INSTITUT FÜR INFORMATIK

UND PRAKTISCHE MATHEMATIK

A Software Agent for Adaptive Navigation Support in a Restricted Internet Area

Dirk Kukulenz

Bericht Nr. 0211
Dec. 2002

CHRISTIAN-ALBRECHTS-UNIVERSITÄT

KIEL

Institut für Informatik und Praktische Mathematik der
Christian-Albrechts-Universität zu Kiel
Olshausenstr. 40
D – 24098 Kiel

A Software Agent for Adaptive Navigation Support in a Restricted Internet Area

Dirk Kukulenz

Bericht Nr. 0211
Dec. 2002

e-mail: dku@ks.informatik.uni-kiel.de

Dieser Bericht enthält die Dissertation des Verfassers

Abstract

This thesis deals with the development of a software system that helps a user to search for information in the World Wide Web. The particular problem considered here is support in a well-defined, restricted Web area. Two support strategies are considered. One strategy is to present a visitor views of a local hyperlink structure depending on the current position in hyper-space and previous navigation decisions. Main partial problems to realize such a support are dealt with, like the registration of user behavior, the registration of information about the Web area and the presentation of support information on the client side. In contrast to similar systems, the developed system may be applied by a large fraction of Internet users instantly. The only requirement on the client side is Java support by the browser.
The second considered support strategy is an estimation of the relevance of data objects and sequences in the Web for a specific client. This estimation is based on the client's previous navigation behavior and registered navigation behavior of other users (collaborative filtering). The approach to estimate relevant data objects in this thesis is to predict a user's future data requests. For this purpose the presented system stores user information on the server side. User behavior is modeled by graphs, consisting of nodes representing requested data objects and edges representing transitions. A new method is presented to predict future navigation steps that is based on a distribution estimation of registered graphs and a classification of a new (partial) navigation profile with regard to the estimated distribution. The different steps of the presented algorithm are evaluated using generated and observed profiles.

1. Gutachter: Prof. Dr. Gerald Sommer

2. Gutachter: Prof. Dr. Gerhard Weber

Datum der mündlichen Prüfung: 30.10.2002

Contents

1 Introduction **7**

2 Internet Technology, Basics **13**
2.1 Technical basics of Internet communication 13
2.1.1 From databases to the Internet 13
2.1.2 Client/server communication, protocols 15
2.1.3 Basic software applied for Internet communication . . . 16
2.1.4 Languages used for Internet communication 17
2.2 The basic problem: perception, search and orientation 18
2.2.1 Human perception, parallel and sequential 18
2.2.2 Search strategies in the Internet 19
2.2.3 Problem description 21
2.3 Similar projects 22
2.4 Fundamental technical problems 26
2.4.1 Static analysis of Internet information 26
2.4.2 Registration of the use of the Internet 28
2.4.3 Information visualization in the Internet 30
2.5 Summary: Internet Technology, Basics 31

3 Internet Agent for Adaptive Hyper Space Visualization **33**
3.1 Objective of the software system 33
3.2 Possible architectures for navigation support tools 34
3.3 Agent description: survey 37
3.3.1 Principal software structure 38
3.3.2 Comparison to other architectures 40
3.4 Basic concepts 42
3.4.1 Static analysis of hyperlink structures 42
3.4.2 Registration of a navigation process in a distributed Web environment 43
3.4.3 Comparison of server-side registration methods 48
3.4.4 Visualization of navigation support information 52

3.4.5 Control of a navigation process . . . 53
3.5 Components and realization . . . 56
3.5.1 The 'trace server' . . . 56
3.5.2 The client's applet . . . 58
3.5.3 Technical realization and problems . . . 59
3.6 Visualization examples . . . 60
3.7 Summary and discussion: Internet Agent . . . 69

4 Pattern Recognition in Graph Spaces 71
4.1 Model definition and notations . . . 71
4.2 Distortion concept . . . 73
4.3 Formal introduction into graph spaces . . . 76
4.4 Methods to define a distance between graphs . . . 81
4.4.1 Topological-index methods . . . 81
4.4.2 Set distance . . . 81
4.4.3 Shape metrices . . . 82
4.4.4 String-edit distance function . . . 83
4.4.5 Contents-based distance function . . . 84
4.5 Characterization of a graph distribution . . . 84
4.5.1 Discrete characterization . . . 85
4.5.2 Topological characterization . . . 85
4.6 Estimation of graph distributions . . . 86
4.6.1 Clustering techniques . . . 86
4.6.2 Determination of a cluster center . . . 87
4.6.3 Quality of a distribution estimation . . . 88
4.7 Summary: Pattern Recognition in Graph Spaces . . . 89

5 Estimation of Graph Distributions 91
5.1 Task description . . . 91
5.2 Simulation procedure . . . 92
5.3 Effect of the information used for segmentation . . . 99
5.3.1 Dependence of distribution estimation on the number of profiles . . . 99
5.3.2 Impact of distortions . . . 100
5.3.3 Effect of topological information . . . 101
5.4 Non-stationary distributions . . . 103
5.5 Summary: Estimation of Graph Distributions . . . 105

6 Classification and Prediction of User Profiles 107
6.1 Prediction methods . . . 107
6.2 A new prediction algorithm . . . 109

6.3 A measure for the prediction quality 112
6.4 Experiments with simulated data 114
6.4.1 Different classification functions 114
6.4.2 Convergence of the estimation 116
6.4.3 Estimation of the clusters number 120
6.5 Application to registered navigation profiles 126
6.6 Summary: Classification and Prediction of User Profiles 133

7 Summary and Conclusions **135**

A Graph Matching Algorithms **141**
A.1 A maximal-common-subgraph algorithm for graphs by clique detection 142
A.2 Error-correcting subgraph isomorphism detection 144

B Clustering Algorithms **149**

C Preparation of Registered Access-log Data **151**

Chapter 1

Introduction

The Internet offers a huge amount of valuable information in almost every field of human knowledge. The disorder of the information, especially the distributed nature of the Internet, makes it often difficult and requires much time to find relevant information and to satisfy complex information needs. The more the amount and the disorder of information increases, the more it is necessary to develop powerful and intelligent systems that support users and make it possible to find information easily.
Examples for such systems are search engines, providing the possibility of a query search. Search engines are frequently applied by almost every user in the Internet for information search. Other systems to support users when looking for information are developed in the field of (intelligent Internet) information agents [58]. Information agents follow various strategies to support information search. Some of these systems perform an automatic retrieval and the working up of information (automation). Other systems apply different learning strategies in order to find relevant data objects for a user. The learning strategies in this context can roughly be divided into two groups. One group of systems learns from a (single) client's behavior, a process which is called *content-based learning*. Another learning strategy is to consider the behavior of many users, which is called *collaborative filtering* or *learning* [120]. The different types of software systems that help users to find information in the Internet apply various strategies to present support information. Search engines present as a rule lists of hyperlinks to web pages, ordered by the estimated relevance for a user. Other systems help a user e.g. to orientate himself in hyper-space by showing his navigation history or by visualizing maps of the hyperlink structure (similar to landscape maps in the real world). Some systems estimate the relevance of data objects and present this estimation e.g. by changing the appearance of existing web pages or by assembling new pages [81]. Some of these systems will be described in detail in the next

chapter (section 2.3).
These systems and the applied techniques are developed in different fields of computer science like information retrieval, Internet research, agent technology, artificial intelligence and software engineering. Various mathematical techniques are applied in this problem field, coming e.g. from algebra, statistics, e.g. clustering and stochastic processes, and graph theory.

The problem of supporting a user who looks for information in the World Wide Web ('WWW' or 'Web') is the main topic of this thesis. The thesis deals with the development of an intelligent information agent that makes it possible on the one hand to visualize (parts of) the hyperlink structure of the Web. The purpose of this visualization is to help a user to orientate himself in the data structure. On the other hand methods are developed to estimate relevant Web objects for a specific user based on collaborative filtering. There are two main questions that are examined.
The first part of the thesis deals with technical aspects of the development of the software system. One important question is to find an adequate software architecture. Web communication depends on processes on different participating computers, like client and server processes. The new software system has to fit into this existing environment and has to work together with the respective programs. Existing support systems apply different architectures with different advantages and disadvantages, which will be compared to each other.
One important aspect concerning the technical development is the possibility to automatically register user behavior. This registration is necessary to provide a support, that is adapted to the specific situation and the interests of one user. There are two main questions concerning user behavior registration. One question is the knowledge, a system may acquire about the behavior of a single user. Another question is the possibility to collect this knowledge of different users in a central place, which is necessary to make collaborative filtering possible. The objective is to collect as much information as possible about a single user and to be able to store this information on a central server. In order to provide an online support, the registration has to be done in real-time.
A second important aspect concerning the technical development is the possibility for the system to acquire knowledge about the data in the Web for support purposes. There are two possible strategies. A support system may on the one hand try to offer a support for information search in the whole Web. This task, performed e.g. by some search engines, is very extensive. The information that can be acquired by the system about the Web data can only be very rough, and requires much effort and many resources. On

the other hand a system may offer a support only for information search in a small fraction of the Web, e.g. the servers of a university or company. In this case the information that may be acquired about the web data can be much more detailed and can easily be extracted from the Web. In this thesis the second strategy is chosen and methods will be shown to extract local Web information.
A third important aspect concerning the technical development is the presentation and visualization of support information. Internet browsers provide the possibility to visualize web pages and make it therefore possible to show support information on web pages. This method is used by some Internet agents (section 2.3). However, in order to apply more advanced visualization techniques, the visualization capabilities of Internet browsers have to be extended. There exist different methods to do so, like browser plug-ins or separate software installations on the client side (section 2.1.3), that will be compared to the applied method in this thesis, which is based on Java applets.
The result of the technical considerations and the development of the software system as presented in the first part of this thesis is a software agent, that registers user behavior (in our case the set of browsing decisions) in real-time and makes it possible to store this information on a central server. The system is capable to acquire knowledge about the web sites where the support shall be active. And the system makes it also possible to extend and to apply the presentation techniques on the client side.
These attributes of the presented system are applied for one specific support problem, namely the orientation of a user in a Web area. The system makes it possible to present a client's navigation history (which is adaptive with respect to the previous navigation steps) and makes it also possible to show hyperlink maps of the local Web structure[1] (which are adaptive with respect to the position of a user in hyper-space and the hyperlink structure). The problem of losing orientation in hyper-space is well known in literature and often refered to as the 'being lost in hyper-space' problem [31]. The system presented in the first part of this thesis shows one way to help to avoid this problem.
The second part of the thesis deals with one specific question concerning the development of intelligent Internet agents. How can the relevance of data objects in the Web and sequences of such data objects be estimated for a specific user? The motivation to develop such methods for a relevance estimation with respect to the system presented in the first part of the thesis

