic and Static Visual Display Technic
33: Automated Motif Identification In Folklore Text Corpora
41: Ontology-Based Web Site Mapping for Information Exp
17: An approach to visual interaction in mixed-initiative pla
8: Visual Information Seeking: Tight Coupling of Dynamic G
Cluster 2 (3 items): interactive, detection, kbs, spatial...
51: SERC/EPSRC GR/G56478 Cooperative KBS Browsing
Cluster 3 (5 items): surface, meshes, surfaces, simplificatior
16: Topology Preserving Data Simplification with Error Bou
Cluster 6 (1 item): adding, static, visualisation, interaction...

Figure 8.10: Objects not satisfying the filter condition are hidden.


Visualisation Islands - Query 2: mvc simulation architectures
Query
Window
Tools
View
Mode
Options
adding
OK
Cancel

Query 2 (Documents: 53, Clusters: 7): mvc simulation architectures

Cluster 1 (40 items): visualization, 3d, environments, interaction...
38: The Virtual Venue: User-Computer Interaction in Information
52: The Effectiveness of Alternative 2D Projections of 3D Form
42: Human Computer Interaction with Global Information Spaces
48: Visual decision-making: Using treemaps for the Analytic Hie
27: Interactive user interfaces, multiscale interfaces, multimedia
47: Human Performance on Clustering Web Pages: A Preliminar
12: LifeLines: Using Visualization to Enhance Navigation and Ar
15: Supporting Creativity with Advanced Information-Abu
23: A Visualization Interface for Document Searching an
20: An Interface for Navigating Clustered Document Sets
25: Fast Isocontouring For Improved Interactivity - Bajaj,
22: Exploring Annotated 3D Environments on the World-W
44: REINAS: Real-Time Environmental Information Netwo
35: Systems and Architectures for Visualization (Resea
5: Monitoring and Visualization in Cluster Environments (
10: Software Visualization in the Large - Ball, Eick (Rese
2: The Paradyn Parallel Performance Measurement Tools
11: Event-Driven Monitoring of Parallel Systems - Klar (R
37: interactive querying -- locating and discovering infor
43: Integrated Volume Compression and Visualization - C
7: A Three-Dimensional User Interface for Neurosurgical
14: Customization Using Support Vector Machines for Informati
0: A Review of Dynamic and Static Visual Display Techniques (
33: Automated Motif Identification In Folklore Text Corpora (Res
41: Ontology-Based Web Site Mapping for Information Explorati
17: An approach to visual interaction in mixed-initiative planning
8: Visual Information Seeking: Tight Coupling of Dynamic Query
Cluster 2 (3 items): interactive, detection, kbs, spatial...
51: SERC/EPSRC GR/G56478 Cooperative KBS Browsing (CKB
Cluster 3 (5 items): surface, meshes, surfaces, simplification...
16: Topology Preserving Data Simplification with Error Bounds
Cluster 6 (1 item): adding, static, visualisation, interaction...

Search in Query 2

[✓] Keywords: document browsing searching

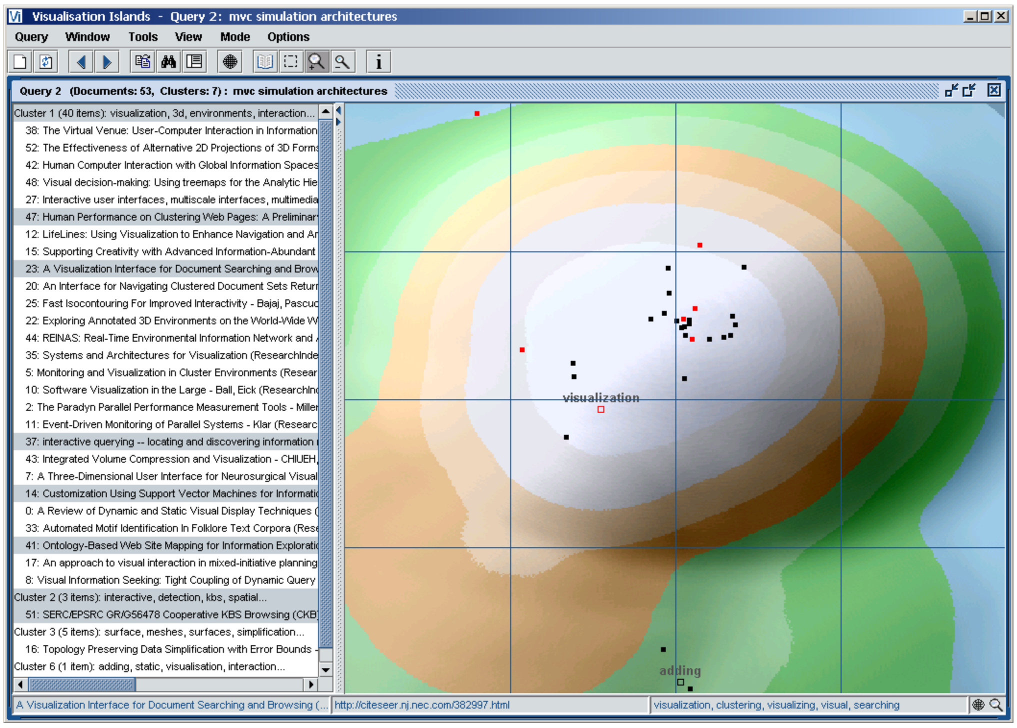
Keywords required: ( ) all [✓ any]

[ ] Size: 0 to 9999999 KB

[ ] Date: 01.01.2001 to 01.01.2010 (dd.mm.yyyy)

The Effectiveness of Alternative 2D Projections of 3D Forms (Res... | http://citeseer.nj.nec.com/265892.html | visual, perception, graphics, 3d, rotation

Figure 8.11: Search dialog.

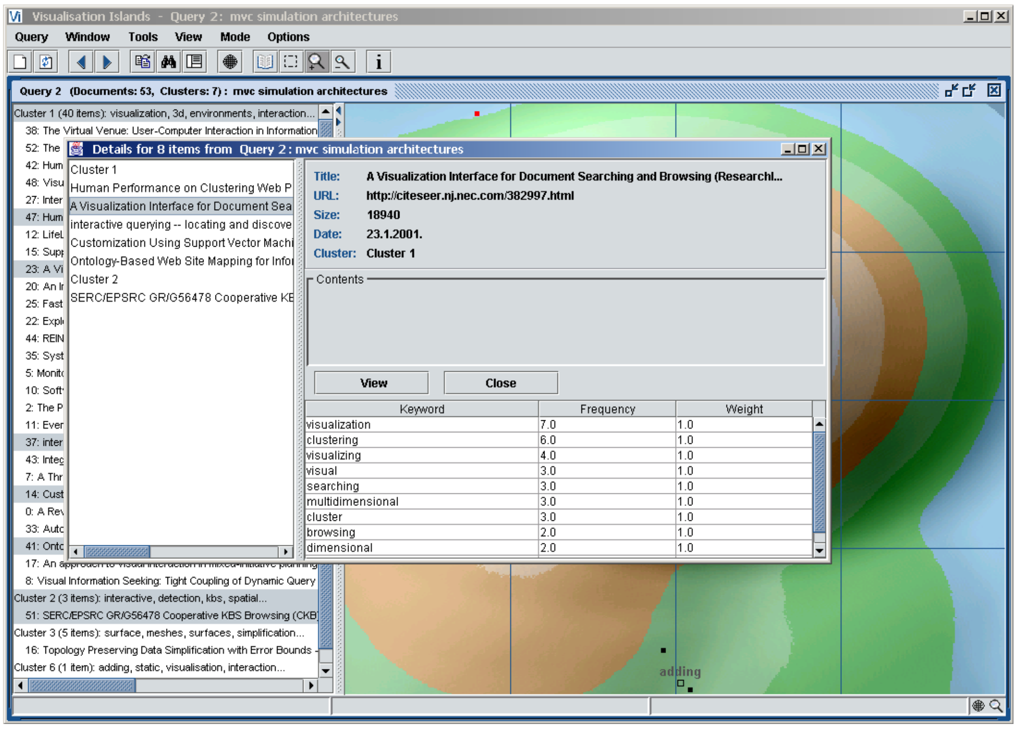

Visualisation Islands - Query 2: mvc simulation architectures
Query
Window
Tools
View
Mode
Options
visualization
adding

Query 2 (Documents: 53, Clusters: 7): mvc simulation architectures

Cluster 1 (40 items): visualization, 3d, environments, interaction...
38: The Virtual Venue: User-Computer Interaction in Information
52: The Effectiveness of Alternative 2D Projections of 3D Form
42: Human Computer Interaction with Global Information Spaces
48: Visual decision-making: Using treemaps for the Analytic Hie
27: Interactive user interfaces, multiscale interfaces, multimedia
47: Human Performance on Clustering Web Pages: A Preliminar
12: LifeLines: Using Visualization to Enhance Navigation and Ar
15: Supporting Creativity with Advanced Information-Abundant
23: A Visualization Interface for Document Searching and Brow
20: An Interface for Navigating Clustered Document Sets Retur
25: Fast Isocontouring For Improved Interactivity - Bajaj, Pascuc
22: Exploring Annotated 3D Environments on the World-Wide W
44: REINAS: Real-Time Environmental Information Network and
35: Systems and Architectures for Visualization (Researchinde
5: Monitoring and Visualization in Cluster Environments (Resear
10: Software Visualization in the Large - Ball, Eick (Researchin
2: The Paradyn Parallel Performance Measurement Tools - Miller
11: Event-Driven Monitoring of Parallel Systems - Klar (Researc
37: interactive querying -- locating and discovering information
43: Integrated Volume Compression and Visualization - CHIUEH,
7: A Three-Dimensional User Interface for Neurosurgical Visual
14: Customization Using Support Vector Machines for Informati
0: A Review of Dynamic and Static Visual Display Techniques (
33: Automated Motif Identification In Folklore Text Corpora (Rese
41: Ontology-Based Web Site Mapping for Information Explorati
17: An approach to visual interaction in mixed-initiative planning
8: Visual Information Seeking: Tight Coupling of Dynamic Query
Cluster 2 (3 items): interactive, detection, kbs, spatial...
51: SERC/EPSRC GR/G56478 Cooperative KBS Browsing (CKB
Cluster 3 (5 items): surface, meshes, surfaces, simplification...
16: Topology Preserving Data Simplification with Error Bounds -
Cluster 6 (1 item): adding, static, visualisation, interaction...

A Visualization Interface for Document Searching and Browsing (... | http://citeseer.nj.nec.com/382997.html | visualization, clustering, visualizing, visual, searching

Figure 8.12: Found objects are selected. All documents can be pointed at with the mouse.


Visualisation Islands - Query 2: mvc simulation architectures
Query
Window
Tools
View
Mode
Options
View
Close
adding

Query 2 (Documents: 53, Clusters: 7) : mvc simulation architectures

Cluster 1 (40 items): visualization, 3d, environments, interaction...
38: The Virtual Venue: User-Computer Interaction in Information
52: The
42: Hum
48: Visu
27: Inter
47: Hum
12: LifeL
15: Supp
23: A Vi
20: An Ir
25: Fast
22: Expl
44: REIN
35: Syst
5: Monit
10: Soft
2: The P
11: Ever
37: inter
43: Integ
7: A Thr
14: Cust
0: A Rev
33: Autc
41: Ontc
17: An approach to visual interaction in mixed-initiative planning
8: Visual Information Seeking: Tight Coupling of Dynamic Query
Cluster 2 (3 items): interactive, detection, kbs, spatial...
51: SERC/EPSRC GR/G56478 Cooperative KBS Browsing (CKB
Cluster 3 (5 items): surface, meshes, surfaces, simplification...
16: Topology Preserving Data Simplification with Error Bounds -
Cluster 6 (1 item): adding, static, visualisation, interaction...

Details for 8 items from Query 2 : mvc simulation architectures

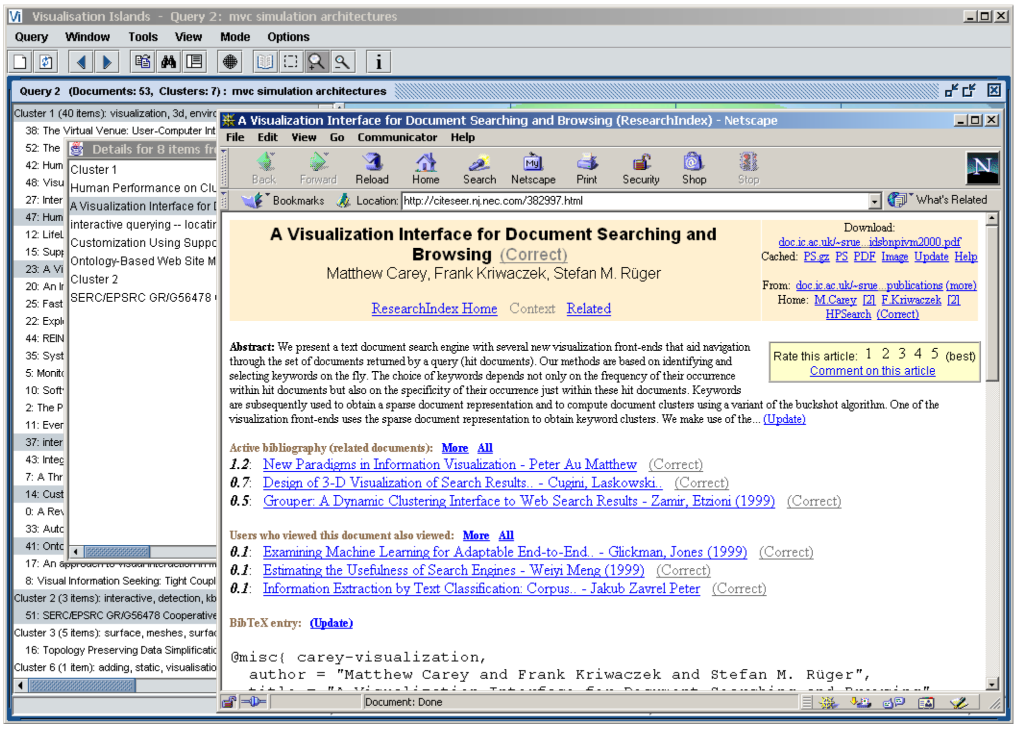
Cluster 1
Human Performance on Clustering Web P
A Visualization Interface for Document Sea
interactive querying -- locating and discove
Customization Using Support Vector Machi
Ontology-Based Web Site Mapping for Infor
Cluster 2
SERC/EPSRC GR/G56478 Cooperative KE

Title:	A Visualization Interface for Document Searching and Browsing (Researchl...
URL:	http://citeseer.nj.nec.com/382997.html
Size:	18940
Date:	23.1.2001.
Cluster:	Cluster 1

Contents

Keyword	Frequency	Weight
visualization	7.0	1.0
clustering	6.0	1.0
visualizing	4.0	1.0
visual	3.0	1.0
searching	3.0	1.0
multidimensional	3.0	1.0
cluster	3.0	1.0
browsing	2.0	1.0
dimensional	2.0	1.0

Figure 8.13: Selected objects can be examined in the details window. An interesting document is located.