[1]Only static hyperlink structures are considered. The implementation considers only HTML pages with hyperlinks marked by the '< A'-tag.

is the fact, that the presented hyperlink maps (and maps in general) can be greatly improved, if only the objects and routes are shown that are relevant for a specific user.
There are different possibilities to perform such an estimation known from literature, like Markov learning, reinforcement learning or clustering procedures (section 6.1). The best known method is applied by search engines, applying methods known from information retrieval, e.g. based on linear algebra and statistics. In this thesis, a new method is introduced to perform such a relevance estimation that is based on graph theory. The reason to consider graphs is the fact, that the developed software system makes it possible to register sets of navigation (browsing) steps of a user. These sets can be regarded as graphs with the nodes representing the requested data objects and the edges representing the activated hyperlinks.
It is difficult to measure the relevance of information objects for a specific user directly and therefore the problem is put slightly different. How can actions of a user, like hyperlink activations, be predicted? We assume that the prediction of data objects and sequences is one possible approach to try to estimate the relevance of data objects and sequences.[2] As will be shown, it is possible to develop simple measures to estimate the quality of a prediction procedure, which is necessary to evaluate the developed prediction algorithm.
There are different steps necessary for the presented prediction algorithm as described in the second part of the thesis. A first problem is the characterization of a distribution of graphs in a space of graphs. The task here is not only to consider relative frequencies of graphs, but also to consider topological aspects of the graph space in order to deal with distortions of graphs. Then methods have to be developed to estimate such a distribution based on registered graphs.
In order to predict new actions of a user based on his previous navigation behavior, the estimated distribution can be used to classify the new registered graph. The classification result can then be used for the prediction process as described in detail in the thesis.
The different steps of the new prediction procedure are tested, using generated graphs that follow a well-known distribution and by applying different quality measures. One specific estimation problem, namely the estimation of the number of clusters in a graph distribution is examined in more detail. Finally, the prediction procedure is applied to graphs registered by an Internet server and the results are discussed.

[2]This assumption is only fully correct, if a user knows the web sites already very well and therefore makes only navigation decision that are optimal with respect to the relevance.

The aim of the research projects in the first and the second part of the thesis is to lay a basis for Web search support systems that provide a user with maps of the hyperlink structure that are adapted to his specific interests. Such systems are likely to be much more valuable than the structure views presented in the first part of this thesis. However much more research has to be done concerning the prediction process. The second part of the thesis provides first main ideas and results to deal with this problem.

The structure of the thesis is as follows. In the **second chapter** the problem of information search in the Internet will be described in more detail by comparing the Internet to a common stand-alone database. Technical aspects of Internet (WWW) communication and necessary terms are introduced. The support strategy, that is examined in this thesis, is motivated and the chapter gives a rough survey of well-known systems that are similar to the one presented here and that will be refered to in subsequent chapters. Partial technical problems, some of which were examined in other research areas and thus can be applied for the presented system, will shortly be described.
In the **third chapter**, the technical development of the presented Web agent is described. At first, the architectures of similar software systems are described. The architecture of the new software system is presented and compared to the architectures of the similar systems. The main ideas and the technical realization of the new software system is then described in detail and support presentation examples are given. The presented software system makes it possible to register user behavior on the server side and thus to collect information about user behavior on a central server.
The registered type of information concerning user behavior is used in the **fourth chapter** to develop a user model. User behavior is modeled by sets of navigation decisions. These data can be regarded as graphs, and therefore the chapter presents necessary definitions known from graph theory. The chapter describes the problem of finding a distribution of graphs in a graph space. Different characterizations of graph distributions are introduced to represent the 'shape' of a graph distribution. These distribution characterizations are based on distance (or similarity) functions between graphs, methods to describe groups of similar graphs and the shape of these groups. At the end of the chapter, a procedure to estimate a graph distribution according to a distribution characterization is presented. The graph distribution estimation is based on a common nearest-neighbor clustering technique that will be described in more detail in appendix B. The chapter finally presents a measure for the quality of a distribution estimation, necessary for the experiments.
Chapter five presents experiments to evaluate this distribution estimation process. At first, a method is described to generate graph data with a previ-

ously fixed distribution. This simulation procedure is applied to show some properties of the estimation process, like convergence for an increasing number, the impact of noise, the impact of topological information etc.
The **sixth chapter** deals with a new technique to predict a user's navigation steps and sets of navigation steps. At first, the new prediction algorithm is described, which is mainly based on a classification of graphs according to an estimated graph distribution. The measure for a distribution estimation quality presented in chapter 4 can't be applied for observed data, because in this case the distribution is not known. Therefore a new quality measure for the prediction process is presented, that can be applied for simulated and for real data. The new prediction algorithm is first applied to simulated data in order to show basic properties (and the correctness) of the algorithm. Then the algorithm is applied to observed profiles. Estimation results like estimations of the number of clusters are presented and discussed.
The thesis concludes with a summary of the developed techniques and the results and gives an outlook in **chapter 7**.
The appendix consists of three parts. In the first part, **appendix A**, the problem of graph matching, which is well-known from the field of pattern recognition, is introduced, the current state of research is described and the algorithms which are applied in the thesis are presented. **Appendix B** gives a short introduction to hierarchical cluster algorithms and presents the algorithm that is used for the distribution estimation. **Appendix C** describes necessary modifications in order to apply the presented estimation procedures to access-log files, registered by web servers.

Chapter 2

Internet Technology, Basics

2.1 Technical basics of Internet communication

In order to make characteristic problems of users and information providers in the Internet clear, it may be helpful to regard the Internet as a (global) database and to compare it to common (stand-alone) database technology. This comparison is dealt with in the first part of this section. Then the main technical requirements and ideas of Internet communication are presented and the necessary terms are introduced.

2.1.1 From databases to the Internet

Information stored in computer memory is called data. If the data is a collection of related facts about some topic, it is called a *database* [73]. Databases operate on so-called entities and entity relationships. An entity is a particular object in the problem domain that has characteristic properties or attributes. There may exist relationships between entities, usually modeled by labeled associations between entities. The choice of entities, entity attributes and relationships depends on a specific problem domain. In order to create and to use a database software, packages have been developed called *database management systems* (DBMS). The main idea of database management is to separate a database structure from its contents. The database structure is also called *database schema*. A DBMS usually provides on the one hand a *data description language* (DDL) in order to define database schemes, and on the other hand a *data manipulation language* (DML) to manipulate data objects and relationships with respect to the database schemas.
A method to organize data in a database has to take two problems into ac-

count. One is to define rules according to which data are structured and the second is to define permitted operations. Such a method is called (Logical) *data model*. It is usually expressed and encapsulated in a DDL and the DML.
One approach to data modeling is the *relational data model* [100]. The term *relation* refers to a two-dimensional table of data that represents a particular entity. Each row, referred to as tuple, is a collection of values about a particular individual of that entity. The term *domain* refers to a set of values of the same kind. The same domain being present in different relations makes it possible to represent relationships between entities. In order to define the structure of a relational database and to retrieve and maintain data, different relational languages have been developed like SQL (structured query language) or QBE (query by example).
A second approach to data modeling is the *object-oriented data model*, used by object-oriented databases [50]. The principal components of such a database are abstract data objects (ADOs), that make it possible to organize more complex data than the relational data model, e.g. multimedia data. The query languages in this case are much more complex, since they strongly depend on member functions provided by the ADOs themselves.
From a client's point of view, the Internet can in some respect be regarded as a database, too, although there exist striking differences to common standalone databases. It is difficult to find a finite number of entities in the case of the Internet. The information is usually located in multimedia data, either in data objects of one single type, like images, text or sound files or larger data objects, containing these single data objects, like *web pages*. The data objects neither fit into a relational nor into a simple object-oriented data model. The Internet may however also contain relational or object-oriented databases, if they are connected to it.
In contrast to a relational or an object-oriented database, it is relatively easy to provide information in the Internet, since no database scheme has to be taken into account. In contrast to an object-oriented database, no retrieval member functions have to be developed to make information available (retrievable). On the other hand the problem is, that there exist no query languages, which makes it difficult to retrieve information. In some respect, the easiness to provide information in the Internet results in a huge effort to retrieve (relevant) information from the Internet.
One most important way to make retrieval of information in the Internet possible is provided by so-called *search engines*. These software systems analyze the contents of web pages and build indexes that represent the contents of pages. The query of a user is sent to the search engine server and is processed with the help of those indexes [10].
The general problem of finding information in the Internet is the main topic

of this thesis. However in contrast to (global) search engines, only a well-defined part of the Internet will be considered, which will be called *Internet area* or *environment*.
A (distributed) Internet area or environment is the data available in the Internet that are located on a (small) number of Internet servers (section 2.1.2), e.g. the servers of a university or of an enterprise.
The term 'distributed' emphasizes, that the data objects may be located on different servers. Those servers should be physically near to each other, as explained in the next chapter, and they should provide information relevant for a certain topic like science, economy/business etc.