A Visualization Interface for Document Searching and Browsing (ResearchIndex) - Netscape

Location: http://citeseer.nj.nec.com/382997.html

A Visualization Interface for Document Searching and Browsing (Correct)
Matthew Carey, Frank Kriwaczek, Stefan M. Rüger

ResearchIndex Home Context Related

Download: doc.ic.ac.uk/~srue...idsbnpivm2000.pdf
Cached: PS.gz PS PDF Image Update Help

From: doc.ic.ac.uk/~srue...publications (more)
Home: M.Carey [2] F.Kriwaczek [2]
HPSearch (Correct)

Abstract: We present a text document search engine with several new visualization front-ends that aid navigation through the set of documents returned by a query (hit documents). Our methods are based on identifying and selecting keywords on the fly. The choice of keywords depends not only on the frequency of their occurrence within hit documents but also on the specificity of their occurrence just within these hit documents. Keywords are subsequently used to obtain a sparse document representation and to compute document clusters using a variant of the buckshot algorithm. One of the visualization front-ends uses the sparse document representation to obtain keyword clusters. We make use of the... (Update)

Rate this article: 1 2 3 4 5 (best)
Comment on this article

Active bibliography (related documents): More All
1.2: New Paradigms in Information Visualization - Peter Au Matthew (Correct)
0.7: Design of 3-D Visualization of Search Results.. - Cugini, Laskowski.. (Correct)
0.5: Grouper: A Dynamic Clustering Interface to Web Search Results - Zamir, Etzioni (1999) (Correct)

Users who viewed this document also viewed: More All
0.1: Examining Machine Learning for Adaptable End-to-End.. - Glickman, Jones (1999) (Correct)
0.1: Estimating the Usefulness of Search Engines - Weiyi Meng (1999) (Correct)
0.1: Information Extraction by Text Classification: Corpus.. - Jakub Zavrel Peter (Correct)

BibTeX entry: (Update)

```
@misc{ carey-visualization,
  author = "Matthew Carey and Frank Kriwaczek and Stefan M. Rüger",
```

Cluster 1 (40 items): visualization, 3d, envirc
38: The Virtual Venue: User-Computer Int
Cluster 2 (3 items): interactive, detection, kb
51: SERC/EPSRC GR/G56478 Cooperative
Cluster 3 (5 items): surface, meshes, surfac
16: Topology Preserving Data Simplificatic
Cluster 6 (1 item): adding, static, visualisatio

Figure 8.14: Chosen document is opened in an external browser.

8.4 Finding the Right Document

Now that we have a collection of documents which are relevant to the search query we cant attempt to find documents on browsing and searching document collections, among these. By typing “document browsing searching” in the search dialog (see Figure 8.11) and performing a search on the current document set, matching documents are selected as shown in Figure 8.12. We can take a more detailed look by viewing the data of found documents in the detail window (see Figure 8.13). On choosing an object all available data on it is displayed, its title, URL, size, creation date, and its keyword profile, making it an easy task to find one fitting our interests. Viewing a document is a matter of one mouse click, Figure 8.14 shows the chosen document in Netscape.

Chapter 9

Future Work

Visualisation Islands version 1.0 is the first implementation of an xFIND client for organising and visualising search results according to their content. Although the system in its present state appears quite complete and is sufficiently stable there are numerous improvements which the author would like to propose for possible future versions. These are aimed at improving the quality of results as well as improving performance with the goal of processing a larger number of objects than is currently possible. Implementing improved algorithms paired with a more usable user interface and possibly extending the system to handle other data sources would make Visualisation Islands far more attractive. In particular future extensions might include:

- Loading and pre-processing: The algorithms implemented in Visualisation Islands could be applied on search results of other search engines, or on any document collection, if a a pre-processing algorithm to analyse documents and transform them into vector representation were implemented.
- Clustering algorithms: Improvement of the k-means algorithm and performance of the hierarchical agglomerative algorithm should be considered.
  - Improving the k-means algorithm by implementing better initial partition generators. This would improve the quality of clustering results greatly, which is important if larger number of documents should be processed, since in that case hierarchical agglomerative clustering becomes too slow. However, the behaviour and quality of such algorithms is relatively unexplored. Currently a random initial partition or the partition produced by single pass clustering algorithm are used.
  - Optimisation of the hierarchical agglomerative clustering algorithm. The current implementation has $O(n^3)$ time complexity in the worst case. Is is possible to reach $O(n^2 log(n))$ time complexity with more aggressive optimisation.
- Force-directed placement: This part of the program offers many possibilities for improvement and further research, both in terms of performance and layout quality.
  - Investigation of other force models with the goal of achieving better discrimination between objects and a higher degree of stability of the simulated system.
  - Computing mechanical stress is computationally intensive and the level of stress reached depends on the processed object set and the initial configuration. Therefore, investigation of less computationally intensive, adaptive termination conditions is recommended. Decreasing of average object movement or average force could be used for this purpose.

  - The current implementation has an $O(n^2)$ execution time and is primarily limited by the need to compute all inter-object similarity coefficients and store them in a similarity matrix for later use. Removing this obstacle should be one of the primary future goals if large number of objects need to be processed in short time. A possible solution would be to use clustering results in the force-directed placement algorithm and compare each document to all cluster centroids and to documents from the same cluster only.

- Map generator: The map generator algorithm could be adapted to produce backgrounds other than the currently implemented island map. Mountains landscapes (Alps) or deep space backgrounds could be implemented by using different peak forms and other colour palettes.

- User interface: Currently little is known about the usability of Visualisation Islands. Performing a usability study would surely be of great significance for discovering problems and drawbacks of the user interface, and would provide fresh ideas how to improve the system. Some suggestions are:

  - Table of contents should be displayed in the form of a tree in addition to the currently implemented list view.
  - The saving of results could be added, giving the possibility to exchange the maps between users or to explore them gradually. This would be an important feature, because the implemented algorithms are not deterministic. Therefore, each time the algorithms are applied a map with a different appearance is computed for the same document set.
  - Possibility of bookmarking interesting documents and clusters should be implemented.

Chapter 10

Concluding Remarks

This thesis discussed the problem of retrieving relevant information. Very large amounts of information are available electronically over the Internet. Searching for a specific topic often results in receiving a huge number of search results in unstructured form. Standard search engines return search query results as a linear list of hits sorted by estimated relevance, but no relations between returned documents are given and it is difficult for the user to obtain an overview and to tell which of the retrieved information is of interest and which is not.

Visualisation Islands was implemented as an attempt to address these problems by organising and visualising topically related documents. It is an xFIND search engine client which processes and organizes search query results depending on their content and similarity, and visualises the results in the form of an interactive, intuitive, topically organised topographic map. Relationships between documents are displayed by proximity: topically similar documents are grouped together to form mountains which are separated by lower areas or water containing a smaller number of less similar objects. Visualisation Islands is a modularly designed software package, written in Java for inter-platform portability. The system can process multiple sets of search results simultaneously and allows the reprocessing of any subset of the returned documents for more detailed insight.

Once the user has typed in the search query the loading phase begins. The data for matching documents is returned, including the title, URL, the size and the creation date, as well as a number of keywords and their frequencies, describing the documents' content. Visualisation Islands uses the extracted keywords to build a vector representation of the documents in a high dimensional space, allowing the definition of a similarity measure between every pair of vectors and thus giving a possibility to compare returned documents.

In the first processing step, clusters of similar documents are formed using standard clustering algorithms such as hierarchical agglomerative clustering, the k-means algorithm, or the single pass algorithm with k-means refinement. The obtained structure is used to display the documents in a table of contents style.

In the second processing step, a force-directed placement algorithm based on stochastic sampling is used to map the documents from the high-dimensional vector space to a 2-D plane preserving their relative distances so far as possible. A force is assigned to every pair of objects according to their similarity and 2-D distance: an attractive force for similar and a repulsive force for dissimilar documents. After a certain number of iterations is performed a 2-D layout is obtained where similar documents are grouped together, and less similar documents are pushed apart. Instead of comparing every pair of documents every iteration, stochastic sampling is used to improve performance.

In the third processing step, a 2-D or 3-D topographic background is generated based on the 2-D document coordinates computed in the previous step. Mountains are formed at areas where document density is high. These areas are divided by areas with smaller document density which are represented as water. As a result islands of topically similar documents are formed. By labeling the mountains with keywords representing the underlaying documents and displaying the documents' icons atop the generated background an intuitive landscape (or seascape) is created.

Having retrieved documents organised in a table of contents together with the landscape of islands of topically similar documents labeled with their keywords, makes it easy for the user to instantaneously locate areas of interest and discard documents or document groups covering less interesting topics. An intuitive user interface provides a means for interactively exploring the map by zooming, searching, filtering, selecting, viewing document details, reading contents of chosen document, and analysing the returned document sets in more detail.

This thesis is organised in two parts: a theoretical part composed of Chapters 2 to 5, and a Visualisation Islands part consisting of Chapters 6 to 10 and Appendices A and B. In Chapter 2 Information Retrieval (IR) was covered, describing techniques such as indexing of documents and the vector space model. Chapter 3 describes clustering methods. A number of partitional, hierarchical and artificial intelligence methods were discussed and compared. Chapter 4 gives an overview of general visualisation principles. A series of document visualisation systems were presented to illustrate how visualisation can be applied to improve the searching for and presentation of document sets. In Chapter 5 xFIND, a scaleable, distributed search engine was described. xFIND supplies Visualisation Islands with documents in response to a search query.

The theory is the foundation on which Visualisation Islands was built. The software was introduced in Chapter 6 where the object-oriented architecture of the program was described and responsibilities and interactions of all packages and classes were specified. In Chapter 7 the algorithms implemented in Visualisation Islands were described: the clustering of the retrieved documents using single-pass, k-means or hierarchical agglomerative clustering algorithms, mapping the documents from the high-dimensional term space to the 2-D viewport space using a force-directed placement algorithm, and generation of a 2-D or 3-D style map background image. Visualisation Island was demonstrated in practice on a number of examples in Chapter 8 and finally some ideas for future research end program improvements were presented in Chapter 9. The user guide can be found in Appendix A, with the exception of advanced parameters which are described in Appendix B.

Appendix A

Visualisation Islands User Guide

A.1 Introduction to Visualisation Islands

A standard search engine returns search query results in a form of a linear list of hits sorted by estimated relevance, but no relations between returned documents are given. Visualisation Islands is an xFIND search engine client that processes and organises search query results depending on their content and similarity, and visualises the results in form of an interactive, intuitive, topically organised map. Similar documents are grouped together to form mountains which are divided by lower areas or see containing a smaller number of less similar objects. The user is offered a 2-D or 3-D style landscape (or seascape) with islands of topically similar documents labeled with their keywords. This makes it easy to instantaneously locate areas of interest and discard documents or document groups covering less interesting topics. It is possible to interactively explore the map by zooming, searching, filtering, selecting, viewing details and reading contents of chosen documents as well as to reprocess any subset of the returned documents for a more detailed insight. Also a "table of contents" style organisation of the retrieved documents is offered. Visualisation Islands is able to process and handle several queries simultaneously.

A.2 Running the Program

Visualisation Islands was written in Java and can be run on almost any machine or operating system, provided a Java virtual machine is available and installed on the system. Visualisation Islands requires Java2 runtime environment (JRE) version 1.2.0 or higher, or Java2 software development kit (SDK) version 1.2.0 or higher (JRE is part of the SDK). For Windows and Linux on x86 machines and for SPARC/Solaris platform the latest version of Java can be downloaded from http://java.sun.com .

If the Java2 JRE is installed on the system Visualisation Islands can be started with one of the following commands:

- java -jar -Xmx128m VisIslands.jar
- java -Xmx128m visislands.VisIslands

The -Xmx128m option is not required but it is strongly recommended as the virtual machine default maximum memory setting of 64MB might not be enough for handling large queries or more queries at once. Under Windows a double click on VisIslands.jar will also start the program, but the -Xmx option can not be set directly in this case.

A.3 Tool Box

Most important program functions can be accessed through buttons in the tool bar. If no query window is active some buttons will be disabled. The buttons have following functions, from left to right, as displayed in Figure A.1:

- New query. Opens a new query window in query input mode where you can enter the query text and set the query parameters.
- Modify query. By pressing this button you will be able to modify the query text and parameters of a loaded and processed query for the active query window. You will have to reload and reprocess the query for changes to take effect except for some display related advanced parameters (see Appendix B).
- Previous query. Visualisation Islands allows you to process and keep open more than one query at a time. This button will make the previous query active and push its query window to front.
- Next query. Makes the next query active and push its query window to front.
- Spawn query. Launches a pseudo-query by processing the selected objects of the current query and displaying them in a new query window. The pseudo-query will be named after keywords with highest importance (product of keyword's frequency and weight), and its title will be placed in the query history list. This allows you to refine the results and explore them in more detail.
- Search. Invokes the search dialog giving you the possibility to search for objects from the active query window with certain features. The object found are selected (highlighted).
- Details. Used to display all selected objects of the active query in the details window giving you the possibility to view information such as title, URL, size, creation date and keywords profiles of each object.
- Filter. Invokes the filter dialog, which is very similar to the search dialog. The difference is that only objects satisfying the filter condition will remain visible, all other objects are hidden.
- View mode. Switches to view mode. In view mode, clicking on a document displayed in the map or in the list will show its contents in an external viewer application (usually a browser).
- Select mode. Switches to select mode. In select mode, the mouse is used to select and deselect objects by clicking on them for single selection, or by dragging the mouse for multiple selection.
- Zoom mode. Switches to zoom mode. In zoom mode the mouse can be dragged to select a rectangular map area which is zoomed in when the mouse button is released.
- Zoom out. Will zoom out one step backwards in the zoom hierarchy.
- Info. Invokes the info dialog displaying program, author, author's advisors and licence information.

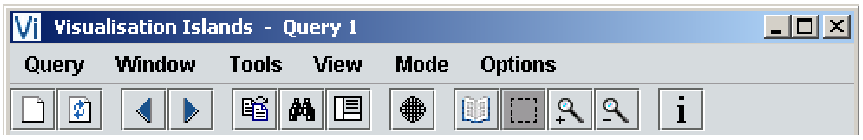


Figure A.1: VisIslands tool box.

A.4 Pull Down Menus

While the tool box gives access to the most often used program functions the pull down menus extend this functionality with a number of additional functions. These functions are ordered into six pull down menus.

A.4.1 Query Menu

Query menu, displayed in Figure A.2 offers basic query operations:

- New creates a new, unloaded query. The query window is in input mode allowing you to input the query text and set the parameters.
- Modify. Allows modification of the query text and the parameters of a loaded query. You will have to reload and reprocess the query for the changes to take effect except for some display related advanced parameters (see Appendix B).
- Close. Destroys the active query and closes its query window.
- Exit. Exits the Visualisation Islands application.

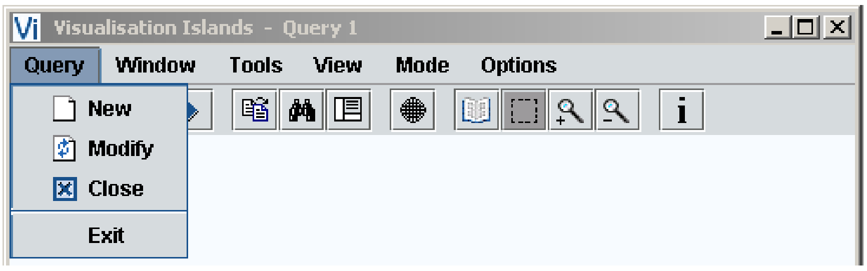


Figure A.2: VisIslands Query menu.

A.4.2 Window Menu

The windows menu displayed in Figure A.3 gives an overview of all currently opened queries and allows you to directly switch to each one. The currently active query window is selected.

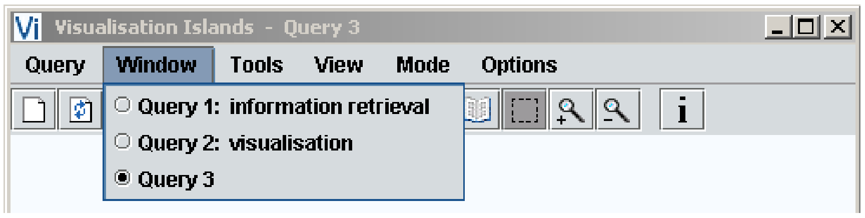

Visualisation Islands - Query 3
Query
Window
Tools
View
Mode
Options
Query 1: information retrieval
Query 2: visualisation
Query 3

Figure A.3: VisIslands Window menu.

A.4.3 Tools Menu

Tools menu, displayed in Figure A.4, offers tools for searching for objects with certain characteristics, viewing detailed object information and launching a pseudo-query by processing selected documents of the active query.

- Spawn query. Launches a pseudo-query by processing the selected documents of the current query and displaying them in a new query window. The pseudo-query will be named after keywords with highest product of frequency and weight and these will be placed in the query history list. This allows you to refine the results and explore them in more detail.
- Search. Invokes the search dialog where you can enter keywords, size and creating date intervals for the objects you are looking for. Objects found are selected.
- Details. Invokes the details window that displays all selected objects of the active query giving you the possibility to view information such as title, URL, size, creation date and keywords profiles of each object.

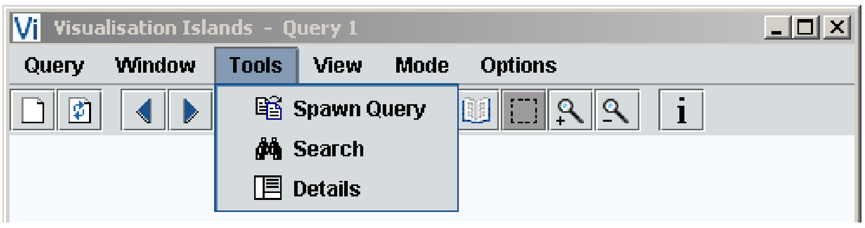


Figure A.4: VisIslands Tools menu.

A.4.4 View Menu

With the view menu, shown in Figures A.5 and A.6, you can adjust how and which results will be displayed. The menu is divided in three parts:

- Filtering. Offers two operations: a switch for enabling and disabling the filtering function and a button wjich invokes the filtering dialog where you can enter keywords, size and creating date intervals for the objects that should be displayed. Only objects with selected characteristics are displayed, all other object are hidden.

- The map part offers a submenu with three switches. With these you can choose if you want to display the grid, the map background and the labels in the map visualisation.

- The clusters part offers a submenu that lets you choose where the clusters will be displayed. They can be shown in both list and map, in the list only or can be hidden completely. In this case the list will display a relevance sorted list of documents as they were returned by the xFIND search engine.

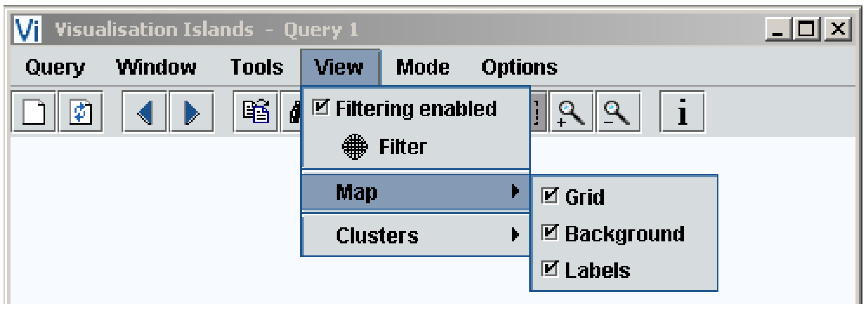


Figure A.5: VisIslands View menu, Map submenu.

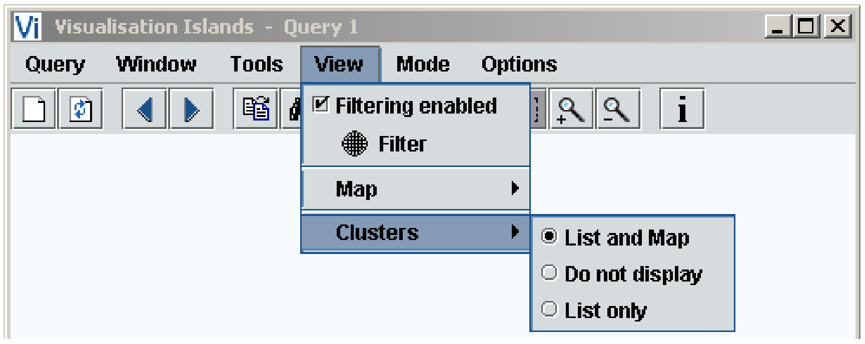


Figure A.6: VisIslands View menu, Clusters submenu.

A.4.5 Mode Menu

The mode menu, shown in Figure A.7, is used to choose one of the three different modes for managing and interacting with the objects displayed in the query window.

- In view mode any document can be viewed in the external viewer application by simply clicking on it.

- In select mode the mouse is used to select and deselect objects by clicking on them for single selection, or by dragging the mouse for multiple selection.

- In zoom mode the mouse can be dragged to select a rectangular map area which will be zoomed in when the mouse button is released.
- The zoom out button will not switch you to another mode, but will zoom out one step backwards in the zoom hierarchy.

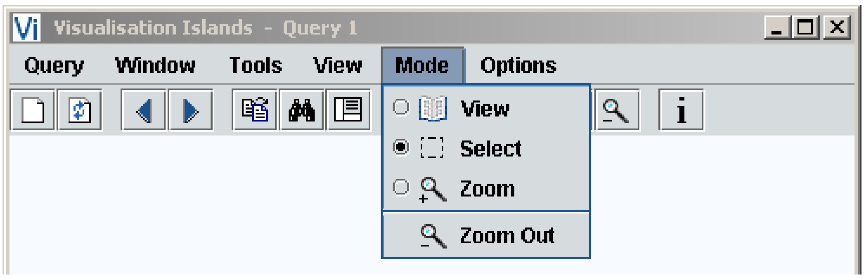


Figure A.7: VisIslands Mode menu.

A.4.6 Options Menu

The options menu, displayed in Figure A.8 offers following three functions:

- Invoking the query parameters dialog to edit the parameters used to initialise each new query.
- Invoking of the preferences dialog for setting some general program parameters and preferences.
- Invoking of the info dialog to display information on the program, author, author's advisors and the licence.

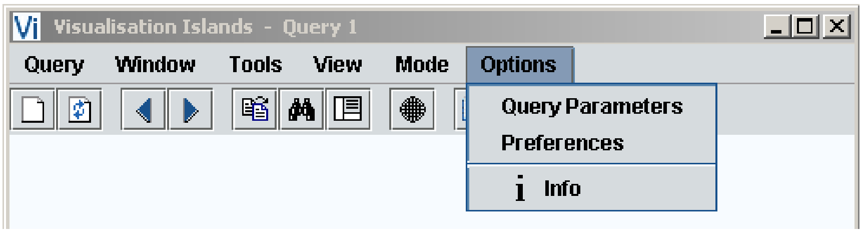


Figure A.8: VisIslands Options menu.

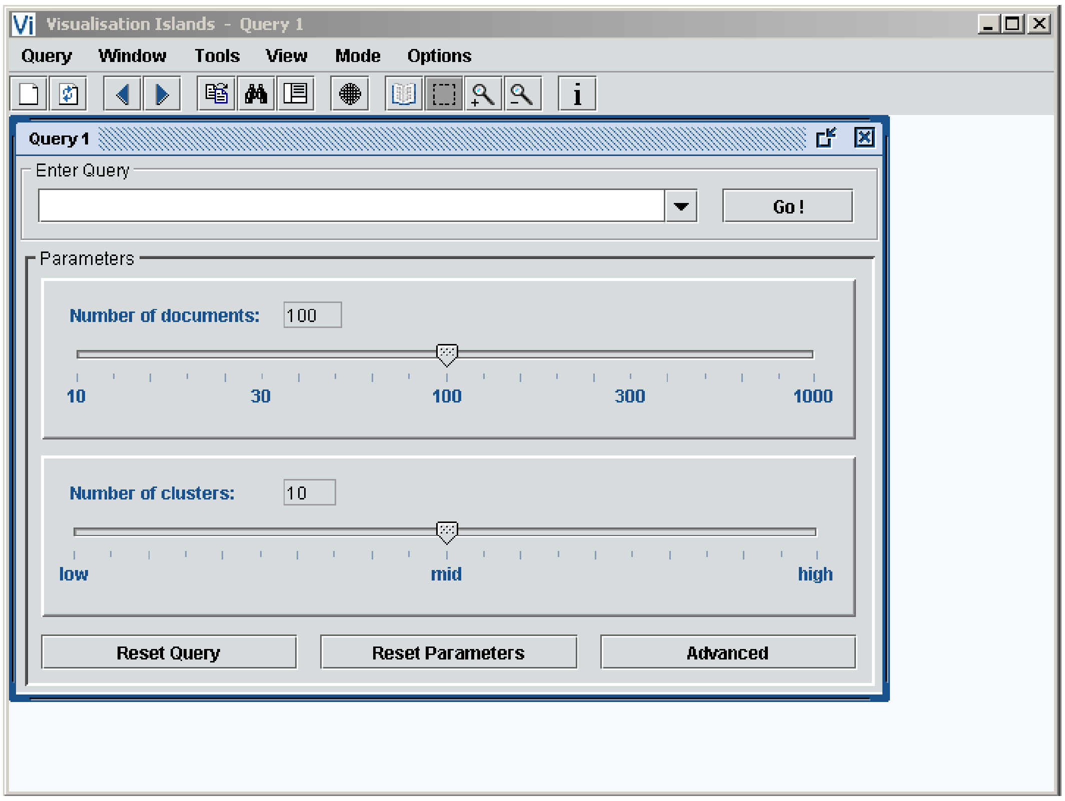
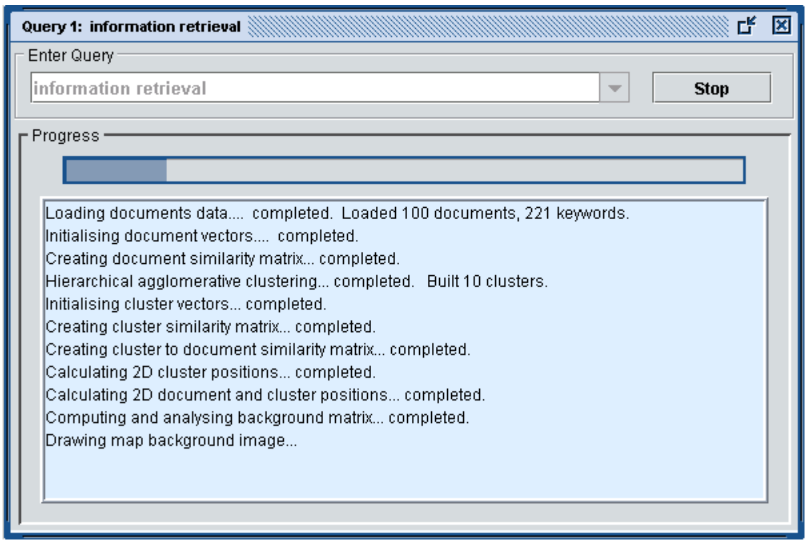


Figure A.9: VisIslands main window and query window waiting for input.



```
Loading documents data.... completed. Loaded 100 documents, 221 keywords.
Initialising document vectors.... completed.
Creating document similarity matrix... completed.
Hierarchical agglomerative clustering... completed.  Built 10 clusters.
Initialising cluster vectors... completed.
Creating cluster similarity matrix... completed.
Creating cluster to document similarity matrix... completed.
Calculating 2D cluster positions... completed.
Calculating 2D document and cluster positions... completed.
Computing and analysing background matrix... completed.
Drawing map background image...
```

Figure A.10: VisIslands loading and processing progress.

A.5 Main and Query Windows

Main window is a container for all existing query windows. It also contains the tool bar and the pull down menus. When a new query window is created, as displayed in Figure A.9, it is in query input mode. Here the query text can be entered or chosen from a query history list, and sent to the xFIND search engine by pressing the Go! button. The lower part of the window displays a parameters panel for editing query parameters. In standard mode the panel offers two sliders for choosing the number of desired documents and the number of desired clusters, a button for resetting the parameters to default values which are guaranteed to give good results and a query reset button which resets the parameters and the query text and discards all loaded data. The advanced parameters button flips the parameter panel into advanced parameters mode, which is reserved for expert users only. Advanced parameters are documented in Appendix B.

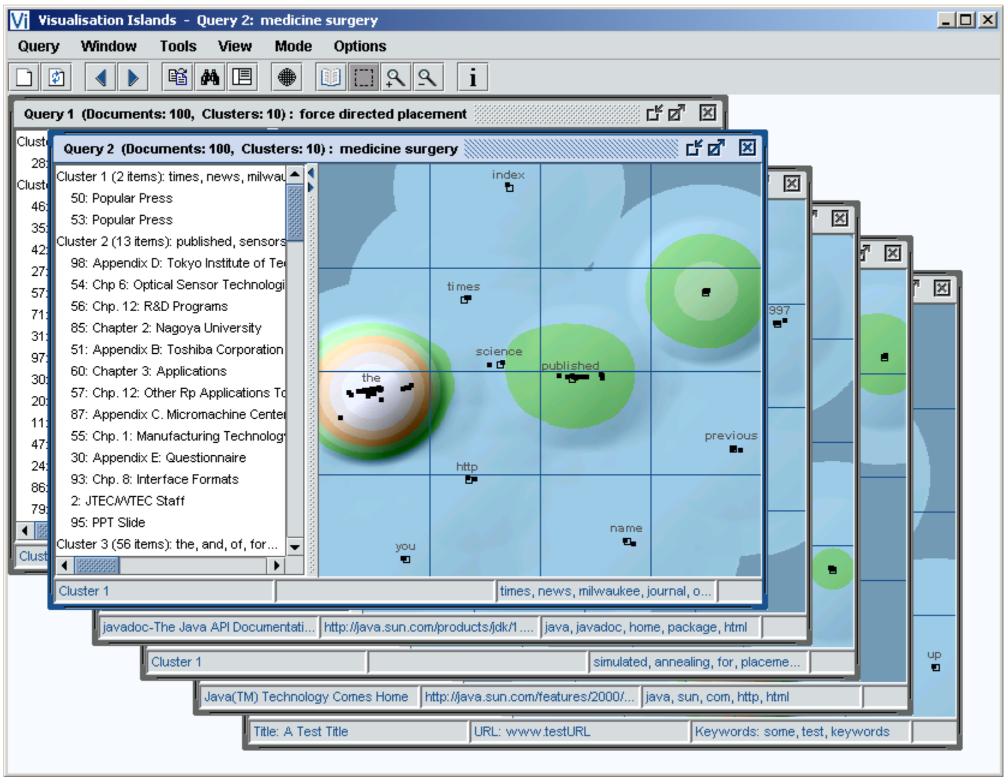
When the query has been sent to the xFIND server it responds by sending the resulting document data which are than processed by Visualisation Islands to create the visualisation. The query window switches to progress mode showing the progress of loading and of different processing stages, as can be seen in Figure A.10. Pressing the Stop button will interrupt the current action and switch the query window back to query input mode. Visualisation Islands supports simultaneous processing of more than one query. Also, while some queries are being processed, at the same time other, finished query windows can be viewed and explored.

When the documents are loaded and the processing is completed the query window switches to query display mode to show the created visualisation, as shown in Figure A.11. The left part of the window displays a scrollable, indented list of clusters and documents in a table of contents style. The right part of the window shows the topically organised map of documents which are represented by filled black icons, and clusters represented by hollow icons. Different regions are labeled according to the contents of documents placed in that area. The map background contains peaks at regions where object density is high indicating where a larger number of similar documents can be found.

The list and the map are divided by a bar which can be slid left and right to assign part of the area of the query window to either. When an object in the map or in the list is pointed to, its title, URL and a few most important keywords are displayed in the bar at the bottom of the window. The visualisation is fully interactive as is explained in detail in the following sections.

A.6 Selecting

To select objects you must first enter select mode. In this mode the mouse is used to select and deselect objects. One click select an object the next one deselects it. To select more objects in the map keep the left mouse button pressed while dragging it. A circle is drawn as you drag the mouse (Figure A.12) and when the button is released all objects inside of the circle will be selected (Figure A.13). The list is scrolled to display (some of) the selected objects. To select an interval in the list, click first on one object and than on another one keeping the “Shift” button down. This will select all objects between the two you clicked on. Double clicking a document or a cluster will, additionally to that object, select all elements of the corresponding cluster. By keeping the “Control” key down while clicking or dragging the mouse, objects can be added to the current selection or a single object can be removed from it.