2.1.2 Client/server communication, protocols

In the Internet there exist two kinds of participants, users (or clients) who usually request information, and those participants who provide information. The respective computers of those participants will be called the client and server computers. From a technical point of view the communication takes place between processes on the respective computers, a client and the server process. In order to make a communication between computers with different operating systems possible, a common network model has to define, how the exchange of data takes place. In the case of the Internet, the TCP/IP layer model is applied, which is similar to the ISO/OSI model [97]. The protocols TCP and IP (Transmission Control Protocol / Internet Protocol) [85], [86], [105] make a transport of data packages from one computer to another computer through a local network or through the Internet possible (network / transport layer). A communication at this level would be very complex; therefore protocols at a higher level are applied to capture, from the client's point of view, the capabilities of a server by the description of message exchanges (requests and responses)[73]. Common protocols that define a communication at this application layer are HTTP (HyperText Transfer Protocol) [35], FTP (File Transfer Protocol) [87] and others. All resources and users on the Internet that use the HTTP protocol are denoted as *World Wide Web* ('WWW' or 'Web'). This is the part of the Internet that is considered in this thesis. A URL (Uniform Resource Locator) can be viewed as a means to uniquely identify resources on the net.

Definition 1 *A URL (Uniform Resource Locator) is the address of a file (resource) accessible on the Internet. The type of resource depends on the Internet application protocol. Using the World Wide Web's protocol (HTTP), the resource can be an HTML page, an image file, a program such as a common gateway interface application or Java applet, or any other file supported by*

HTTP (section 2.1.4). The URL contains the name of the protocol required to access the resource, a domain name that identifies a specific computer on the Internet, and a hierarchical description of a file location on the computer or a query string[1].

A URL is a vital part of any HTTP request, since it may e.g. contain the information about which data object is requested. A second component of a HTTP request is the *MIME content-type* [46]. The HTTP header contains information about the type(s) of the contained data.
As far as the presented system in this thesis is concerned, only the HTTP protocol is considered and HTML data[2] (section 2.1.4 and 3.5.3). Requests addressed to CGI scripts or requests for Java applets or XML data etc. (section 2.1.4) are not considered.[3] 'Fragment identifiers' [46] referencing a particular location within a resource are also not considered. Thus in this thesis a URL usually denotes the address of an HTML page or of an image in the WWW.
In the following text the term *data object* will be used to denote data that are addressed by a URL (e.g. a web page, an image etc.) and that are not part of another data objects (e.g. an image being contained in a web page).

2.1.3 Basic software applied for Internet communication

Communication in the Internet is made convenient (and therefore popular) by means of different software systems that have been developed in the last decades and that are frequently changed and further developed. These software systems are a required basis for the presented Internet agent in this thesis and will therefore shortly be described.
An *Internet browser* (short *browsers*) is the process, a user can start in order to (easily)

- submit (HTTP-) requests for documents in the Internet
- view documents or other data objects, execute programs
- send information to a server

The third item is necessary to send e.g. a password or business-related data like a bank account to a server. There exist several of such programs that

[1] e.g. a Common Gateway Interface (CGI) query string
[2] and some other MIME-types like GIF-files, JPEGs etc.
[3] Some of the results may however be extended to these data types.

are widely spread, like Internet Explorer [76], Netscape [13] or Opera. In order to provide information, Internet servers [72] like IIS, Apache [65] are applied. These programs make is possible to

- accept and process requests from clients
- fetch and send the requested data back to a client
- communicate with other programs for more complex procession[4], like databases etc.

Plug-in applications are programs that can easily be installed and used as part of a web browser. Usually plug-ins define supplementary programs, e.g. in order to play sounds or motion videos.
In the connection between client and server different programs may be applied like *proxy servers* and *firewalls*. Proxy servers [37] make it possible to redirect requests to another server. It may be used e.g. as a gateway, linking a local network to the Internet. Proxy servers may also store requested data objects temporarily (*caching*) in order to shorten the response time, which will be an important aspect later on. *Firewalls* are programs that are often located on gateways in order to make the Internet communication safer [84].

2.1.4 Languages used for Internet communication

The Internet protocols mentioned above make a communication between client and server process through the net possible. A further question is, how to impart information. HTML (HyperText Mark-up Language) [52] and XML (Extensible Markup Language) [48] are languages that make a presentation of multimedia information for the user's browser possible.[5] One of the powerful presentation techniques of HTML is the *frame* concept which will be important for the technical realization of the described software agent (section 3.4.5). Frames make it possible to divide the visualization area of the browser into several segments. Each segment may visualize a certain data object. Other developments like Java Script [40] or Java [39] make it even possible for a server to run small programs on the client side, supervised by the browser software. It is obvious that this technique may contain many security risks. Therefore the rights of these programs (e.g. Java applets) are restricted. Java applets run in a so-called *sand-box* on the client side [26], [38]. They are e.g. not allowed to write data persistently on the client

[4]This is usually implemented using the common gateway interface (CGI) [46].
[5]For XML a special browser is required [45].

computer and they must not read files.[6] Most languages make it possible for a user to request new data objects by activating *hyperlinks*.

Definition 2 *A hyperlink is a special area on a web page which can be activated (usually with a mouse). The hyperlink can appear as text (anchor text) or graphics. A hyperlink contains a (usually invisible) URL address, which is the target address of the HTTP request that is initialized by the browser when the hyperlink is activated by a user.*

Most hyperlinks take a user to another web page or perform special functions, like submitting a form, accessing an ftp site or execute a database query. In this thesis only hyperlinks on HTML pages marked by the $< \mathsf{A} >$ and $< /\mathsf{A} >$ tags are considered [46] (the restrictions concerning URL addresses in hyperlinks were described in section 2.1.2).

2.2 The basic problem: perception, search and orientation

2.2.1 Human perception, parallel and sequential

When discussing the development of a software system that makes it easier to find electronically stored information, it may be helpful to take into account how humans perceive information. One characteristic concerning e.g. the visual and the contact sense is the fact that images or touches are processed in a parallel manner. However there exists a minimal resolution in time, e.g. about 40 msec for visual impressions and 10 msec for acoustic impressions. Impressions that follow each other in a shorter time, can't be separated by a human in time and will be perceived as one single impression. Therefore there exists a minimal time interval $\Delta \mathtt{t}_{\mathtt{min}}$,which satisfies the following:
Every two impressions perceived by a human on an arbitrary sense occurring in a time interval smaller than $\Delta \mathtt{t}_{\mathtt{min}}$ *will be perceived as (at most) one impression and can't be separated in time.*
The amount of information, a human can process in this time interval is finite; it is limited e.g. by the finite number of respective perception cells. The information a human perceives in the time interval $\Delta \mathtt{t}_{\mathtt{min}}$ will be refered to as a *perception object*. The information a system provides in this time interval will be refered to as an *impression object*. Human cognition can therefore be regarded as the processing of a sequence of perception objects, that will be called *perception* or *impression sequence*. If a human wants

[6] A user may assign applets more rights.

to acquire information, it may be the case that it is possible to satisfy the information need with one impression, e.g. an image that is shown for a short time. In most cases it will be necessary to provide him with a sequence of impressions. One example is a text, that can (and has to) be processed in a sequential manner by reading it, although a single letter is processed in a parallel way.
The result of this consideration is that no matter what information need a human has, a software system always has to provide or generate a sequence of information objects, an impression sequence (with length ≥ 1). In many cases the information need can be satisfied by textual data that provide such impression sequences without any active sequence generation by the software system.[7] More complex questions or information needs make it necessary to provide a sequence of texts (passages), images, web pages and other information. It is important to note, that the information need can't be satisfied by presenting a *set* of data objects. In this case, the human would have to actively convert this set into a sequence in order to be able to perceive it.
To summarize, there are two things that have to be considered by a machine, providing information for a human. One is the presentation or generation of single impression objects, the other is the presentation or generation of a sequence of such objects.
As an example, current Internet search engines may be considered. Search engines provide as a search result a set of links to web pages that is ordered by the estimated relevance for a specific user. Usually a user perceives the hyperlinks on the presented list one after the other, sometimes moves to the respective data objects. Each web page provides sequences of impressions when e.g. a text is read or images are looked at. The user may return to the list of links and continue to perceive this list [51]. By this means a search engine presents (proposes) a sequence of impressions as a result to a query of a user. The list of hyperlinks is generated actively by the system, the respective web objects are usually already present in the Internet.

2.2.2 Search strategies in the Internet

Up to now, information systems don't present perception sequences a user is committed to.[8] In the usual case, as in the case of Internet search engines, the presentation is only a proposal, how a user may perceive the information.

[7] A common text provides a sequence of impressions usually generated by a human author.

[8] An example for a strict information presentation is e.g. a film or an animation.

In practice, he will only choose a subset (subsequence) of the proposed information objects and may also change the order of objects. In most cases it will not be sufficient to use a search engine only, in order to find information. Therefore, a user usually has to apply a 'strategy' in order to satisfy his information need [4]. A first strategy is to apply other media like magazines, etc. in order to get 'positions' (URLs) of information objects. A second strategy is to use a software system like a search engine or intelligent information agents in order to find data objects. Third, a user may request new data objects by activating hyperlinks on a web page, a process that is also called *browsing*. Usually the first two strategies are not sufficient to satisfy information needs, a user will in most cases also apply the third strategy. This strategy can be further subdivided. A thorough insight into different browsing strategies in hypertext is given in [18]. In [27] and [94] three rough kinds of browsing strategies are described. One is search browsing, where the goal is exactly known. This goal can be one single data object or a sequence of such objects, as described above. A second browsing strategy is general purpose browsing, where sources are consulted that have a high likelihood to be interesting. A third strategy is random browsing.
It is obvious, no matter what strategy a user applies in order to find information, he will acquire a sequence of perception objects. This sequence will be influenced by his behavior and by the applied software systems, that may propose (partial) sequences. The term *navigation* will be used in the following in order to describe such a process.

Definition 3 *A navigation in the Internet is a sequence of a user's actions like using search engines or hyperlink activations that results in a sequence of perception objects.*

In this thesis not all possible actions of users in the Internet are considered. The technique that is applied to register user behavior, which will be presented in section 3.4.2, registers only hyperlink activations on HTML pages. The respective URL[9] has to address a resource on the WWW (HTTP protocol) and only requests for HTML pages and some basic data types (like images) are considered.[10]
In order to develop a software system that helps a user to find information, it is the goal to minimize the *effort* for a navigation necessary to satisfy his information need, i.e. it is the aim to minimize the length of the sequence

[9] the target URL of the HTTP request that is initialized when the hyperlink is activated

[10] Requests addressed to CGI scrips or requests for Java applets etc. are not considered. The registration technique may however be extended in order to deal with such requests (section 3.4.2).

and thereby the time requirement and the number and difficulty of a user's necessary actions.

2.2.3 Problem description

With the considerations above in mind it is now possible to define the objective of this thesis.
The general problem that can be derived from the previous sections is the question, how a software system can be developed, that provides a user in the Internet with an impression sequence that satisfies his information need with minimal effort. The ideal case for a user would be a system, that makes a passive navigation possible that, once being started, presents an impression sequence of minimal length, which satisfies the information need without further actions of the user. In practice, the information need of the user has to be estimated since it is not known to the system in advance. This estimation must be based on actions of a user, like entering of search words or an activation of hyperlinks. The first problems therefore are to register information about a user and to estimate the user's information need with minimal effort for the user. Based on this estimation, an optimal impression sequence has to be chosen. A further question is, how a software system can present the estimation results for a user.
Therefore there are four basic steps to be performed by the ideal system.

- The registration of information about a user (e.g. a user's behavior) with minimal effort for the user.
- The optimal estimation of a user's information need.
- The computation of the optimal impression sequence.
- The presentation of the support data.

Usually, the quality of the estimation of the user's information need depends on the amount of information the system knows about the user. Moreover, the amount of information a system registers about a user is usually increasing the effort for the user (the information may e.g. consist of the set of a user's actions). Therefore a main problem, when developing such an ideal system can be emphasized: The task to minimize the effort for a user is usually contrary to the problem to optimally estimate a user's information need. Usually in practical applications it will not be possible to optimally estimate a user's information need due to a lack of information about the user. However, if the estimation of a user's information need is not good enough, an optimal impression sequence can't be computed and the impression sequence

isn't of much worth for the user.
Due to these problems, in this thesis a different support strategy is applied. The searching process isn't interrupted by the system, e.g. in order to present an impression sequence, but it is basically unchanged. The registration of information about a user is done in a way that the user can hardly realize it. The system presents the support information in a separated window and the user may choose to look at this window in order to get the support information.
The kind of information that is presented in the support window doesn't concern the information need of the user directly, but it deals with his orientation in the Internet area and thereby his understanding of the structure of the information. In section 2.2.2 the term *navigation* was introduced to describe a search process combining the use of search engines i.e. query based searching and browsing search. It was described that in contrast to navigation in the physical world, this navigation in an information space denotes the sequence of a user's actions necessary to obtain a sequence of perception objects. Possible actions are e.g. actuating a hyperlink, usage of a search engine or directly entering of an URL-address into the browser's location text-field.
One problem that navigation in the physical world and navigation in an information space have in common is the necessity of orientation. The traveler or user normally has to choose among a number of possibilities which way to go or what data object to load onto his screen. In most cases a decision has however an influence on possible future navigation decisions since e.g. hyperlinks (which are possible future navigation steps) are not the same on different web pages. It is therefore decisive to have an idea of the set and the structure of decision possibilities out of which are resulting future decision possibilities, to achieve an 'optimal' navigation with respect to a certain goal. This goal can be to find a certain data object that contains desired information. As described in section 2.2.1 in more complicated cases the goal can't be achieved by presenting a single data object. It is then the objective to acquire a sequence of data objects due to the nature of human perception. Orientation in the information space is even more important in this second case, since the required navigation process to achieve the complex goal easily causes disorientation.

2.3 Similar projects

In section 2.2.3 the importance of orientation during information search in the Internet was discussed, which is the main topic of this thesis. Different pos-

sibilities are known from literature that make orientation in an information space for a user easier. There are solutions or systems that are non-adaptive and present the same support information for all users. Other systems are adaptive with respect to the specific situation of a user, e.g. his position or his history in hyper-space. More advanced systems apply statistical methods ('intelligence') to estimate the interests of a user.

Basic strategies and systems to make orientation in the Internet easier

A first basic possibility to make orientation in the Internet easier is to design web pages, that have a clear structure. Similar problems are discussed in the field of web page design and will not be examined in this thesis.
A possibility to make the structure of the hyper-space easier to understand is to help a user to understand the 'consequence' of a hyperlink activation. A first basic possibility is to create meaningful anchor texts of hyperlinks or hyperlink images, that clearly represent the contents of the data object where a hyperlink points to. A more advanced method is to provide a user with additional information about the data object, pointed at by the hyperlink. One project to achieve this was presented in [107], where an enhanced link user interface provides a user with meta-data about a hyperlink. A similar project called 'HyperScout' was presented in [113]. Here additional information about a hyperlink is presented to the user, like the size and the type of the document. In [59] a system is described that provides a user with a visual (thumbnail) preview of the target of a hyperlink. In [3] the relevance of a hyperlink is estimated for a specific user. A system is then presented to mark hyperlinks on web pages if they are estimated to be relevant for this user.
Another possibility to make orientation for a user easier is a well-designed structure of the web site, i.e. a web site design that can easily be understood by a user. It was e.g. in [93] shown, how the architecture of web information systems can be improved using design patterns.
A further possibility to make orientation easier is to give a user a survey of the data objects on a web site. This presentation has strong similarities to a map of a real landscape, although the 'space' of objects is different. Here the presentation of the set of data objects on a web site or a map of the hyperlink structure is possible. These kinds of presentations are usually denoted as 'site maps'. In [55] a survey is given, how site maps can be and are actually presented on modern web sites. The layout of such maps may become very difficult for extensive web sites especially when connections like hyperlinks are taken into account. Such layout questions are analyzed in the field of in-

formation visualization. In [49] a survey of common graph and information visualization techniques is presented. Such site map presentations can't go into detail because the maps would otherwise become very complex in many cases. They are only a means to give a visitor of a web site a rough idea of the (logical) site structure. The presentation technique is very simple since the map can be loaded by the client like a normal web page and is presented in the current browser window.
Most of the previous possibilities to achieve orientation of a user are not 'personalized', i.e. the presentations are not developed with respect to the individual situation or interests of a certain user. The support is identical for all visitors of a web site. However different users usually have different backgrounds, histories in the data structure and goals or questions in mind, which leads to the concept of *adaptive hyper-space*. The support systems may e.g. be adaptive with respect to

- the position of a user in hyper-space
- previous positions (i.e. his history)
- query texts (, questionnaires etc.)
- his preferences[11]

One example is the presentation of data objects that contain hyperlinks pointing to the current data object in the browser window. A second example is the presentation of possible navigation decisions using hyperlinks up to a certain depth. For such presentations the system must register a user's current position in the web site. It must also be able to acquire knowledge about a web site. The Hyper-G system presented in [28], which consists of an advanced browser software running on the client side (HyperWave) and a specially developed Internet server makes such presentations possible. Other systems for this kind of presentation have been presented in [6] and [114].
In [114] not only the local hyperlink structure can be presented but also the former navigation path of a user. This navigation history presentation is an additional attribute of the system to avoid loss of orientation for a certain user. It is also possible to show former usages of other users and thereby to present a first hint to global relevance of data objects. A system to acquire this information was presented in [53], too, but this system doesn't provide a visualization possibility on the client side. The contribution of this thesis (chapter 3) with regard to similar adaptive visualization techniques was presented in [62]. The systems are compared to each other in chapter 3.