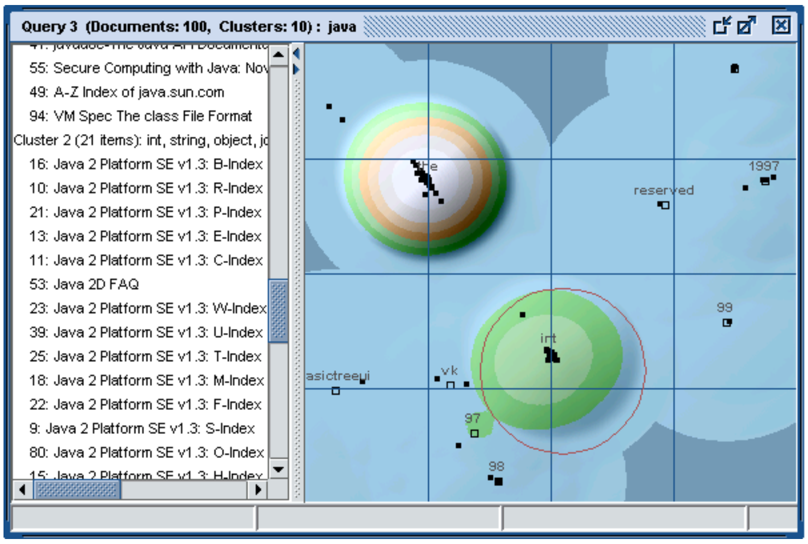
Query 1 (Documents: 100, Clusters: 10) : force directed placement

Query 2 (Documents: 100, Clusters: 10) : medicine surgery

Cluster 1 (2 items): times, news, milwau
50: Popular Press
53: Popular Press
Cluster 2 (13 items): published, sensors
98: Appendix D: Tokyo Institute of Te
54: Chp 6: Optical Sensor Technologi
56: Chp. 12: R&D Programs
85: Chapter 2: Nagoya University
51: Appendix B: Toshiba Corporation
60: Chapter 3: Applications
57: Chp. 12: Other Rp Applications To
87: Appendix C. Micromachine Center
55: Chp. 1: Manufacturing Technolog
30: Appendix E: Questionnaire
93: Chp. 8: Interface Formats
2: JTEC/WTEC Staff
95: PPT Slide
Cluster 3 (56 items): the, and, of, for...

Cluster 1		times, news, milwaukee, journal, o...
javadoc-The Java API Documentati...	http://java.sun.com/products/jdk/1....	java, javadoc, home, package, html
Cluster 1		simulated, annealing, for, placeme...
Java(TM) Technology Comes Home	http://java.sun.com/features/2000/...	java, sun, com, http, html
Title: A Test Title	URL: www.testURL	Keywords: some, test, keywords

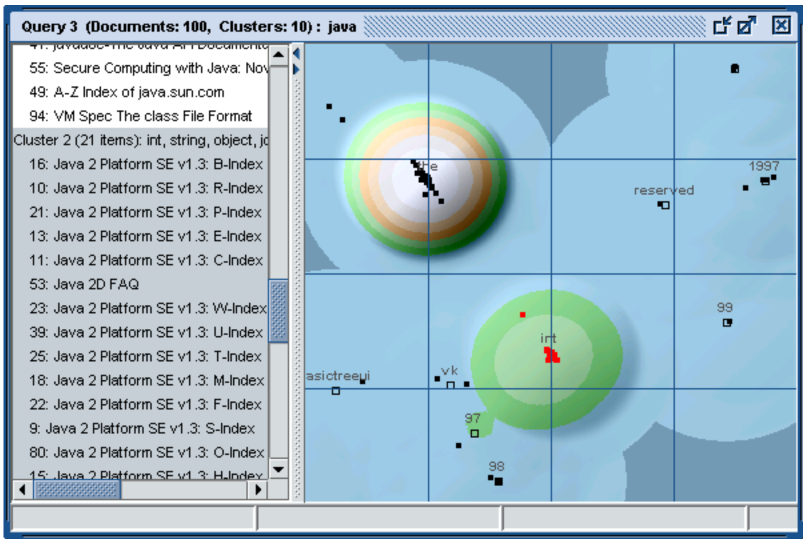
Figure A.11: VisIslands multiple query windows.



Query 3 (Documents: 100, Clusters: 10) : java

55: Secure Computing with Java: Nov
49: A-Z Index of java.sun.com
94: VM Spec The class File Format
Cluster 2 (21 items): int, string, object, ja
16: Java 2 Platform SE v1.3: B-Index
10: Java 2 Platform SE v1.3: R-Index
21: Java 2 Platform SE v1.3: P-Index
13: Java 2 Platform SE v1.3: E-Index
11: Java 2 Platform SE v1.3: C-Index
53: Java 2D FAQ
23: Java 2 Platform SE v1.3: W-Index
39: Java 2 Platform SE v1.3: U-Index
25: Java 2 Platform SE v1.3: T-Index
18: Java 2 Platform SE v1.3: M-Index
22: Java 2 Platform SE v1.3: F-Index
9: Java 2 Platform SE v1.3: S-Index
80: Java 2 Platform SE v1.3: O-Index
15: Java 2 Platform SE v1.3: H-Index

Figure A.12: VisIslands: dragging the mouse in select mode.



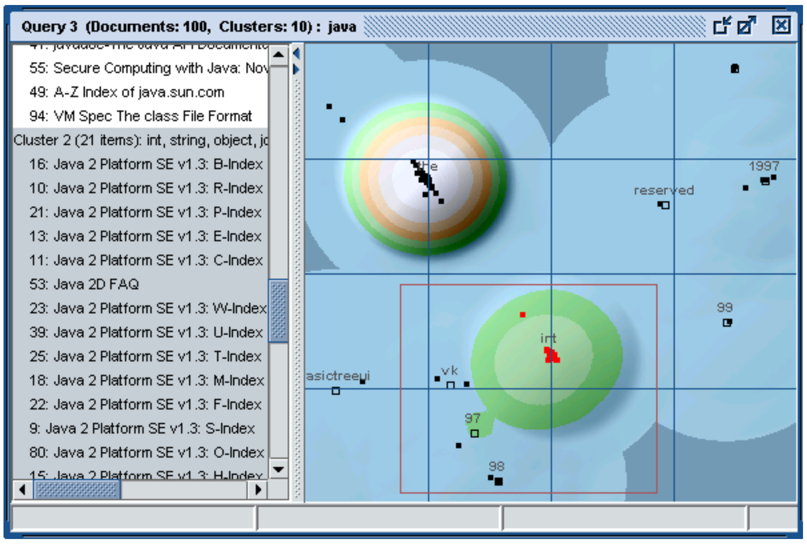
Query 3 (Documents: 100, Clusters: 10) : java

55: Secure Computing with Java: Nov
49: A-Z Index of java.sun.com
94: VM Spec The class File Format
Cluster 2 (21 items): int, string, object, jc
16: Java 2 Platform SE v1.3: B-Index
10: Java 2 Platform SE v1.3: R-Index
21: Java 2 Platform SE v1.3: P-Index
13: Java 2 Platform SE v1.3: E-Index
11: Java 2 Platform SE v1.3: C-Index
53: Java 2D FAQ
23: Java 2 Platform SE v1.3: W-Index
39: Java 2 Platform SE v1.3: U-Index
25: Java 2 Platform SE v1.3: T-Index
18: Java 2 Platform SE v1.3: M-Index
22: Java 2 Platform SE v1.3: F-Index
9: Java 2 Platform SE v1.3: S-Index
80: Java 2 Platform SE v1.3: O-Index
15: Java 2 Platform SE v1.3: H-Index

Figure A.13: VisIslands: selected objects are highlighted.

A.7 Zooming

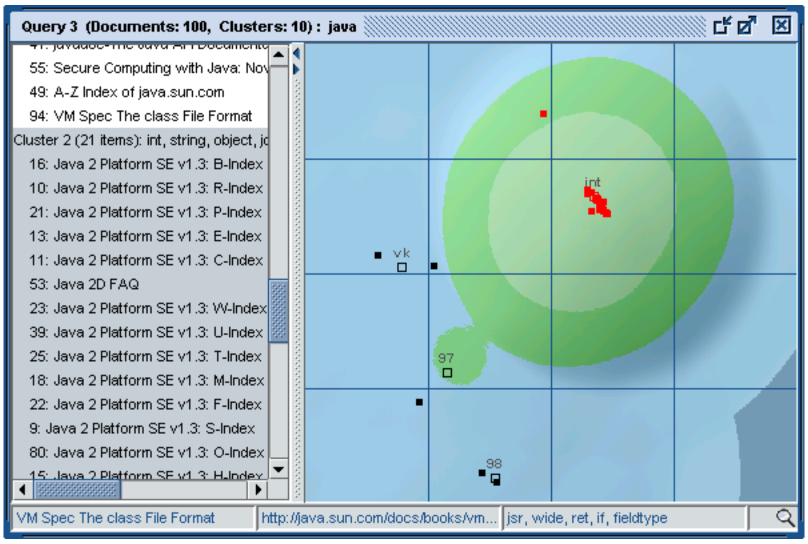
To zoom in to a part of the map you must first enter zoom mode. To select an area to be zoomed keep the left mouse button pressed while dragging it. A rectangle is drawn as you drag the mouse (Figure A.14) and when the button is released this part of the map will be displayed enlarged (Figure A.15). You can repeat this procedure to explore ever smaller portions of the map. Note that when zooming in, additional labels might be displayed to mark regions of the currently viewed area. You can move backwards in the zoom hierarchy by using the zoom out function.



Query 3 (Documents: 100, Clusters: 10) : java

55: Secure Computing with Java: Nov
49: A-Z Index of java.sun.com
94: VM Spec The class File Format
Cluster 2 (21 items): int, string, object, jc
16: Java 2 Platform SE v1.3: B-Index
10: Java 2 Platform SE v1.3: R-Index
21: Java 2 Platform SE v1.3: P-Index
13: Java 2 Platform SE v1.3: E-Index
11: Java 2 Platform SE v1.3: C-Index
53: Java 2D FAQ
23: Java 2 Platform SE v1.3: W-Index
39: Java 2 Platform SE v1.3: U-Index
25: Java 2 Platform SE v1.3: T-Index
18: Java 2 Platform SE v1.3: M-Index
22: Java 2 Platform SE v1.3: F-Index
9: Java 2 Platform SE v1.3: S-Index
80: Java 2 Platform SE v1.3: O-Index
15: Java 2 Platform SE v1.3: H-Index

Figure A.14: VisIslands: dragging the mouse in zoom mode.


Query 3 (Documents: 100, Clusters: 10) : java

55: Secure Computing with Java: Nov
49: A-Z Index of java.sun.com
94: VM Spec The class File Format
Cluster 2 (21 items): int, string, object, jc
16: Java 2 Platform SE v1.3: B-Index
10: Java 2 Platform SE v1.3: R-Index
21: Java 2 Platform SE v1.3: P-Index
13: Java 2 Platform SE v1.3: E-Index
11: Java 2 Platform SE v1.3: C-Index
53: Java 2D FAQ
23: Java 2 Platform SE v1.3: W-Index
39: Java 2 Platform SE v1.3: U-Index
25: Java 2 Platform SE v1.3: T-Index
18: Java 2 Platform SE v1.3: M-Index
22: Java 2 Platform SE v1.3: F-Index
9: Java 2 Platform SE v1.3: S-Index
80: Java 2 Platform SE v1.3: O-Index
15: Java 2 Platform SE v1.3: H-Index

VM Spec The class File Format | http://java.sun.com/docs/books/vm... | jsr, wide, ret, if, fieldtype

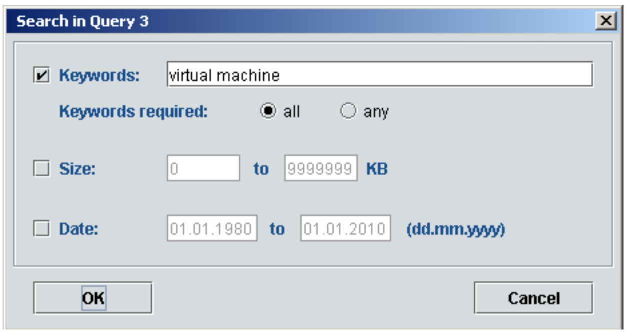
Figure A.15: VisIslands: a zoomed in part of the map.

A.8 Searching

You can search for objects having desired characteristics using the search dialog displayed in Figure A.16. You can search according to three different criteria and combine them by selecting or deselecting the corresponding switches:

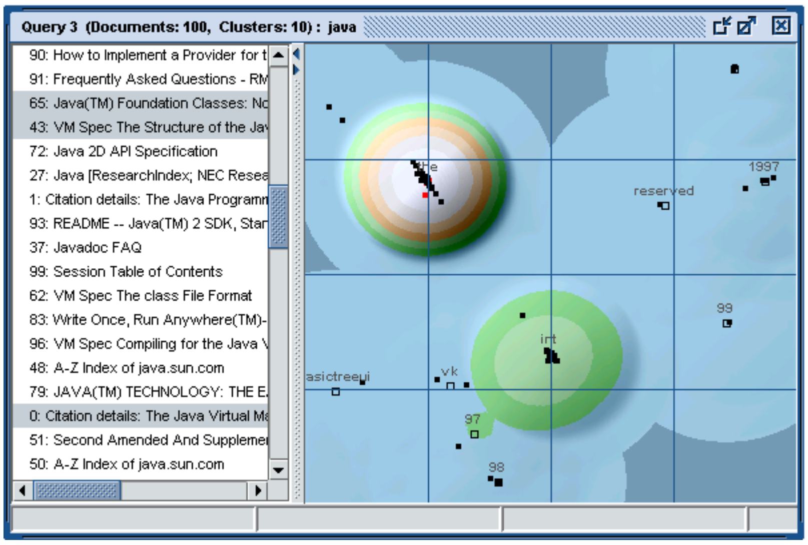
- Keyword search. Enter keywords you are looking for and choose if all or only some of them must be in the objects you are looking for.
- Size search. Enter the size interval in KB for the objects you are looking for.
- Date search. Enter the creation date interval for the objects you are looking for.

Objects with the selected features will be selected as shown in Figure A.17.


Search in Query 3

[✓] Keywords: virtual machine
Keywords required: [✓ all] any
[ ] Size: 0 to 9999999 KB
[ ] Date: 01.01.1980 to 01.01.2010 (dd.mm.yyyy)

Figure A.16: VisIslands search dialog.


Query 3 (Documents: 100, Clusters: 10) : java

90: How to Implement a Provider for t
91: Frequently Asked Questions - RM
65: Java(TM) Foundation Classes: No
43: VM Spec The Structure of the Jav
72: Java 2D API Specification
27: Java [ResearchIndex; NEC Resea
1: Citation details: The Java Programm
93: README -- Java(TM) 2 SDK, Stan
37: Javadoc FAQ
99: Session Table of Contents
62: VM Spec The class File Format
83: Write Once, Run Anywhere(TM)-
96: VM Spec Compiling for the Java V
48: A-Z Index of java.sun.com
79: JAVA(TM) TECHNOLOGY: THE E
0: Citation details: The Java Virtual Ma
51: Second Amended And Suppleme
50: A-Z Index of java.sun.com

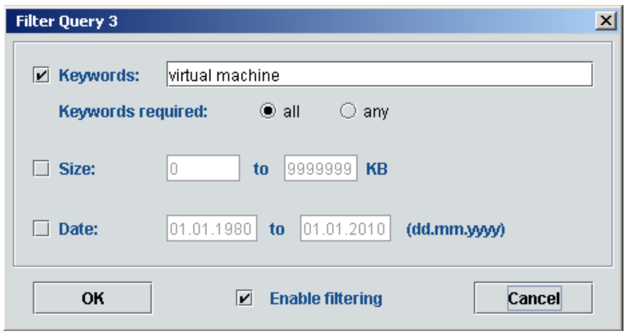
Figure A.17: VisIslands: found objects are selected.

A.9 Filtering

You can filter out objects having desired characteristics using the filter dialog displayed in Figure A.18. Only objects satisfying the filter condition will be displayed, all other objects are hidden. Filtering can be turned on and off using a general switch placed in the bottom of the dialog. You can filter according to three different criteria and combine them by selecting or deselecting the corresponding switches:

- Keyword filtering. Enter keywords you are looking for and choose if all or some of them must be in the objects you would like to have displayed.
- Size search. Enter the size interval in KB of the objects you would like to filter have displayed.
- Date search. Enter the creation date interval of the objects you would like to have displayed.

Objects with selected features are displayed as shown in Figure A.19.