[11] as estimated by the system with regard to the available information about a user

Adaptive information agents

Some of the systems described so far are adaptive with respect to the position of a user and his previous searching behavior (his navigation history). These systems are not adaptive with respect to a user's preferences or individual interests. As described in section 2.2.3 these individual interests have to be estimated by the support system with regard to actions of a user. More advanced systems that are capable to perform such an estimation are called *intelligent* and were usually presented in the field of *intelligent software agents* [9], in this special case in the field of *(adaptive Internet) information agents* [58].
Some overviews of existing systems are given in [58], [9] and [70]. Similar research problems are examined in the field of *intelligent hypertext* [78], [14], [30].[12]
Adaptive Information agents can be subdivided into single agent systems or multi-agent systems, that may learn in a collective way. Multi-agent learning will not be considered here. The different adaptive single agent systems can be characterized by the applied learning strategy. *Content-based learning* is based on a single user's past behavior [120]. Obviously this method requires relatively large amounts of data from a user in order to make the formulation of a statistical model possible. *Collaborative learning* [120] is based on the observed behavior of a group of users [89], [3], [114]. The approach can be applied whenever a user behaves in a similar way as a group (or a subgroup) of other users. Possible learning methods are e.g. learning by example, analogy or discovery. The methods also differ with regard to the kind of feedback for the agent, e.g. by means of reinforcement, supervised or unsupervised learning, the type of interaction among agents and human users, the purpose of learning, i.e. the kind of support information, and the applied presentation technique [58]. Different techniques are applied to acquire information about a user, like an observation of his online activities, preferences and questionnaires.
Most single adaptive information agents that have been developed so far are personal web assistants or adaptive interface agents that search for relevant information by anticipation (estimation) of information needs and preferences of users.
One example for such a system is the web browser 'Intelligence', developed by IBM [44]. The browser provides e.g. a ranking of web pages previously visited by a user. The ranking considers the number of times a page was visited and the time needed to load a web page.
Similar to this, Letizia [68] is an agent that assists web browsing. Letizia

[12] An intoduction into technical aspects is given in [7].

is an extension to common browser software like Netscape (section 2.1.3), that observes the actions of a user, like entering URLs or exploring Web pages. Based on a user's preferences, the system loads and evaluates web pages automatically, that are pointed at by hyperlinks on web pages that were already requested. Web pages that are considered to be relevant are recommended to the user.
The system Web Watcher also recommends web pages to a user [3]. Learning is accomplished by means of two different strategies. A first strategy is based on a similarity between interests (provided keywords) of users. At the beginning, a user has to describe his interest by providing keywords. During his navigation process, every applied hyperlink is augmented by adding the keywords to the hyperlink. An evaluation of hyperlinks for a user is done by comparing a user's interest (keywords) and the descriptions of hyperlinks on the current page in the browser window. The hyperlinks estimated to be relevant are marked by a special symbol. The comparison applies a TFIDF distance measure [95]. A second learning strategy based on the previous one applies a reinforcement learning technique and takes into account the web site structure. Paths are found through the web which maximize a reward function taking into account possible future navigation decisions of a user.
There are many projects similar to the ones described, as e.g. WebMate [23], Let's browse [71], Syskill Webert [80] and Info Agent [29]. The adaptive support system presented in this thesis was published in [63].

2.4 Fundamental technical problems

This section offers a survey of basic technical problems that have to be solved in order to put the described software system into practice, some of which were examined and solved in other problem fields. Some solutions can directly be used as modules by the presented software system in this thesis.

2.4.1 Static analysis of Internet information

One important question is how a software system can acquire knowledge about data objects in the Internet, which will be referred to as *static analysis of Internet information*. The term 'static' will be used although the amount of information in the Internet is currently subjected to a fast growing process. Due to this fast increase of the amount of information, due to the different (script-) languages that are used to present and to make information available (section 2.1.4) and due to frequent changes, knowledge acquisition in the Internet is in the general case a very complex topic. This knowledge

acquisition is e.g. an important topic for the development of search engines or information agents.

Finding data objects

One first issue is how to find information objects in the Internet. Well known software systems to find data objects are Internet *spiders* (also called robots or crawlers) [36], [22]. These software systems assume that most data objects in the Internet can be reached by a path in the hyperlink structure.
The rough idea of these systems is first to load down certain starting points (web pages). The web objects are filtered (parsed) for URL addresses being contained as hyperlinks in the pages. The procedure continues recursively at the extracted URL addresses. Different strategies were presented to pursue this kind of search for different purposes [22] (e.g. breadth-first search [11]). In [11] problems were discovered resulting from this link-following strategy. In the article a considerable part of the Internet was analyzed with respect to hyperlink connections of data objects. The whole analyzed structure could roughly be divided into 3 groups. A first group contains about one third of the analyzed data objects. It characterizes the 'strong connected component' (SCC), where every data object (e.g. web page) is connected to any other data object by a sequence of hyperlinks. A second group, also containing about one third of the data objects, is denoted as 'OUT'. It represents data objects that can be reached from the SCC component, but there is no link traversal to the SCC component from OUT possible. A third component, denoted as 'IN' has hyperlink routes to the SCC component, but no link traversal to the IN component from SCC or OUT is possible (possibly IN contains many new sites that people have not yet discovered and linked to). It is obvious that although two thirds of the Internet objects, the OUT and the SCC component, can be reached by link traversals, one third can't be accessed in this way. These results reveal severe limitations of spider or crawler systems to analyze Internet contents. Finding data objects in the Web is thus a non-trivial problem.

Hyperlink structure retrieval

The web pages and other data objects and hyperlinks can be regarded as a graph structure (a formal definition is given in section 4.3), where the data objects are the nodes and the hyperlinks the (directed) edges of the graph.
This graph structure doesn't have to be static. It may e.g. be the case that a Java applet in the user's browser window shows different hyperlinks at different times. A second problem concerning the graph structure is that some

web pages are generated dynamically (e.g. search engine results). In the following these advanced aspects are not considered. Only a static hyperlink graph will be considered and generated pages are modeled by representatives that contain all hyperlinks that may occur.
The information provided by the spider software above can be used to register, what hyperlinks exist in the Internet and may thus be applied to register the (mostly a partial) graph structure of the Internet [11]. This process, 'mining of the Web's link structure' [20], was applied e.g. by [57] and [21] in order to find characteristic properties of Internet topologies and (topological) properties of objects in the Internet. In this thesis this knowledge about the structure is applied for a visualization of a local hyperlink structure (section 2.4.3).

Information retrieval and multimedia retrieval in the Internet

A second question concerning the static analysis is to examine the contents of data objects. Since the data objects are in most cases multimedia objects, a contents analysis may consider texts, images etc. and even the arrangement of the media objects in web pages. As far as texts are concerned, a contents analysis was examined in the field of information retrieval (IR), e.g. in [96] and [47]. The contents analysis of images, falling into the research area of computer vision, was e.g. described in [67].

2.4.2 Registration of the use of the Internet

In order to develop a software system that presents information depending on the interests of a user, the system has to acquire information about the user. For collaborative filtering purposes, information about a group of clients has to be acquired. In the case of Internet usage, this is a non-trivial problem since the participants (client and server) may be very distant from each other. It is the goal to maximize the information about a client with respect to a minimization of the effort for a user.[13] This problem will be extensively discussed in the next chapter. At this point, a rough survey of principal methods to acquire information about users is given.
A first possibility is obviously to ask users directly what they are looking for and why they make certain decisions. In [51] every step of the interaction during an Internet search is made verbally explicit, a technique that is used e.g. in [32]. The same technique is employed in [18]. It is obvious that this

[13]The knowledge can e.g. be used for a service provided by a company to customers. Clients obviously can't be expected to actively provide information because it may increase the risk that they change the company.

strategy is very extensive and may be applied for scientific purposes but may hardly be applied in real applications. In the case of Internet interaction, it is also technically difficult to implement because of the physical distance between client and server. It is therefore crucial to develop methods to automatically register user behavior on the server side.
A method that is made possible by HTML and other languages (section 2.1.4) is directly to request information from a user. A user may e.g. fill in questionnaires, provide query texts etc. Search engines depend on such information in order to search for (i.e. to estimate) relevant documents. Some web pages ask users for feedback information. The registered information is simpler to process than e.g. audio-taped information but it requires additional effort for a user.
One possibility to automatically register information about a user, without

```
62.226.76.63 - - [01/Aug/2001:00:03:11 +0200] "GET / jpa/images/ars.mpg
HTTP/1.0" 200 1883839
138.15.10.5 - - [01/Aug/2001:00:03:46 +0200] "GET / tbl/links.html HTTP/1.0" 404
294
216.239.193.84 - - [01/Aug/2001:00:18:17 +0200] "GET
/ chp/Pictures/AFPAC/Day1/pm/picture-017.html HTTP/1.0" 304 -
212.135.130.131 - - [01/Aug/2001:00:20:00 +0200] "GET /
216.239.193.76 - - [01/Aug/2001:00:25:39 +0200] "GET
/ chp/Pictures/AFPAC/Day2/am/picture-017.html HTTP/1.0" 304 -
216.239.193.81 - - [01/Aug/2001:00:42:06 +0200] "GET
/ chp/Pictures/AFPAC/Day1/dinner/picture-007.html HTTP/1.0" 304 -
```

Figure 2.1: Example of a partial access-log file. The first column shows the IP address of a client, the following three columns store the time when the request reached the server, a *GET* request, containing e.g. the URL of a requested data object, and information about the applied HTTP-version.

increasing his effort, is the analysis of the so-called *access-log-files*. In the most common case a server receives requests from a client as HTTP requests (section 2.1.2). These requests are usually stored by common server software like Apache (section 2.1.3) in log files, the access-log files. A sample file, consisting of 6 requests is shown in figure 2.1. As can be seen, the information available from the original HTTP requests is stored in the form of entries, containing information about the IP address of a client, the time when the request reached the server and usually a *GET* request, containing the URL of a requested data object (section 2.1.2), and information about the applied

HTTP-version.[14] It is possible to identify a client by his IP address and thus to obtain the set of his requested data objects. This procedure and the exploitation of such information, called *mining of access-log files*, is widely examined and applied in literature [117], [82].
Another method to automatically register a user's actions will subsequently be refered to as the *ping* method. This method is based on the experience that caching procedures in the Internet (section 2.1.3) have usually common properties with respect to which data objects are cached. For example a HTTP-get request that looks like a database query because it contains an additional query string (separated by the '?' symbol in the GET string) is likely not to be cached [72]. The idea of the ping method is to provide each web page with a small data component (the components of web pages like images etc. are loaded automatically) that is likely not to be cached and that points (by its URL) to a special registration server. By this means it can be registered, which web pages are requested. The method is not applicable to other data objects, like images etc.
A further possibility is to force the Internet traffic to a certain server or a number of servers to go through a proxy server (2.1.3). In this case, all requests can be registered by the proxy server.
There exist problems resulting from these strategies that will thoroughly be discussed in the next chapter in section 3.4.3 and a more advanced method will be presented.

2.4.3 Information visualization in the Internet

As described in sections 2.2.1 and 2.2.3, an optimal search support tool should present an impression sequence to a user depending on his information need. This strategy would require a precise estimation of the information need and a reliable quality of the impression sequence. For principal reasons a software system will have poor information about a user at the beginning of a search and possibly at later stages of a search, too, which makes it hardly possible to provide an optimal impression sequence.
The support strategy that is chosen here (section 2.2.3), is therefore different and makes a simultaneous presentation of different estimation results possible. The strategy is to present adaptive maps of the data structure.[15]
The presentation of Internet (hypertext) structures is a problem that is well-known in Information Visualization [55], [43]. It was mentioned in section

[14] The type of the data object (MIME content-type, section 2.1.2) can be determined from the extension of the requested filename.

[15] Only static (hyperlink) maps are considered (section 2.4.1). Dynamically generated web pages are not considered.

2.4.1 that data objects and hyperlinks constitute a graph structure. A presentation of this multimedia structure taking the topology into account is therefore a problem known from the field of graph visualization [49].
In this thesis the mere (hyperlink) graph structure is considered. Another approach taking semantic aspects into account is provided by *topic maps* [115].

2.5 Summary: Internet Technology, Basics

In this chapter, basic technical concepts of Internet communication have been introduced like network protocols and languages used for communication; the most important software systems like browsers and Internet servers were described. This chapter introduced the fundamental problems of information search in the Internet by comparing it to a common stand-alone database. Considering the nature of human perception, an ideal support tool for information search was described, providing sequences of impression objects at minimal costs for a user, that satisfy his information need. The objective of this thesis to help a user to orientate himself in hyper-space was presented, and compared to the general task of an ideal information search support tool. This chapter described similar projects that were presented in the field of adaptive information agents. Some solutions concerning technical problems of the presented system were described, which are known from other research fields. At the beginning of the next chapter, the objective of the system will be described in more detail. Then the system will be technically described and results concerning the development of the support agent, the visualization of navigation history and local hyperlink maps, will be presented.

Chapter 3

Internet Agent for Adaptive Hyper Space Visualization

The aim of this chapter is to describe the developed adaptive information agent. Main ideas of the software system are presented and compared to similar systems which were described in section 2.3. The components and technical attributes of the agent are presented. Finally, some visualization examples are shown.

3.1 Objective of the software system

The objective is to present an Internet agent that makes a navigation support possible for users, who visit a distributed (but restricted) Internet environment'. A 'distributed Internet environment' as defined in section 2.1.1 denotes a number of Internet servers, e.g. the Internet servers of a university or of an enterprise. The special case considered here is navigation in the WWW. The agent will make it possible to acquire knowledge about the searching behavior of a certain user at the server side[1] and also to persistently collect knowledge about past navigations of users in a database. The navigation support will be achieved by presenting maps showing the local hyperlink structure or other decision possibilities in the Web environment or by presenting the navigation history of a user. The agent makes it possible to present these maps on the client side. One important aspect is how to technically perform the presentation of these maps, taking into account the most likely software constellation on the client side. Another important aspect is the possibility of an autonomous presentation. Once activated, the

[1] This registration should be as independent from the applied browser software as possible.

presentation can be pursued without further actions of the user (i.e. actions concerning the navigation support presentation). Autonomy is an important attribute of an agent [9].
The technical descriptions in this chapter show, which information about a user's searching behavior can be measured by the agent on the server side. In the next chapter this data type will be used to derive a user model. The data type is then used to develop an estimation procedure for the relevance of future navigation steps or sequences of future navigation steps.

3.2 Possible architectures for navigation support tools

Comparing the architectures of the different navigation support strategies or tools described in section 2.3, it can be outlined that there exist four different fundamental architectures as shown in figure 3.1. These basic architectures will be used to demonstrate main advantages or disadvantages of the respective support strategies.
The navigation support strategies that present the same information for all users, like well designed web pages, well designed site structures or site maps can be presented using common browser and Internet server software and the usual communication, as shown at the top of figure 3.1. 'C' in the figure denotes an Internet browser process running on the client side and 'S' denotes a process running on the server side or a (proxy-)server in the Internet connection between client and server. 'S' may also represent a number of server processes. The arrow between client and server denotes a connection through the Internet.
If adaptive hyper-media has to be realized, where the navigation support system presents different information for different users, more advanced architectures have to be applied. There are (at least) three different basic possible software architectures to be considered to realize such an adaptive navigation support denoted in figure 3.1 as architectures no. 2,3 and 4.[2]

One possibility, architecture 2 in figure 3.1, is to install a software on the client side, denoted as 'AC' in the figure (AC = **a**gent component on the **c**lient's side). This software can be a specially developed browser software or a plug-in for an existing browser software (section 2.1.3). It can also be a software communicating with the existing browser software. Such an architecture makes it possible to register the searching behavior of an individual

[2]More advanced architectures are conceivable as described in the discussion in section 3.7.

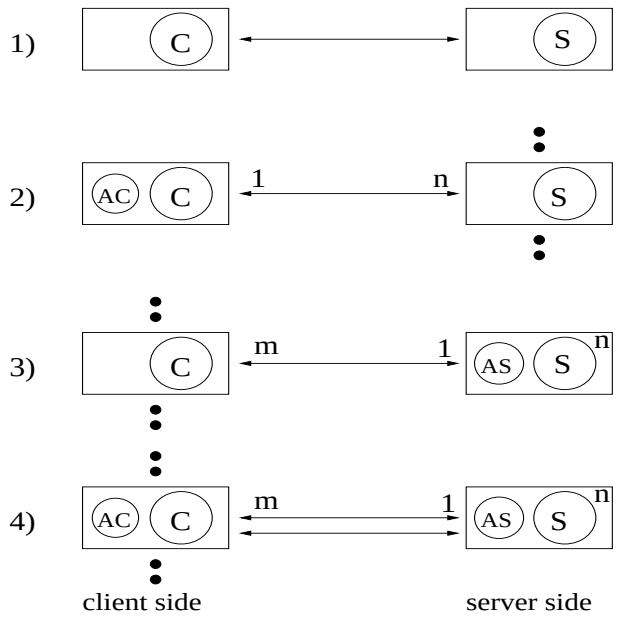


Figure 3.1: Different possible architectures for a navigation support tool, 'C' denotes the browser process on the client side, 'AC' denotes an agent process on the client side, 'S' denotes an Internet server process and 'AS' is an agent process on the server side. In the 2nd architecture there is one AC component and a number of (n) server processes. In the 3rd architecture there is a number of (m) client processes and one AS component. There may exist several (n) server process (S).

user, depending on the 'rights' of the respective software, which is important e.g. for content based filtering [119], [120]. However the accumulated knowledge about the user behavior is located in this case on the client side. As described above it is our aim to collect usage profiles of different users on the server side. Using this architecture it would be necessary to transmit the usage information to a central server. But this task is difficult to put into practice and it is in most cases not wanted by the user because of data protection reasons. A second disadvantage is the necessity for the user to install a software on his computer which makes it inconvenient for visitors to use the navigation support tool.[3] An installation is especially inconvenient for those visitors who visit the web site only once or from time to time. Such an installation is also an additional security risk for a client. As expressed by the black dots in figure 3.1 one navigation support software is active on the

[3]This inconvenience is an important aspect, if the site is offered e.g. by a company.

client's computer, but the support covers navigation on many (n) web sites, in most similar systems it covers the whole WWW [68]. A serious drawback of this architecture in this context is the fact that the support system can hardly acquire knowledge about data objects on web sites. Since the support is in many cases not restricted to a specific web area (as in [68]), a thorough analysis of Web data like indexing or a structure analysis is not possible. In some cases this analysis is e.g. restricted to the data objects that are pointed at by the current web page in the client's browser window [68]. An important advantage of the architecture 2 is the (potential) ability of the system to present and visualize support information. Since the AC component is a persistently stored program it can add presentation abilities to the original browser's presentation techniques in principal without any restrictions.
Another possible architecture, architecture 3 in figure 3.1, is to install the agent component (AS) on the server side. The system registers a user's behavior on the server side as far as this is possible and uses communication with the client only for presentation or visualization purposes. The black dots in the figure denote that there may be a number of (n) clients (simultaneously) but there is one agent server component.[4] The main advantage compared to the previous architecture is the possibility to collect information about searching behavior of different users at the server side and thereby to apply collaborative filtering or learning techniques. A problem with this second architecture is that not all the information about a user's behavior is known on the server side as described in section 3.4.2 in detail. As described in section 3.1 we consider an Web area consisting of a small number of Internet servers. It is e.g. possible in this case to evaluate the access-log files storing requests of clients on these servers (section 2.4.2). Requests from clients being addressed to other servers can of course not be registered in these access-log files. Thus only the partial navigation behavior of a client in the considered Web area can be registered. However even the (HTTP) requests to data objects on the considered servers[5] can not always be traced due to caching strategies in the Internet in the connection between client and server.[6] These problems will be dealt with in detail in section 3.4.2. A second disadvantage using this architecture is that the possibilities to present a navigation support are limited, since only the common presentation possibilities of the browser software can be applied.
A further architecture, architecture 4 in figure 3.1, combines the two previous architectures as shown at the bottom of figure 3.1. Here an additional soft-

[4]However there may be several (n) Internet server processes covered by the agent or proxy process.