Filter Query 3

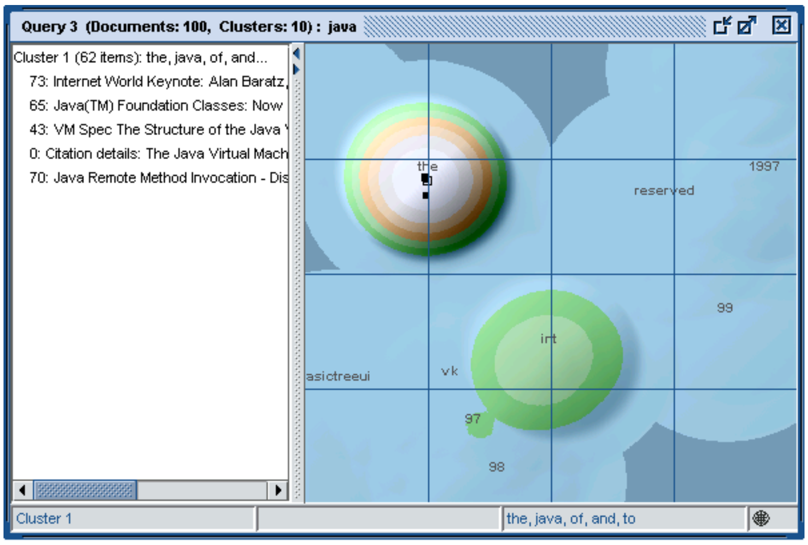
☑ Keywords: virtual machine

Keywords required: [✓ all] any

☐ Size: 0 to 9999999 KB

☐ Date: 01.01.1980 to 01.01.2010 (dd.mm.yyyy)

Figure A.18: VisIslands filter dialog.


Query 3 (Documents: 100, Clusters: 10) : java

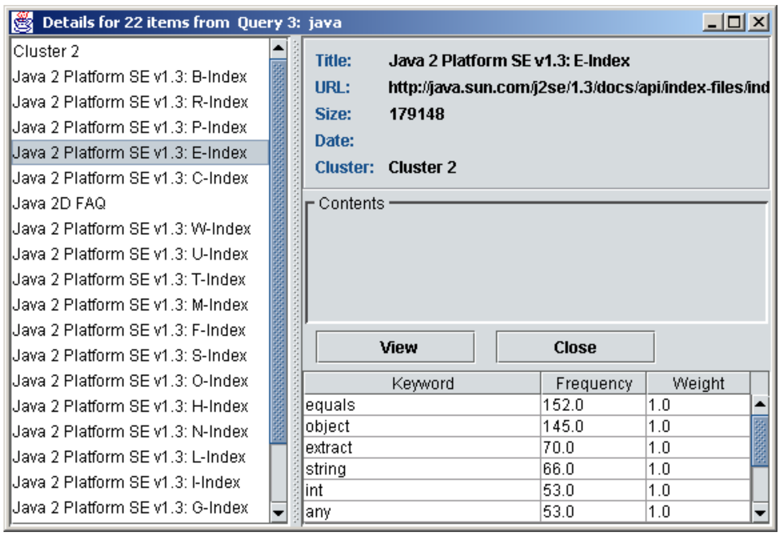
Cluster 1 (62 items): the, java, of, and...
73: Internet World Keynote: Alan Baratz,
65: Java(TM) Foundation Classes: Now
43: VM Spec The Structure of the Java
0: Citation details: The Java Virtual Mach
70: Java Remote Method Invocation - Dis

Cluster 1 | | the, java, of, and, to

Figure A.19: VisIslands: only objects satisfying the filter condition are displayed.

A.10 Details Window

When the details window, shown in Figure A.20, is invoked, it displays selected documents and clusters as well as detailed information for each of these objects. The left side of the window contains a list of objects letting you choose any of them with a mouse click. Detailed information for that object is displayed: title, URL, size, creation date, cluster it belongs to and a part of its content if it is a document[1]. A table displaying object's keywords and their frequencies and weights is also displayed. A view button is available for viewing documents in an external viewer application.


Details for 22 items from Query 3: java

Cluster 2
Java 2 Platform SE v1.3: B-Index
Java 2 Platform SE v1.3: R-Index
Java 2 Platform SE v1.3: P-Index
Java 2 Platform SE v1.3: E-Index
Java 2 Platform SE v1.3: C-Index
Java 2D FAQ
Java 2 Platform SE v1.3: W-Index
Java 2 Platform SE v1.3: U-Index
Java 2 Platform SE v1.3: T-Index
Java 2 Platform SE v1.3: M-Index
Java 2 Platform SE v1.3: F-Index
Java 2 Platform SE v1.3: S-Index
Java 2 Platform SE v1.3: O-Index
Java 2 Platform SE v1.3: H-Index
Java 2 Platform SE v1.3: N-Index
Java 2 Platform SE v1.3: L-Index
Java 2 Platform SE v1.3: I-Index
Java 2 Platform SE v1.3: G-Index

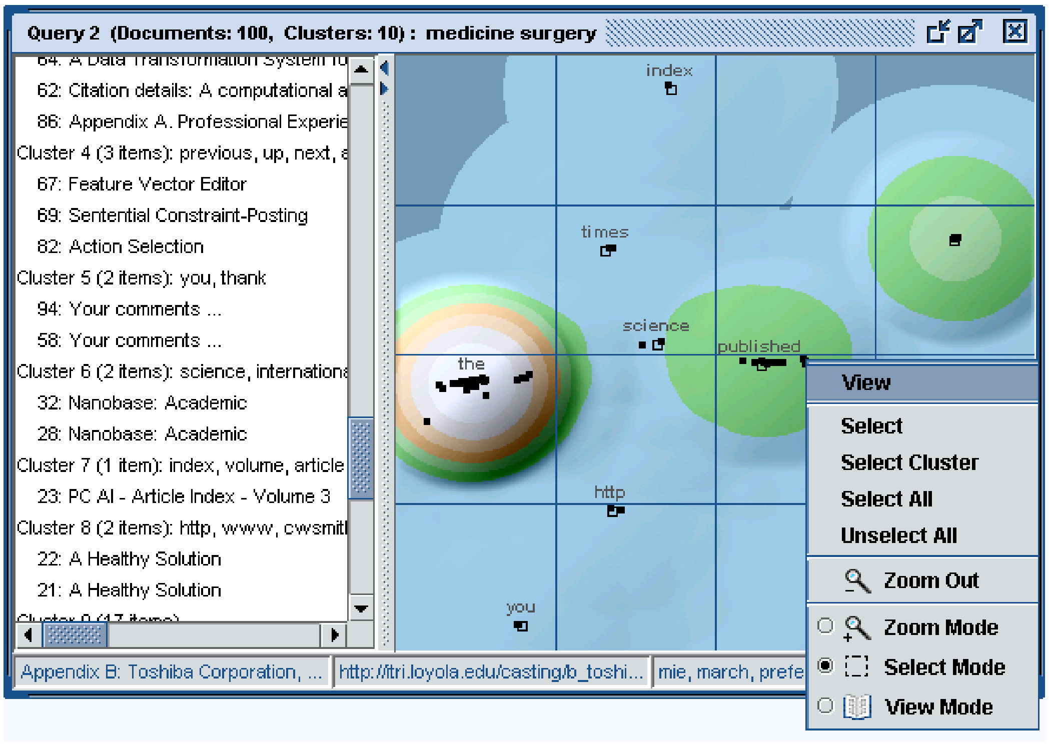
Title: Java 2 Platform SE v1.3: E-Index
URL: http://java.sun.com/j2se/1.3/docs/api/index-files/ind
Size: 179148
Date:
Cluster: Cluster 2

Contents

Keyword	Frequency	Weight
equals	152.0	1.0
object	145.0	1.0
extract	70.0	1.0
string	66.0	1.0
int	53.0	1.0
any	53.0	1.0

Figure A.20: VisIslands details window.

[1]For technical reasons the content information is not loaded from the xFIND server in this Visualisation Islands version.


Query 2 (Documents: 100, Clusters: 10) : medicine surgery

64: A Data Transformation System fo
62: Citation details: A computational a
86: Appendix A. Professional Experie
Cluster 4 (3 items): previous, up, next, e
67: Feature Vector Editor
69: Sentential Constraint-Posting
82: Action Selection
Cluster 5 (2 items): you, thank
94: Your comments ...
58: Your comments ...
Cluster 6 (2 items): science, internationa
32: Nanobase: Academic
28: Nanobase: Academic
Cluster 7 (1 item): index, volume, article
23: PC AI - Article Index - Volume 3
Cluster 8 (2 items): http, www, cwsmitl
22: A Healthy Solution
21: A Healthy Solution
Cluster 9 (17 items)

Appendix B: Toshiba Corporation, ... | http://itri.loyola.edu/casting/b_toshi... | mie, march, prefe

Figure A.21: VisIslands popup menu.

A.11 Popup Menu

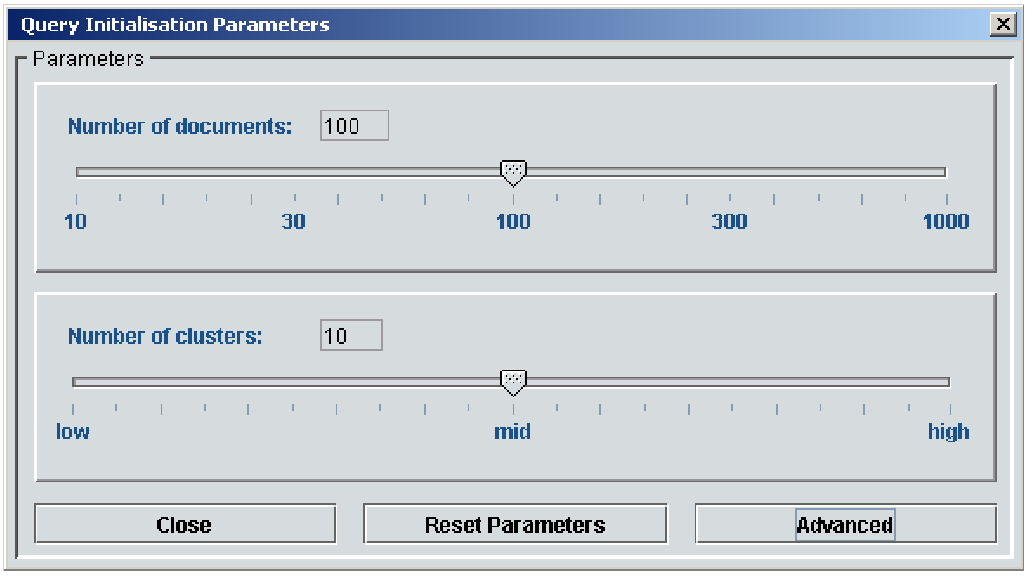
The popup menu, displayed in Figure A.21, appears when the right mouse button is clicked inside of the map or inside the list. The following operations are offered, although some might be disabled depending on the kind of object the mouse pointer was positioned over:

- View. Opens the document, if any, that was under the mouse pointer when the right mouse button was clicked in the viewer application.
- Select. Selects the object, document or cluster, that was under the mouse pointer when the right mouse button was clicked.
- Select cluster. Selects all objects in the cluster if applied to a cluster, or, if applied to a document, all documents which are elements of the same cluster.
- Select all. Selects all objects of that query.
- Unselect all. Unselects all objects of the query.
- Zoom out. Zooms out one step backwards in the zoom hierarchy.
- Zoom mode. Switches to zoom mode. In zoom mode the mouse can be dragged to select a rectangular map area which is zoomed in when the mouse button is released.
- Select mode. Switches to select mode. In select mode the mouse can be dragged to select a circular map area inside of which all objects are selected. Objects in the list are selected by simply clicking on them.

- View mode. Switches to view mode. In this mode, when the mouse is clicked over a document, an external viewer application is launched to display the document's content.

A.12 Parameters Dialog

The parameters dialog, shown in Figure A.22, displays a special panel which enables you to change all parameters which determine how the query results will be loaded, processed, and visualised. The parameters entered in this parameter dialog will be used to initialise every new query, existing queries are not affected by any changes. These parameters are saved when the program is exited and reloaded again the next time it is started. In the standard mode you are able to change the maximum number of documents that should be loaded[2] and the desired number of clusters to be created with two sliders. You have the possiblity to enter the advanced parameters mode, and to reset all parameters to standard values that are likely to give good results. Advanced parameters mode is reserved for the expert user only. Details on advanced parameters are given in Appendix B. Note that the same parameters panel which is used to edit parameters in the parameters dialog, is also displayed in a query window when it is in query input mode. In this case it is only used to edit local parameters of that query.


Query Initialisation Parameters

Parameters

Number of documents: 100

Number of clusters: 10

Figure A.22: VisIslands parameters dialog.

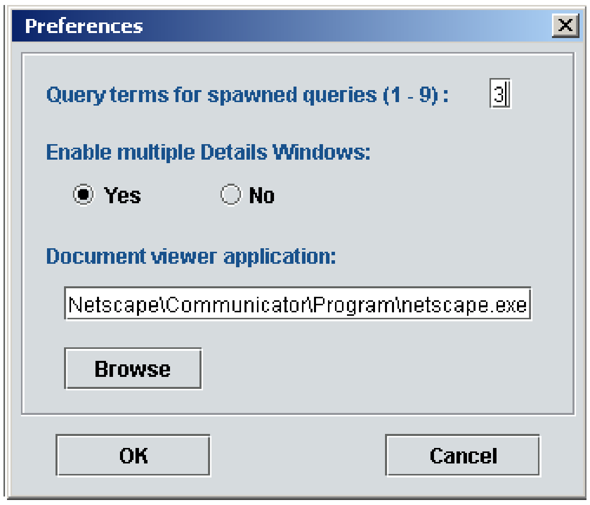
A.13 Preferences Dialog

The preferences dialog, displayed in Figure A.23, gives you the possibility to adjust three settings which influence the behaviour of the whole program, not just a single query. These are:

- Text entry field for the number of query terms for spawned pseudo-queries lets you adjust how many keywords are extracted from the selected documents to act as a title of a pseudo-query spawned from them.

[2]Due to a technical reasons the number of documents is currently limited to maximally 200.

- Enable multiple detail windows switch determines if details of selected objects are always displayed in a single window or each time a new details window will be opened.
- Document viewer application text field lets you type in the exact path and name of the application used to view the document content. With the browse button a file dialog is opened giving you the possibility to browse the file system and select the application with a mouse click.



Query terms for spawned queries (1 - 9) : 3

Enable multiple Details Windows:

[✓ Yes] No

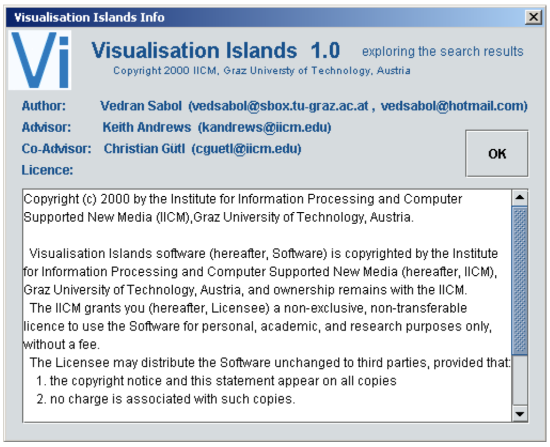
Document viewer application:

Netscape\Communicator\Program\netscape.exe

Figure A.23: VisIslands preferences dialog.

A.14 Info Dialog

Info dialog, shown in Figure A.24, displays information about the author of the Visualisation Islands software package, his advisors, the program version and a licence agreement.



Visualisation Islands 1.0 exploring the search results

Copyright 2000 IICM, Graz Universty of Technology, Austria

Author: Vedran Sabol (vedsabol@sbox.tu-graz.ac.at , vedsabol@hotmail.com)

Advisor: Keith Andrews (kandrews@iicm.edu)

Co-Advisor: Christian Gütl (cguetl@iicm.edu)

Licence:

Copyright (c) 2000 by the Institute for Information Processing and Computer Supported New Media (IICM),Graz University of Technology, Austria.