[5]Only requests for basic MIME-types are considered (section 2.2.2).

[6]Browser specific techniques like 'DDE' are not considered here.

ware is installed on the client side (AC) communicating with an additional server or agent component on the server side (AS). The visualization possibilities may be as extensive as discussed for the second architecture. This architecture also profits from the second architecture with respect to the knowledge about the client's behavior. Learning from different users on the server side is possible because of the server side component. A disadvantage of this system may be its complexity, the extensive network communication, and the client side installation as discussed for the second architecture.
The architectures of the navigation support systems presented in section 2.3 can be assigned to one of the different architectures respectively. The intelligent information agent 'Letizia' presented in [68] and the web browser 'Intelligence' presented in [44] apply the second architecture. The agent Letizia is a specifically developed browser software that has to be installed on the client side. The first agent is an extension of a common browser software (section 2.1.3).
The Web Watcher system presented in [3] applies the third architecture. The navigation support is sent to a client as a modified web page being presented by a common browser software. All registration and learning steps are performed by a special developed server system.
The fourth architecture in figure 3.1 is applied by the software systems presented in [114] and [28]. Communication between a special developed browser (HyperWave) and the Hyper-G server makes the extensive navigation support possible in [28]. In [114] the software on the client side is an extension to common browser software communicating with the server system.
The described architectures have simple structures that are currently applied by many navigation support systems. It is conceivable that future architectures are more complex using different agent processes and an extended communication between the agent components as discussed in section 3.7.

3.3 Agent description: survey

According to section 3.1 there are three main tasks to be performed by the software system. The first task is to register the navigation behavior of users on the server side. Here it is the objective to accumulate as much information about a user as possible. However it is often the case that this accumulation of information creates 'costs', i.e. drawbacks of the system like the number of necessary TCP/IP connections, the complexity of the program or the software requirements on the client and on the server side. It is one decisive objective to be able to offer the navigation support to as many clients as possible and to make it as fast as possible. Therefore the network communi-

cation and the software requirements on the client side should be minimal.
The second task is the generation and the preparation of the information to be presented to the user. In this chapter mainly technical aspects concerning the development of maps of the local hyperlink structure will be described and methods to extract this information from the considered Web area (section 3.4.1). More refined presentation methods will be discussed in subsequent chapters.
The third main task is the presentation technique on the client side. This technique has to take into account the most likely software constellation on the client side in order to make it possible for all or at least most of the users in the Internet to use the system and to make it as easy as possible to use it. Another objective concerning the presentation is an automatic update when new information is available without the client having to initiate this process.
There are different software constellations possible on the client side, i.e. different operating systems, Internet browsers (2.1.3), different browser settings etc. It is assumed here that the user uses a browser software like Netscape or Internet Explorer, that the language support for Java is activated (section 2.1.4) and that there is no firewall in the connection between client and server (section 2.1.3). The different requirements will be examined in detail later on.

3.3.1 Principal software structure

In this section the software architecture is described that was used to implement an Internet agent with respect to the objective in section 3.1. It applies an architecture which is similar to the architecture 4 in figure 3.1 which will be explained in the following.

The figure 3.2 shows the structure of the software system. The circle on the right shows a number of Internet servers representing the Web area being considered. These servers have to be defined in advance and made known to the software system. Possible sets of servers are e.g. the servers of a university or of an enterprise. The provided navigation support is only active when the client requests data objects from these servers.[7] The rectangles at the bottom of the image show a number of client computers that may use the system simultaneously. The arrows denote the communication between client and server computers via the Internet, usually through Socket connections. In the common case, client and server would directly communicate through the Internet. In this architecture however, an agent server is situated in the

[7]The support system is also active on 'border' objects as defined in section 3.4.3.

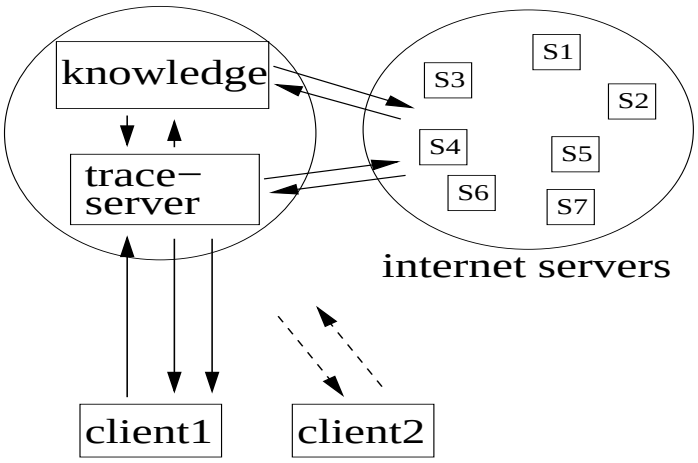


Figure 3.2: Structure of the navigation support tool; the trace server is located near the considered Internet servers.

connection between a client and the Internet servers. In section 3.4.2 it will be shown how this redirection of the Internet communication can be put into practice. The agent is represented by the ellipse on the left. It consists of three main components. One component is a database denoted as 'knowledge' in the figure. It contains knowledge about the considered Web area, like the hyperlink structure and possibly information about the contents of data objects, like index lists. It may also contain knowledge about former usage profiles of clients. The second component of the agent is a specifically developed server being described in 3.5.1 in detail, refered to as 'trace server' in figure 3.2. This server makes the redirection of the Internet communication between client and server through the agent software possible. For this purpose it has to communicate with the Internet servers and the client computers simultaneously. For practical operation it must be able to serve a number of clients simultaneously. A third component of the agent system that isn't shown in the figure, is a program being active on the client's computer. As described in detail in section 3.5.2, it is based on functionalities of modern browser software presented in section 2.1.4. The software is downloaded by the client from the server as a Java applet and automatically installed, which can be regarded as a temporary software installation.

3.3.2 Comparison to other architectures

Similar to the considerations in section 3.2 it is now possible to compare the described system from an abstract point of view to former systems and thereby to give arguments why the architecture was chosen.

With respect to the different architectures in section 3.2 it can be seen that

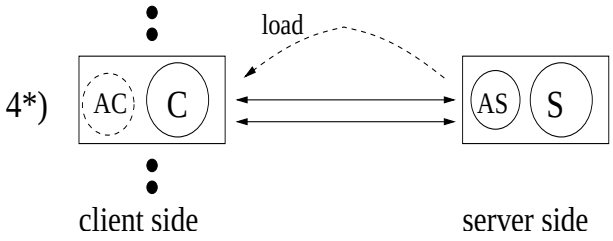


Figure 3.3: Architecture for the navigation support system. The client side agent component is loaded from the server component and installed temporarily for each navigation session.

the described agent system applies the fourth architecture, having an agent component on the client and on the server side respectively. However it is a decisive difference compared to the Hyper-G system [28] and the system described in [114] that the software component on the client side is not a persistent program that has to be installed manually by the user but an automatically installed Java applet loaded from the server for each navigation session as shown in figure 3.3 by the dotted arrow. Another difference is, that the processes S and AS are usually located on different computers and that there may exist more than one server process S. The architecture will therefore be denoted as $4^{\star}$ in the following.

When comparing the software architectures, a decisive aspect of the value of the navigation support is its usability, i.e. the number or percentage of clients in the Internet who may (and actually want to) use the system which is obviously depending on the benefit an individual user has if he uses the system and his necessary investment to apply the system[8]. In the case of the systems applying architecture 4, [28] and [114], in principle every user can use the navigation support if he has installed the client component or browser software, provided that the software system supports his operating system and works together with (or is a plug-in of) the browser software. In practice it is obvious that the support system is very unlikely to be used by users visiting the considered web site only once or from time to time, since

[8]Possible 'investments' are e.g. time, effort, security risk etc.

the investment to install the system would then be too high, compared to the benefit. Though this architecture can potentially reach nearly the whole Internet community it would therefore likely to be used only by a small percentage of users.
Our system, though it has nearly the same architecture, can be used by users without any effort. Since the Java language is currently supported by nearly all browser types[9] a large fraction of Internet users may use the system. This is a main advantage of this architecture $4^\star$ compared to the architectures 2 and 4. A disadvantage that results from this architecture is however the presentation abilities for support information. Though the presentation techniques of the browser can be extended, which is an advantage compared to the 3. architecture, they are potentially less extensive than for the 2. and the 4. architecture. The Java applet has to be small since it is loaded over the network which would take too much time if more advanced presentation techniques are offered. This loading time or increased network traffic would increase the necessary 'investment' for a user. An advantage that this architecture inherits from architecture 2 and 4 is the knowledge that can be acquired concerning a user's behavior. This registration is also restricted by the size and the rights of the applet on the client's computer (section 2.1.4). This important aspect will be discussed in detail in sections 3.4.2 and 3.4.3. The architecture inherits the possibility to register navigation information on the server site from architectures 3 and 4 as described in section 3.2, which is an important attribute to make collaborative filtering and learning possible. In contrast to architecture 2, architecture $4^\star$ also inherits from architectures 3 and 4 the possibility to acquire knowledge about a specific web area as described in section 3.2. This can also be regarded as a disadvantage since the architectures are thereby restricted to a specific web area and provide a navigation support only for a very small, well-defined fraction of the Internet. It can be summarized, that choosing this specific architecture $4^\star$, the navigation support can easily be applied by a vast majority of Internet users. But this advantage limits presentation and registration techniques due to the restricted rights and the size of the AC component. The architecture makes server-side registration of clients' behavior and thus collaborative filtering and learning possible. The architecture may provide a navigation support only for a well defined Web area. However a thorough examination of this Web area for supporting purposes is thus possible.