Visualisation Islands software (hereafter, Software) is copyrighted by the Institute for Information Processing and Computer Supported New Media (hereafter, IICM), Graz University of Technology, Austria, and ownership remains with the IICM.
The IICM grants you (hereafter, Licensee) a non-exclusive, non-transferable licence to use the Software for personal, academic, and research purposes only, without a fee.
The Licensee may distribute the Software unchanged to third parties, provided that:
1. the copyright notice and this statement appear on all copies
2. no charge is associated with such copies.

Figure A.24: VisIslands info dialog.

Appendix B

Visualisation Islands Advanced Parameters

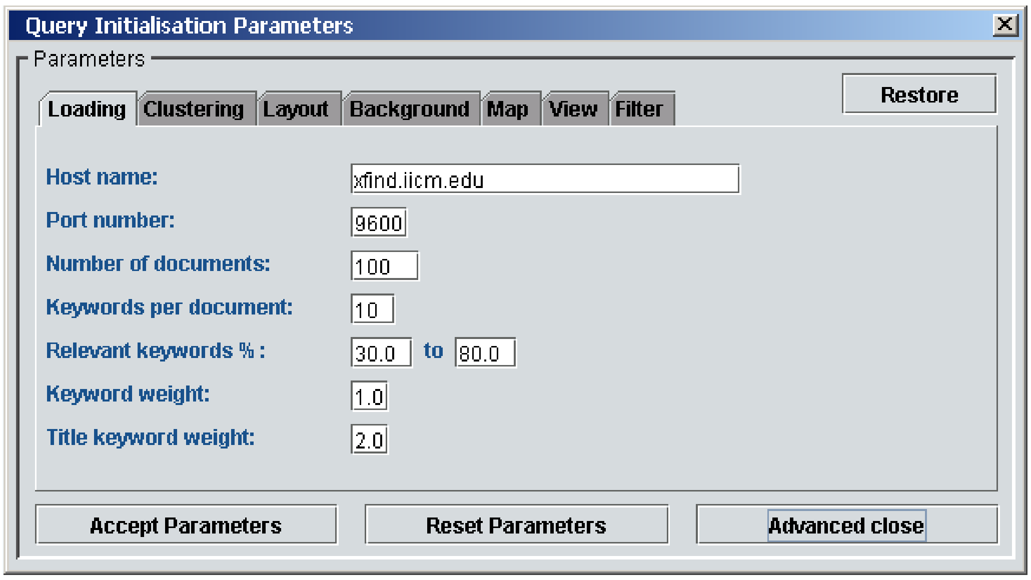
Advanced parameters are intended for the expert user only. These parameters give you the possibility to change almost every parameter concerning the loading, processing and visualisation of the data. They are organised into seven groups: loading parameters, clustering parameters, layout parameters, map background parameters, map parameters, view parameters and filter parameters. It is important to mention at this point that if query parameters from the first four groups are modified the query will have to be reloaded and reprocessed for changes to take effect. Changes on the last three groups of parameters can be applied directly on the currently loaded data set.

B.1 Loading Parameters

The loading panel (see Figure B.1) gives you the possibility to set the parameters concerning the communication with the xFIND search engine and the loading of document data.

- Host name: The name of the xFIND server the query is sent to.
- Port number: The port on which the xFIND server is waiting for requests.
- Number of documents: The maximum number of documents which should be sent back by the xFIND server in response to a query[1].
- Keywords per document: The maximum number of relevant keywords returned per document.
- Relevant keywords percentage interval : A keyword is considered relevant if it is contained by the percentage of documents defined by this interval.
- Keyword weight: The weight assigned to a standard keyword.
- Title keyword weight: The weight assigned to a keyword which is part of the document's title.

[1]Due to technical reasons the number of documents is currently limited to maximally 200.



Host name:	xfind.iicm.edu
Port number:	9600
Number of documents:	100
Keywords per document:	10
Relevant keywords % :	30.0 to 80.0
Keyword weight:	1.0
Title keyword weight:	2.0

Figure B.1: Loading parameters.

B.2 Clustering Parameters

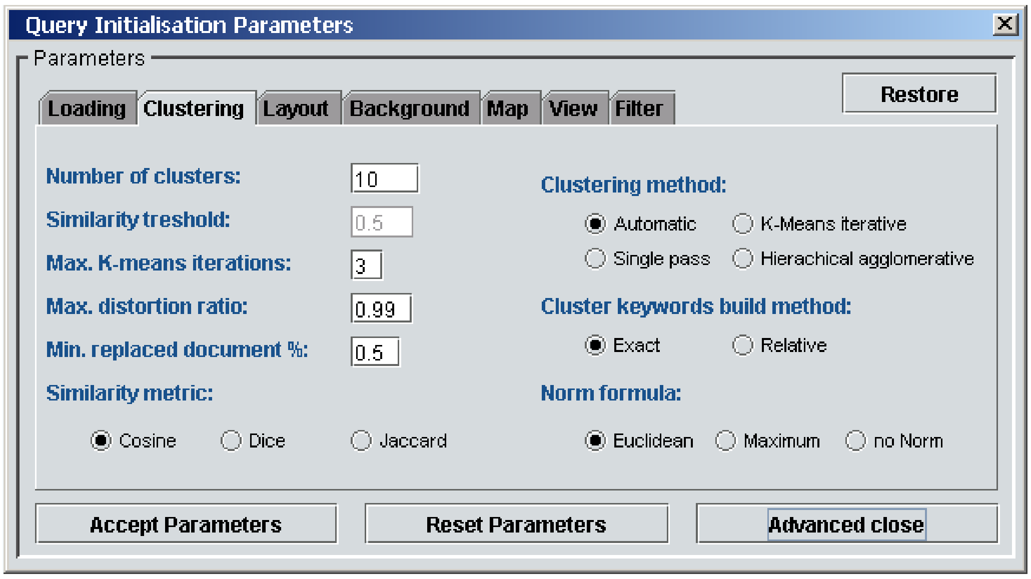
Clustering parameters (see Figure B.2) determine the usage and behaviour of different clustering algorithms.

- Number of clusters: The number of clusters which should be created by the hierarchical agglomerative or k-means clustering algorithm.
- Similarity threshold: This value is relevant for the single pass clustering algorithm. It determines a minimum similarity between a document and a cluster. If the document's similarity to a cluster is smaller than this value it can not be assigned to that cluster.
- Maximum k-means iterations: The upper limit on the number of iterations performed by the k-means clustering algorithm.
- Maximum distortion ratio: A k-means clustering algorithm stop condition value. When the ratio between the distortion value for the current and the previous iteration becomes larger than this value it means that no significant improvement in the partition quality was achieved during the last iteration and the algorithm will be stopped.
- Minimum replaced document percentage: Another k-means clustering algorithm stop condition value. If the percentage of replaced documents in the last iteration was smaller than this value the algorithm is stopped.
- Similarity metric: A similarity metric is used to compare two objects. Three different metrics can be chosen: cosine, dice and jaccard.
- Clustering method: Four choices are available: k-means clustering applied on a random initial partition, single pass clustering with k-means refinement, hierarchical agglomerative clustering and auto mode. Note that in the current implementation hierarchical agglomerative clustering is always used in auto mode, due to the 1000 document limitation of the current implementation.

Eventual future program versions, allowing the processing of more than 1000 documents, might switch to a faster algorithm beyond this point.

- Cluster keywords build method: Two methods are offered. The exact method reads keywords belonging to each document in that cluster, adds their frequencies of occurrence and assigns the keywords to the cluster. The relative method is faster since it extracts the keywords from the documents' high dimensional vector. As these may be normalised the keyword frequencies are floating numbers instead of integers in this case. Note that the keyword profile of the cluster (and its high-dimensional vector) is in both cases the same.

- Norm formula: Determines how document high-dimensional vectors will be normalised. Three possibilities are available: Euclidean norm, maximum norm and no normalisation at all. No normalisation leads to larger documents having proportionally more influence on cluster centroids, because their high-dimensional vector will be larger. This is not the case when using the Euclidean norm, where all documents will have the same influence on their cluster's centroid.



Number of clusters: 10
Similarity treshold: 0.5
Max. K-means iterations: 3
Max. distortion ratio: 0.99
Min. replaced document %: 0.5
Similarity metric: [✓ Cosine] Dice Jaccard

Clustering method: [✓ Automatic] K-Means iterative Single pass Hierachical agglomerative
Cluster keywords build method: [✓ Exact] Relative
Norm formula: [✓ Euclidean] Maximum no Norm

Figure B.2: Clustering parameters.

B.3 Layout Parameters

This section describes the parameters which determine the behaviour of the force-directed placement algorithm with stochastic sampling (see Figure B.3).

- Similarity force factor: The attractive component of the force proportional to the similarity of the interacting pair of objects is multiplied by this factor.

- Distance force factor: The attractive component of the force proportional to the distance of the interacting pair of objects is multiplied by this factor.

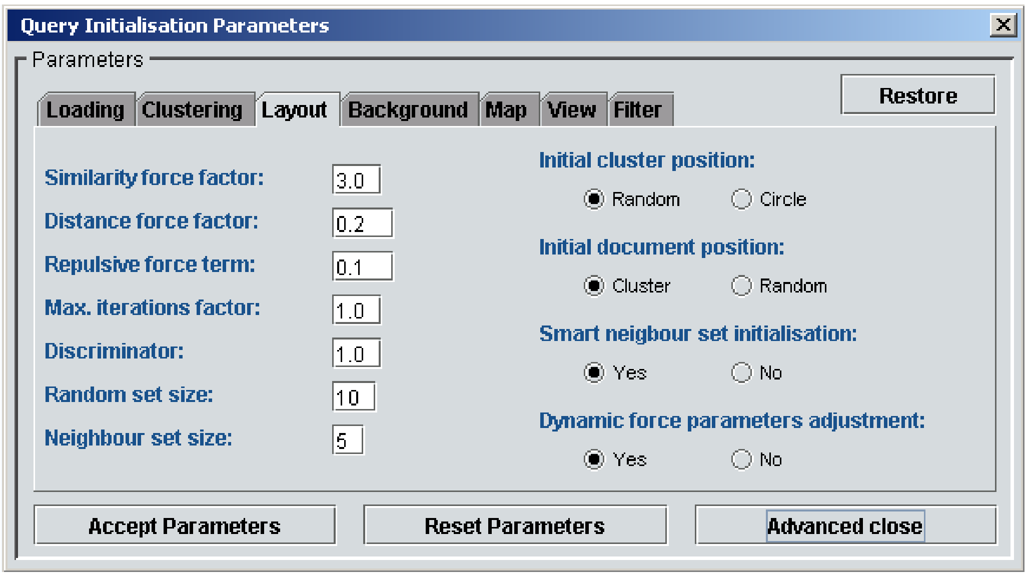
- Repulsive force term: This is a constant repulsive component of the force.

- Maximum iteration factor: The number of processed objects is multiplied by this factor to calculate the maximum number of iterations which will be performed by the force-directed placement algorithm.
- Discriminator: The similarity values used to compute the force are raised to the power of this coefficient. If the objects in the considered set are all very similar to each other then increasing this value will improve the discrimination power of the force-directed placement algorithm. Care has to be taken as an inverse effect will be achieved if the similarity values are small.
- Random set size: Size of the random set used for stochastic sampling. For each iteration of the algorithm it is filled with a random sample of objects.
- Neighbour set size: Size of the neighbour set used for stochastic sampling. It contains objects similar to the object owning the set.
- Initial cluster position: Before applying the force-directed placement algorithm clusters can be placed in a form of a circle or they can be placed on random positions of the low dimensional space.
- Initial document position: Before applying the force-directed placement algorithm on documents, clusters have already been processed. At the beginning the documents can be placed at random positions in the low dimensional space, or can be arranged around the cluster positions to improve the computed layout quality. This will also contribute to avoiding a document becoming stuck in a local minimum far from objects similar to it.
- Smart neigbour set initialisation: Before the force-directed placement algorithm is applied the neighbour set of each object is filled with random elements. If this option is set then the objects's neigbour set elements are chosen from the cluster the object belongs to, as far as these are available.
- Dynamic force parameters adjustment: If this option is set the distance force factor and the repulsive force factor are inverse proportionally scaled with the number of processed objects. If this option is not turned on and a large number of objects is processed a tendency of the algorithm to place the objects in a form of a ring around the origin of the 2-D space will be noticed. This is a result of pressing them into a small area. When too many objects are pulled together by a high distance force factor then the repulsive forces, resulting from a high the repulsive force term, are those that determine how the final layout will look like, not the attractive forces resulting from the inter-objects similarities.

B.4 Map Background Parameters

Map background parameters (see Figure B.4) determine the characteristics of the generated background image.

- Light shade limit: If a 3-D background is generated from each of the eight base map colours a palette of darker or lighter shades is computed. This value determines how much lighter than the original shade, which has a shade value of 0, the lightest shade is allowed to be.



Similarity force factor:	3.0
Distance force factor:	0.2
Repulsive force term:	0.1
Max. iterations factor:	1.0
Discriminator:	1.0
Random set size:	10
Neighbour set size:	5

Initial cluster position: [✓ Random] Circle

Initial document position: [✓ Cluster] Random

Smart neigbour set initialisation: [✓ Yes] No

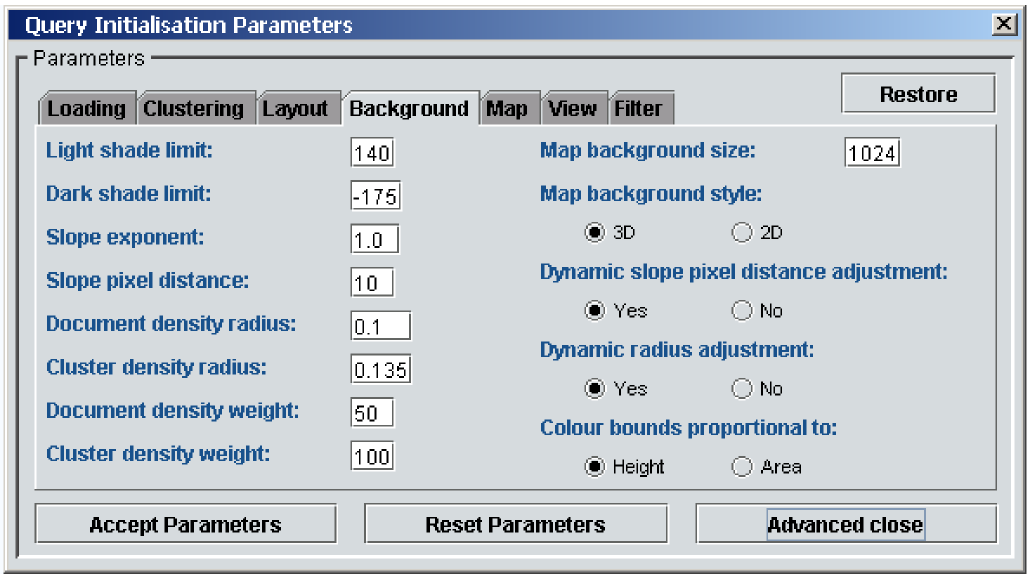
Dynamic force parameters adjustment: [✓ Yes] No

Figure B.3: Layout parameters.

- Dark shade limit: If a 3-D background is generated every of the eight base map colours is made darker or lighter to provide shading. This value determines how much darker than the original shade, which has a shade value of 0, the darkest shade is allowed to be.
- Slope pixel distance: A base colour assigned to a pixel is made lighter or darker depending on the slope value at that pixel's position. The slope for a pixel is computed as a difference of heights of two pixels surrounding it. The distance of these two pixels from the original pixel is given by this value. Larger values results in smooth, realistic peaks and islands, however at the cost of filtering out small details.
- Slope exponent: The slope value for each pixel is raised to the power of this value. Values smaller than 1 result in sharper colour transitions between the darker and lighter regions of the background image, while values larger than 1 result in very smooth colour transitions.
- Document density radius: The radius of the peak surrounding each document.
- Cluster density radius: The radius of the peak surrounding each cluster.
- Document density weight: The height of the centre of the peak surrounding each document. The weights are added for overlapping peaks to determine the local object density at each pixel position.
- Cluster density weight: The height of the center of the peak surrounding each cluster. The weights are added for overlapping peaks to determine the local object density at each pixel position.
- Map background size: This value defines the size (resolution) of the raster, in pixels, containing the map background image.
- Map background style: A 2-D style or a 3-D style map background image can be chosen.
- Dynamic slope distance adjustment: If this switch is enabled the slope pixel distance will be slightly reduced for large object sets allowing smaller details to be recognised. For small objects

sets the slope pixel distance value will be slightly enlarged for better smoothing of transitions between overlapping peaks. The default value of objects is 100.

- Dynamic radius adjustment: If this option is set to true the radius of documents and clusters is is made slightly larger as the number of processed get smaller. This improves the quality of the background image for small object sets.
- Colour bounds proportionality: Normally one of the eight base colours is assigned to each pixel proportionally depending on the height at that pixel position. The height is computed by adding overlapping peaks surrounding each object. Sometimes better results can be achieved if base colours are not assigned exactly proportionally to the height, but so that each colour is assigned to approximately the same amount of pixels. Recommended only for 2-D style backgrounds.



Light shade limit:	140	Map background size:	1024
Dark shade limit:	-175	Map background style:	[✓ 3D] 2D
Slope exponent:	1.0		
Slope pixel distance:	10	Dynamic slope pixel distance adjustment:	[✓ Yes] No
Document density radius:	0.1		
Cluster density radius:	0.135	Dynamic radius adjustment:	[✓ Yes] No
Document density weight:	50	Colour bounds proportional to:	[✓ Height] Area
Cluster density weight:	100		

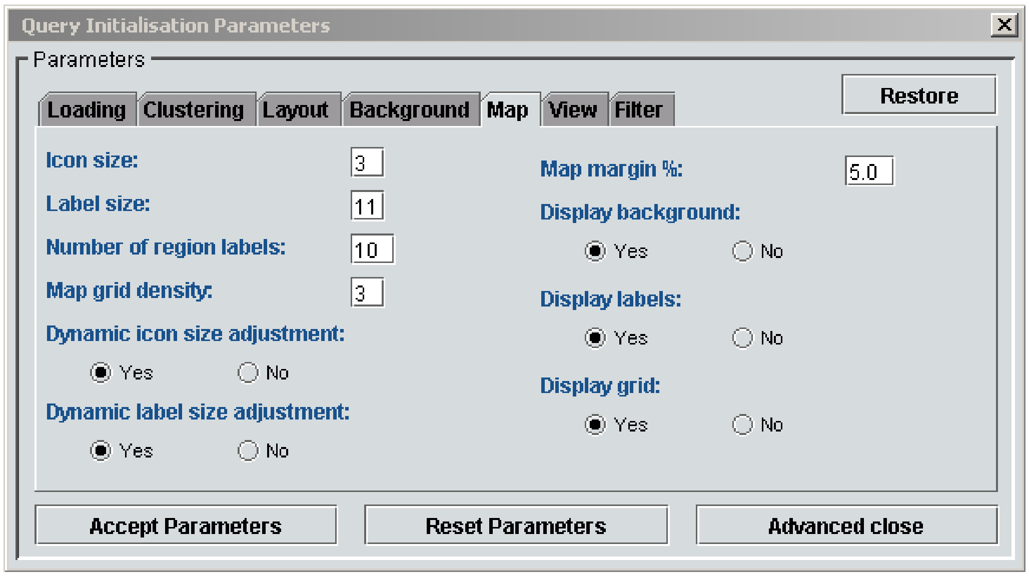
Figure B.4: Map background parameters.

B.5 Map Parameters

These parameters (see Figure B.5) determine how the map of topically organised documents will be displayed on the screen.

- Icon size: Size of a document icon. Cluster icon is one pixel larger than the document icon.
- Label size: Size of the font used to display the labels.
- Number of region labels: Maximum number of labels that will be displayed at once. Labels with a higher importance have precedence.
- Map grid density: Number of horizontal and vertical lines (parallels and meridians) that should be displayed.

- Dynamic icon size adjustment: If this option is set the size of document and cluster icons will be dynamically adjusted taking into consideration the current on-screen width and height of the map, in pixels, as well as the number of icons currently displayed (may vary depending on the zoom factor).
- Dynamic label size adjustment: If this option is set the size font used to display the labels will be dynamically adjusted taking into consideration the current on-screen width and height of the map, in pixels, as well as the number of labels currently displayed (may vary depending on the zoom factor).
- Map margin percentage: A margin around the borders of the map, having the size determined by this value and the on-screen size of the map, will be left free. All document and cluster icons are displayed in the map area surrounded by this margin.
- Display background: A switch determining if the map background image is to be displayed.
- Display labels: A switch determining if the labels is to be displayed.
- Display grid: A switch determining if the grid (parallels and meridians) is to be displayed.



Icon size: 3

Label size: 11

Number of region labels: 10

Map grid density: 3

Dynamic icon size adjustment: [✓ Yes] No

Dynamic label size adjustment: [✓ Yes] No

Map margin %: 5.0

Display background: [✓ Yes] No

Display labels: [✓ Yes] No

Display grid: [✓ Yes] No

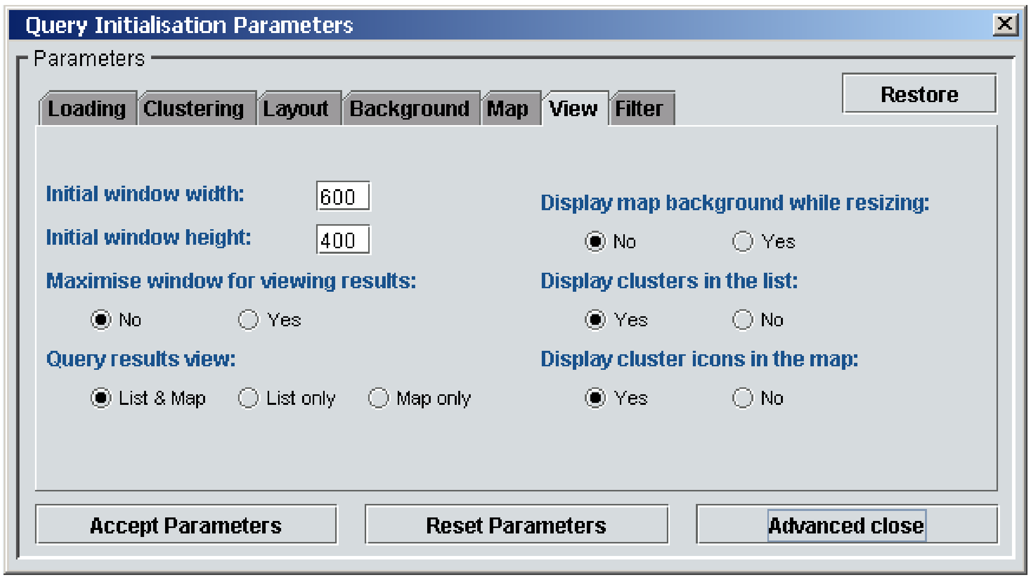
Figure B.5: Map parameters.

B.6 View Parameters

View parameters (see Figure B.6) are general parameters defining how the processing results will be displayed inside their query window.

- Initial window width: This is the width the query window will be resized to when switched to results display mode.

- Initial window height: This is the height the query window will be resized to when switched to results display mode.
- Maximise window for viewing results: If this switch is set to true the query window will be maximised when switched to results display mode.
- Query results view: When the processing of the loaded documents is completed the split pane of the query window can be configured to display the table of contents style list only, the topically organised map view only, or both the list and the together.
- Display map background while resizing: If this option is chosen the map background image will be continuously resized while the query window is being resized by the user. This is recommended only if you are running Visualisation Islands on a fast machine. On slower machines turn this option off.
- Display clusters in the list: If this switch is set to on, clusters are displayed in the list together with the documents forming a table of contents style structure. If this switch is set to off the list will contain the documents in the order as they were returned by the xFIND search engine (which is the order of their estimated relevance). Note, that in this case, the cluster icons will not be displayed in the map.
- Display cluster icons in the map: This switch determines if the cluster icons will be displayed in the map.



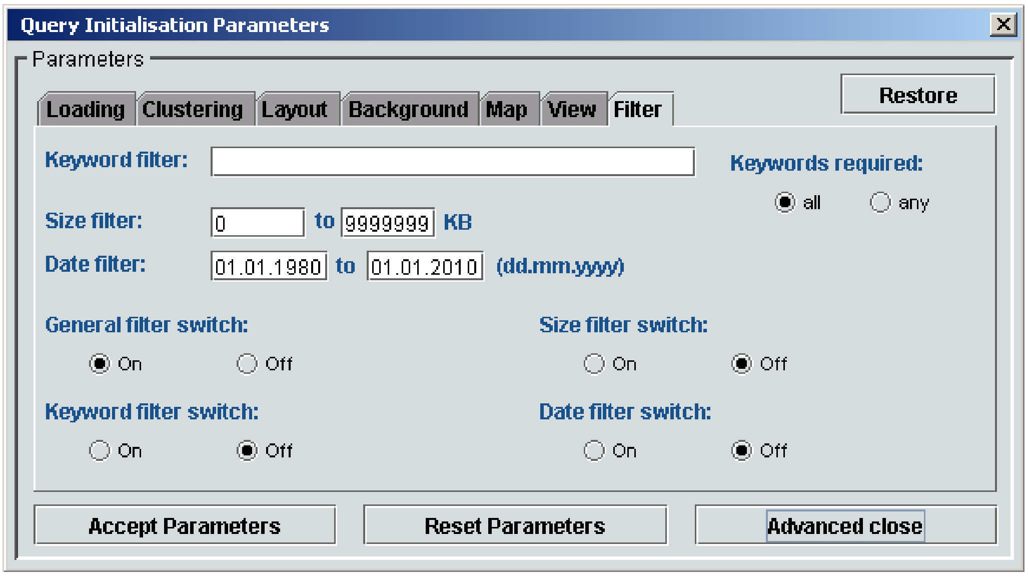
Initial window width: 600
Initial window height: 400
Maximise window for viewing results: [✓ No] Yes
Query results view: [✓ List & Map] List only Map only
Display map background while resizing: [✓ No] Yes
Display clusters in the list: [✓ Yes] No
Display cluster icons in the map: [✓ Yes] No

Figure B.6: View parameters.

B.7 Filter Parameters

This set of parameters (see Figure B.7) determines if and how the document set will be filtered.

- Keyword filter: Enter here the keywords that must be contained by an object. If an object contains the keywords it will be displayed, otherwise it will be hidden.
- Keywords required: Choose if an object must contain all or at least one of the entered keywords to be displayed.
- Size filter: Enter here the size interval that an object's size must be within. If an object's size is within the given interval it will be displayed, otherwise it will be hidden.
- Date filter: Enter here the creation date interval that an object's creation date must be within. If an object's creation date is within the given intervall it will be displayed, otherwise it will be hidden.
- General filter switch: This switch determines if filtering of the object set is performed or not.
- Keyword filter switch: Keyword filter switch can be turned on and off here. Filtering after keywords is performed only if this switch is set to on.
- Size filter switch: Size filter switch can be turned on and off here. Size filtering is performed only if this switch is set to on.
- Date filter switch: Date filter switch can be turned on and off here. Date filtering is performed only if this switch is set to on.



Keyword filter:

Keywords required: [✓ all] any

Size filter: 0 to 9999999 KB

Date filter: 01.01.1980 to 01.01.2010 (dd.mm.yyyy)

General filter switch: [✓ On] Off

Size filter switch: On [✓ Off]

Keyword filter switch: On [✓ Off]

Date filter switch: On [✓ Off]

Figure B.7: Filter parameters.

Bibliography

[AGM01] Keith Andrews, Christian Guetl, Josef Moser, Vedran Sabol, Wilfried Lackner. Search Result Visualisation with xFIND. *In Proc. of Second International Workshop on User Interfaces to Data Intensive Systems (UIDIS 2001)*, Zurich, Switzerland, May 2001. `ftp://ftp.iicm.edu/pub/papers/uidis2001.pdf`

[AKM95] Keith Andrews, Frank Kappe, and Hermann Maurer. Hyper-G and Harmony: Towards the Next Generation of Networked Information Technology *CHI'95 Conference Companion*, Denver, Colorado, May 1995. `http://www.acm.org/sigchi/chi95/Electronic/documnts/demos/ka_bdy.htm`

[AND95] Keith Andrews. Visualising Cyberspace: Information Visualisation in the Harmony Internet Browser. *Proc. of IEEE Symposium on Information Visualization (InfoVis 95)*, Atlanta, Georgia, Oct. 1995. `ftp://ftp.iicm.edu/pub/papers/ivis95.pdf`

[AND96] Keith Andrews. Browsing, Building, and Beholding Cyberspace: New Approaches to the Navigation, Construction, and Visualisation of Hypermedia on the Internet. PhD thesis, Graz University of Technology, Austria, 1996. `ftp://ftp.iicm.edu/pub/keith/phd/`

[BEZ81] J. C. Bezdek. Pattern Recognition With Fuzzy Objective Function Algorithms, 1981. Plenum Press, New York, NY.

[BHJ99] Israel Ben-Shaul, Michael Herscovici, Michal Jacovi, Yoelle S. Maarek, Dan Pelleg, Menachem Shtalhaim, Vladimir Soroka, Sigalit Ur. Adding support for Dynamic and Focused Search with Fetuccino. IBM Haifa Research Laboratory, Dept. of Computer Science Carnegie-Mellon University, Dept. of Electrical Engineering Technion, Haifa, Israel. `http://www8.org/w8-papers/5a-search-query/adding/adding.html`

[BRF98] P. S. Bradley, Usama M. Fayyad. Refining Initial Points for KMeans Clustering. [1] Computer Sciences Department University of Wisconsin Madison, [2] Microsoft Research Redmond. *Proc. 15th International Conf. on Machine Learning.* 1998.

[BSG95] Steve Benford, Dave Snowdon, Chris Greenhalg, Bob Ingram, Ian Knox, and Chris Brown. VR-VIBE: A Virtual Environment for Co-operative Information Retrieval. *Computer Graphics Forum*, 14, (3), pp.349-360, 1995, NCC Blackwell

[CAK97] Jeromy Carrire, Rick Kazman. WebQuery: Searching and Visualizing the Web through Connectivity. [1] Nortel Ottawa, ON Canada, [2] Software Engineering Institute Pittsburgh, PA. 6th International World-Wide Web Conference. 1997. `http://www.cgl.uwaterloo.ca/Projects/Vanish/webquery-1.html`

[CAG90] G. Carpenter, S. Grossberg. ART3: Hierarchical search using chemical transmitters in self-organizing pattern recognition architectures, 1990. Neural Networks 3, 129152.

[CAR01] Cartia Themescape Homepage: `http://www.cartia.com/`

[CHA93] Matthew Chalmers. Using a landscape methaphor to represent a corpus of documents. *In Proceedings European Conference on Spatial Information Theory*, COSIT 93, pages 337-390, Elba, September 1993.

[CHA96] Matthew Chalmers. Adding imageability features to information displays. *In Proceedings UIST96*, Seattle, Washington, November 1996. ACM.

[CHA96a] Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. *In Proceedings Visualization96*, pages 127-132, San Francisco, California, October 1996. IEEE Computer Society.

[CHT95] Avijit Chatterjee. Parallel visual explorer at work in the money markets, 1995. `http://www.ibm.com/news/950203/pve-03.html`

[CHC92] Matthew Chalmers and Paul Chitson. Bead: Explorations in information visualization. *In Proceedings SIGIR92*, pages 330337, Copenhagen, June 1992. ACM.

[CKO97] D. Scott McCrickard, Colleen M. Kehoe. Visualizing Search Results using SQWID. Graphics, Visualization, and Usability Center College of Computing Georgia Institute of Technology. *In Proceedings of the Sixth International World Wide Web Conference*, April 1997. `http://www.cc.gatech.edu/grads/m/Scott.McCrickard/sqwid/Doc/www6.html`

[CKP92] Douglass R. Cutting, David R. Karger, Jan O. Pedersen, John W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. [1] Xerox Palo Alto Research Center [2] Stanford University [3] Princeton University. *In Proceedings of the Fifteenth Annual International ACM SIGIR Conference*, pages 318-329, June 1992. Also available as Xerox PARC technical report SSL-92-02.

[CKP93] Douglass R. Cutting, David R. Karger, Jan O. Pedersen. Constant Interaction-Time Scatter/Gather Browsing of Very Large Document Collections. [1] Xerox Palo Alto Research Center [2] Stanford University. *In Proceedings of the Sixteenth Annual International ACM SIGIR Conference*, 1993.

[CLP97] J. Cugini, S. Laskowski, C. Piatko. Document Clustering in Concept Space: The NIST Information Retrieval Visualization Engine (NIRVE). National Institute of Standards and Technology (NIST) Gaithersburg and Johns Hopkins University. *CODATA Euro-American Workshop on Visualization of Information and Data*, Paris, France, June 1997. `http://zing.ncsl.nist.gov/ cugini/uicd/concept-clusters.html`

[CLS00] J. Cugini, S. Laskowski, M. Sebrechts. Design of 3D Visualization of Search Results: Evolution and Evaluation, *Proceedings of IST/SPIE's 12th Annual International Symposium: Electronic Imaging 2000: Visual Data Exploration and Analysis (SPIE 2000)*, San Jose, CA, 23-28 January 2000. `http://www.itl.nist.gov/iaui/vvrg/cugini/uicd/nirve-paper.html`

[CSB97] E. Churchill, D. Snowdon, S. Benford, P. Dhanda. Using VR-VIBE: browsing and searching for documents in 3d-space. *In the 7th International Conference on Human-Computer Interaction (HCI International'97)*, San Francisco, CA, USA, August 1997

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

[FUH98] Norbert Fuhr. Information Retrieval, Skriptum zur Vorlesung, 1998. `http://ls6-www.informatik.uni-dortmund.de/ir/teaching/courses/ir/`

[GAM98] Christian Guetl, Keith Andrews, Herman Maurer. Future Information Harvesting and Processing on the Web. *Conference European Telematics: advancing the information society*, Barcelona, 4-7 February 1998. `http://www2.iicm.edu/˜cguetl/papers/fihap`

[GRG97] Kenkat N. Gudivada, Vijay V. Raghavan, William I. Grosky, Rajesh Kasanagottu. Information retrieval on the World Wide Web. *IEEE Internet Computing*, 1(5):58-68, September/October 1997.

[GUT00] Christian Gütl . xFIND: Extended Framework for Information Discovery V0.95, Programmers Documentation. Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Austria, 2000

[HAR75] J. Hartigan. Clustering Algorithms, 1975. Books on Demand.

[HAS94] Harvest, Internet Research Task Force Research Group on Resource Discovery (IRTF-RD) at the University of Colorado - Boulder. `http://www.ccu.edu.tw/doc/harvest/hpccbb.html` `http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Searching/schwartz.harvest/schwartz.harvest.html`

[HEA95] M. A. Hearst. TileBars: Visualization of Term Distribution Information in Full Text Information Access. *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, Denver, CO, ACM, May 1995, pp. 55-66.

[HEK97] Marti Hearst, Chandu Karadi. Cat-a-Cone: An Interactive Interface for Specifying Searches and Viewing Retrieval Results using a Large Category Hierarchy. [1] Xerox Palo Alto Research Center, [2] Stanford University. *Proceedings of the 20th Annual International ACM/SIGIR Conference Philadelphia*, PA, July 1997. `http://www.sims.berkeley.edu/ hearst/papers/cac-sigir97/sigir97.html`

[HEM93] Matthias Hemmje. Lyberworld - eine 3D-basierte benutzerschnittstelle fuer die computerunterstuetzte informationssuche in dokumentenmengen. In Der GMD-Spiegel 193, January 1993.

[HEP96] Marti A. Hearst and Jan O. Pedersen. Reexamining the Cluster Hypothesis: Scatter/Gather on Retrieval Results. Xerox Palo Alto Research Center. *In Proceedings of the Nineteenth Annual International ACM SIGIR Conference*, Zurich, June 1996.

[HEP96a] Marti A. Hearst and Jan O. Pedersen. Visualizing information retrieval results: A demonstration of the Tilebar interface. *In Proceedings of ACM CHI 96 Conference on Human Factors in Computing Systems*, pages 394-395, 1996.

[HEX82] P. Heckbert. Color image quantization for frame buffer display. *ACM Trans. Comput. Cr. 16, 3,* July 1982, 297-307.

[HIL68] D. R. Hill. A vector clustering technique. In Samuelson (ed.), Mechanised Information Storage, Retrieval and Dissemination, North-Holland, Amsterdam, 1968.

[HKW94] Matthias Hemmje, Clemens Kunkel, and Alexander Willet. Lyberworld - a visualization user interface supporting fulltext retrieval. German National Research Center for Computer Science (GMD). *In Proceedings SIGIR94*, Dublin, Ireland, July 1994. ACM.

[HYP01] Hyperwave Homepage: `http://www.hyperwave.com/`

[IHT01] Inxight Hyperbolic Tree Technology. Xerox Palo Alto Research Center. `http://www.inxight.com/products/star_tree/overview.html`

[IMU01] Inxight Murax text analysis tool. `http://www.inxight.com/products_sp/murax/index.html`

[IND87] Alfred Inselberg, Bernard Dimsdale. Parallel Coordinates for Visualizing MultiDimensional Geometry. *Proceedings of Computer Graphics International Conference*, 1987.

[IPN01] Information Visualisation at Pacific Northwest National Laboratory, operated for the United States Department of Energy by Battelle Memorial Institute. `http://www.pnl.gov/infoviz/index.html`

[IVD01] Working groups Visualisation of Dataspaces. InfoViz VRML-Document-Finder. Fachhochschule Potsdam and the Daten- und RechenZentrum at the GeoforschungsZentrum Potsdam, `http://fabdo.fh-potsdam.de/infoviz/docfinder.html`

[JAT01] The Java Tutorial, 2001. `http://java.sun.com/docs/books/tutorial/index.html`

[JEN96] John R. Jensen. Introductory Digital Image Processing (2nd ed.). Chapter 8, Thematic Information Extraction: Image Classification, 1996. Upper Saddle River, NJ: Prentice-Hall. pp. 236-238.

[JST01] The JFC Swing Tutorial, 2001. `http://java.sun.com/docs/books/tutorial/uiswing/index.html`

[KCP01] Kansas City Public Library: Introduction to Search Engines, 2001. `http://kclibrary.org/search/srchengines.htm`

[KOD99] Helmut Kopka and Patrick W. Daly. A Guide to LaTeX 2e. Addison-Wesley, third edition, 1999.

[KOH88] T. Kohonen. Self-Organization and associative memory, Springer-Verlag, Berlin, Second Edition, 1988.

[KOH90] T. Kohonen. The self-organizing map, *Proceedings of the IEEE, Vol. 78, No. 9*, 1990, pp. 1464-1480.

[KRO95] U. Krohn. Vineta: visualizing navigational information retrieval. *Paper presented at FADIVA 2, the Second International Workshop on Foundations of Advanced 3D Information Visualization*, July 20-22, 1995, University of Glasgow, Glasgow, Scotland, UK.

[KRO96] Krohn, U. VINETA: Navigation through virtual information spaces. *Paper presented at AVI '96, Advanced Visual Interfaces: An International Workshop*, May 27-29, 1996, Gubbio, Italy.

[LAA99] Bjornar Larsen and Chinatsu Aone. Fast and Effective Text Mining Using Linear-time Document Clustering. *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining.* 1999.

[LAR94] John Lamping, Ramana Rao. Laying out and visualizing large trees using a hyperbolic space. *In Proceedings UIST94*, pages 13-14, Marina del Rey, California, November 1994. ACM.

[LEA00] Anton Leuski and James Allan. Lighthouse: Showing the Way to Relevant Information. *In the Proceedings of IEEE Symposium on Information Visualization 2000 (InfoVis 2000)*, Salt Lake City, Utah, October 9-10, 2000. pp. 125-130.

[LIG01] Lighthouse homepage: `http://toowoomba.cs.umass.edu/ leouski/ lighthouse/`

[LRP95] John Lamping, Ramana Rao, Peter Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. *In Proceedings CHI95*, pages 401-408, Denver, Colorado, May 1995. ACM.

[LTK94] Thomas M. Lillesand, Ralph W. Kiefer. Remote Sensing and Image Interpretation (3rd ed.). Minimum Distance to Means Classifier in Chapter 7, Digital Image Processing, 1994. New York: John Wiley and Sons. pp. 590-591.

[LUH59] Hans Peter Luhn. Auto-Encoding of Documents for Information Retrieval Systems. Pergamon Press, London, 1959.

[LYB01] Demos of the LyberWorld Project: `http://www.darmstadt.gmd.de/˜hemmje/Activities/Lyberworld/demos.html`

[LYB01a] The LyberWorld Homepage: `http://www.darmstadt.gmd.de/˜hemmje/Activities/Lyberworld/`

[MAU96] Hermann Maurer. HyperWave: The Next Generation Web Solution. Addison-Wesley, May 1996. `http://www.iicm.edu/hgbook`

[MAY97] Susanne Mayr. Masters Thesis: SearchVis, Visualising Search Result Sets Using a Force-Based Method to Form Clusters of Similar Documents. Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Austria, 1997. `ftp://ftp.iicm.edu/pub/theses/smayr.pdf`

[NPM97] David A. Nation, Catherine Plaisant, Gary Marchionini, and Anita Komlodi. Visualizing websites using a hierarchical table of contents browser: WebTOC, 1997. University of Maryland.
`http://www.cs.huji.ac.il/˜sdbi/1999/elik/nation.html`

[NFF96] Lucy Terry Nowell, Robert K. France, Edward A. Fox. Visualizing search results with Envision. *In Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, System Demonstrations: Abstracts, pages 338-339, 1996.

[NFH96] Lucy Terry Nowell, Robert K. France, Deborah Hix, Lenwood S. Heath, Edward A. Fox. Visualizing search results: Some alternatives to query-document similarity. *In Proc. SIGIR96*, pages 67-75, Zurich, Switzerland, August 1996. ACM.

[NFH96a] Lucy Terry Nowell, Robert K. France, Deborah Hix, Lenwood S. Heath, Edward A.Fox. Visualizing search results: Some alternatives to query-document similarity. *In Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Visualization, pages 6775, 1996.

[NIR01] NIRVE Homepage: `http://zing.ncsl.nist.gov/˜cugini/uicd/nirve-home.html`

[OKS93] Kai A. Olsen, Robert R. Korfhage, Kenneth M. Sochats, Michael B. Spring, James G. Williams. Visualisation of a Document Collection: The VIBE System, Information Processing and Management, 29 (1), 6981, 1993.

[RHS97] Michael Reed, Dan Heller, Ben Shneiderman. Online library of information visualization environments, 2001. `http://www.otal.umd.edu/Olive`

[RIJ79] C. J. van Rijsbergen. Information Retrieval. Butterworth and Co Ltd., London, second edition, 1979.

[RMC91] George G. Robertson, Jock D. Mackinlay, Stuart K. Card. Cone trees: Animated 3D Visualizations of Hierarchical Information. *In Proceedings CHI91*, pages 189-194, New Orleans, Louisiana, May 1991. ACM.

[RMC94] George G. Robertson, Jock Mackinlay, Stuart K. Card. Display of Hierarchical Three-dimensional Structures with Rotating Substructures. US Patent 5,295,243, Xerox Corporation, March 1994. Filed 29th Dec. 1989, granted 15th March 1994.

[ROC66] J. J. Rocchio. Document retrieval systems - optimization and evaluation. Ph.D. Thesis, Harvard University, 1966.

[RUS69] E. H. Ruspini. A new approach to clustering, 1969. Inf. Control 15, 22-32.

[SEI84] S. Z. Selim, M. A. Ismail. K-means-type algorithms: A generalized convergence theorem and characterization of local optimality. *IEEE Trans. Pattern Anal. Mach. Intell. PAMI-6, 1*, 1984, 81-87.

[SES01] Search Engine Showdown, 2001. `http://searchengineshowdown.com`

[SPI01] SPIRE - Spatial Paradigm for Information Retrieval and Exploration. Pacific Northwest National Laboratory. `http://www.pnl.gov/infoviz/spire/spire.html`

[SEW01] Search Engine Watch, 2001.
`http://www.searchenginewatch.com`

[SWC01] The Swing Connection.
`http://java.sun.com/products/jfc/tsc/index.html`

[SYW75] Salton, G., Yang, C. and Wong, A. A vector space model for automatic indexing. *Communications of the ACM , 18 (11)* 613-620. 1975.

[TOG74] Julius T. Tou, Raphael C. Gonzales. Pattern Recognition Principles. A Simple Cluster-Seeking Algorithm in Chapter 3, Pattern Classification By Distance Functions, 1974. Reading, MA: Addison -Wesley. pp. 90-92.

[TOG74a] Julius T. Tou, Raphael C. Gonzales. Pattern Recognition Principles. Isodata Algorithm in Chapter 3, Pattern Classification By Distance Functions, 1974. Reading, MA: Addison-Wesley. pp. 97-104.

[VHN96] A. Veerasamy, S. Hudson, S Navathe. Querying, navigating and Visualizing an Online Library Catalog, GVU Tech report. 1996.

[VSM01] Visual SiteMap, Drexler University.
`http://faculty.cis.drexel.edu/sitemap/`

[VOO86] E. M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. Information Processing and Management, vol. 22, 1986, pp 465-476.

[VRV01] VR-VIBE screen shots. `http://www.emptiness.org/vr/vrvibe.html`
`http://www.crg.cs.nott.ac.uk/research/technologies/visualisation/vrvibe/`

[WDB95] Andrew Wood, Nick Drew, Russell Beale, Bob Hendley. HyperSpace: Web Browsing with Visualisation. School of Computer Science of The University of Birmingham.
`http://www.igd.fhg.de/archive/1995_www95/proceedings/posters/35/`

[WDB95a] A. Wood, R. Beale, N. Drew, R. Hendley. HyperSpace: A World-Wide Web Visualiser and its implications for Collaborative Browsing and Software Agents. HCI'95, UK.

[WEI99] Erwin Weitlaner. Masters Thesis: Metadata Visualisation Visual Exploration of File Systems and Search Result Sets based on Metadata Attributes. Institute for Information Processing and Computer Supported New Media (IICM), Graz University of Technology, Austria, 1999. `ftp://ftp.iicm.edu/pub/theses/eweit.pdf`

[WET01] WebTOC Homepage: A Tool to Visualize and Quantify Web Sites using a Hierarchical Table of Contents. University of Maryland Human-Computer Interaction Laboratory.
`http://www.cs.umd.edu/hcil/webtoc/`

[WIL88] P. Willet. Recent Trends in Hierarchic Document Clustering: A Critical Review. Information Processing and Management, 24(5):577-597, 1988.

[WIT85] X. Wu, I.H. Witten. A fast k-means type clustering algorithm. Dept. Computer Science, Univ. of Calgary, Canada, May 1985.

[WLM01] Wilmascope homepage: `http://www.wilmascope.org`

[WMP01] WebMap Homepage: `http://www.webmap.com/`

[WSM01] WEBSOM - Self-Organizing Maps for Internet Exploration.
`http://websom.hut.fi/websom/`

[XFI01] xFIND Homepage: xFIND - eXtended Framework for Information Discovery, 2001.
`http://xfind.iicm.edu`

[YOU96] Peter Young. Three Dimensional Information Visualisation. Visualisation Research Group, Centre for Software Maintenance, Department of Computer Science, University of Durham. 24th March, 1996, Revised: 1st November 1996, Computer Science Technical Report, No. 12/96.

[ZAE98] Oren Zamir and Oren Etzioni. Web Document Clustering: A Feasibility Demonstration. Department of Computer Science and Engineering University of Washington Seattle. *In ACM SIGIR 98*, pages 46–54, 1998.
`http://www.cs.washington.edu/research/clustering`

[ZAE99] Oren Zamir and Oren Etzioni. Grouper: A Dynamic Clustering Interface to Web Search Results. Department of Computer Science and Engineering, University of Washington. *Proceedings of the Eighth International World Wide Web Conference, Computer Networks and ISDN Systems*, 1999. 66
`http://www.cs.washington.edu/research/clustering`