[9]Browser not supporting Java Script and Java like Opera are currently used only by a small fraction of Internet users.

3.4 Basic concepts

In this section it is the aim to describe and to discuss the main ideas that were applied to realize the agent system. The aspects that will be dealt with are the static web site analysis by the agent, the registration technique of the client's navigation behavior at the server side and the developed presentation technique at the client side. The technical implementation will be presented in section 3.5.

3.4.1 Static analysis of hyperlink structures

In section 2.4.1 it was discussed how a software system can acquire knowledge about data in the Internet (WWW). The problem of finding data objects in the Web was described in the case that common spider techniques are used, because data objects may not be reached by a sequence of link traversals. This problem reveals severe limitations of spider or crawler systems (section 2.4.1) to analyze Web contents, which are applied in search engines and other Web analysis tools.
In the special case described here (section 3.1) the situation is far more simple than in the case of general Web indexing machines. The Web area that has to be analyzed by the system contains only a relatively small number of well-known servers, as e.g. the servers of a university or of a company. It would be possible in the described case not to apply spider techniques, but directly to scan the respective directories on the servers that contain data objects presented in the web.
However there are severe drawbacks of this (local) technique. The method requires (reading-)rights on the considered servers and vast effort to supervise and update the system, since the indexing system would have to be informed about locations on the considered web servers where web data are stored.
In contrast to this, the spider technique can be activated and updated automatically. The only variables to consider are the starting URLs for the crawling procedure (which have to be part of the IN or the SCC component as denoted in section 2.4.1).
A second important issue is how to process multimedia information i.e. how to build indexes, which is for textual data a research problem known from the field of information retrieval [96], [95]. The problem in the case of Web data is much more complex since not only textual data are available but also images and other multimedia data objects.
The presented system considers the hyperlink structure as described in section 2.4.1, which will be sufficient to demonstrate main aspects of the agent in the following sections. The actual contents of data objects will not be

considered.
The crawling software that was used as a basis for this structure analysis is presented in [36]. Originally this software was developed to maintain a web site, especially to find inconsistent links. However the presented system uses the output of this software to convert it into a graph structure with the nodes representing the data objects and the edges representing hyperlinks on web pages that make a transitions between pages possible. Only static hyperlink structures are considered (section 2.4.1).
It has to be mentioned that this graph may not represent the whole set of possible navigation decisions of users navigating on the considered web sites (section 2.2.2). With the help of search engines it is possible to get from one web page to possibly any other web page. Those transitions may not be part of the set of hyperlinks.
It can be resumed that the main difference of this system to common web indexing procedures is that emphasis is laid on the hyperlink structure analysis in a relatively small web area. This structure knowledge will be used to demonstrate main aspects of the developed agent system in subsequent sections. Some disadvantages are inherited from common spider techniques as described above, since a spider is used to find data objects by hyperlinks traversals. Some data objects and hyperlinks can't be retrieved if they are located in the 'IN' component. These objects can however neither be found by search engines in many cases, because they apply similar spider techniques to find web objects.[10] Therefore these data objects are 'invisible' for a client in any case. However, since the Web area is relatively small and well known (to the web administrator), it is possible to choose 'good' starting points in the 'In' component and to manually add other 'In' objects.

3.4.2 Registration of a navigation process in a distributed Web environment

In section 2.4.2 different possibilities were discussed to register the use of the Web and feedback information, like requested data objects, registration of navigation decisions, search words or questionnaires. The described software system considers the registration of the sequence of navigation decisions of a specific client, which is called 'navigation' in section 2.2.2. In section 2.4.2 different methods were presented to register this kind of user behavior at

[10]This is not true if there exists a hyperlink from an external data object to those objects, since search engines search for web objects mostly in a very large web area and are thus likely to find the respective hyperlink. It would be more precise to differ between local and global 'IN' component.

the server side. In this section problems resulting from these strategies are described and a new method to register user behavior at the server side is introduced. This method is applied by the presented software system.
The objective is to register user behavior at the server side. It was discussed in section 3.2 that the comprehensive knowledge about the behavior of a user can usually only be registered on the client side. In order to make e.g. collaborative filtering possible, this knowledge about the user behavior has to be collected somewhere; a possible place may be one of the servers.[11] When applying a client/server architecture with a specifically developed browsing software, registering user profiles on the client side, these data can be sent to a registration server through the Internet. This is possible for the architecture 4 in section 3.2 used by [28] and this technique was actually applied in [114]. However, as discussed in section 3.3.2 it is the objective not to apply a software or an Internet browser extension (plug-in) on the client side.
Methods to register user behavior directly on the server side are the access-log method, i.e. the registration of HTTP requests by the Internet servers themselves, the ping method, i.e. to provide each web page with a small pointer object, pointing to a registration server, and the proxy method, i.e. a proxy server in the link between client and server for registration purposes (section 2.4.2).
In contrast to the client side registration, these methods suffer from a common problem. Due to caching strategies in the connection between client and server, not all requests of a client actually reach the server. Caching in this context means the temporal storage of data objects in the link between client and server. Caching strategies are a means to reduce net traffic and to shorten the response time. They are applied by proxy servers in the link between client and server (section 2.1.3) or by a browser itself.
Another problem is that different clients might use the same Internet provider and therefore may have the same IP-address as registered by the server. If two clients use the navigation tool at the same time, the server may thus only register the profile as coming from one person. This would cause a wrong navigation profile and may lead to wrong conclusions.
Some of the problems caused by caching strategies can however partially be solved by estimating the real profiles, called 'action inferring' in [53]. Because of caching techniques in the Internet, the profiles that are registered are possibly only a subsequence of the actual sequence of requests of a client. (The original sequence will be refered to as the *real sequence* in the following.) Since the hyperlink structure of a web site can easily be determined, as described in the previous section, it is possible to estimate the real navigation

[11] Extensions of this concept are thinkable, as described in the discussion in section 3.7

path with regard to the hyperlink structure. This strategy was applied e.g. in [2], [53]. An example is shown in figure 3.4. The figure shows 4 web pages

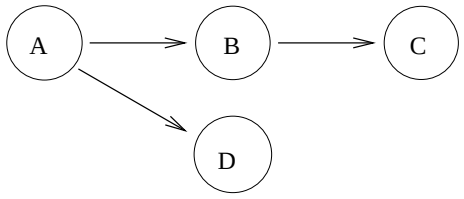


Figure 3.4: Hyperlink structure with 4 web pages A,B,C and D and 3 hyperlinks, AB,BC and AD

A,...D and hyperlinks between the pages constituting the hyperlink structure as determined e.g. by the method described in the previous section. Provided that the server registers the sequence of requested data objects ABCD, it is obvious that the likely actual client's navigation path was ABCBAD, as can be seen from the hyperlink structure.[12] The elements B and C were stored in this example by an intermediate proxy-server or by the browser software itself.
However the real navigation path could also have contained additional revisits. The sequence could e.g. also have been ABCBCBAD. The action inferring techniques may therefore only be an approximation to the real revisiting information The ping method may provide the revisiting information, since the pointer object is likely not to be cached. However the reliability of this technique also strongly depends on the applied caching strategies in the Internet that are not known to the software system in advance.
Another problem caused by caching in the link between client and server can be seen in figure 3.5. Provided that the client moves from A to B and then to C, the server would store the access sequence ABC. With regard to the hyperlink structure, the original sequence could have been ABC or ABAC.
This ambiguous registration is the main motivation to use the technique described in the following, that will be denoted as *redirection technique*. This method was presented in [3] for the realization of collaborative filtering on the server side. It was further developed in [62] for the realization of an adaptive visualization tool for distributed Web structures[13] and it was recently described in detail and applied in [53] for a visualization of Web usage on the server side.

[12]It is assumed that only browsing navigation is applied. In the case of the use of a search engine, the direct transition ABC{Search-Engine}D is possible.

[13]This is a publication on main aspects of this chapter

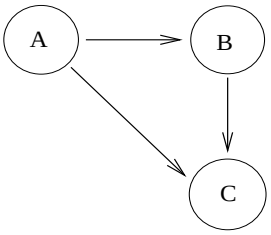


Figure 3.5: Hyperlink structure with 3 web pages A,...C and hyperlinks, AB,BC and AC

Before going into detail it has to be emphasized that the presented method assumes navigation to take place in the World Wide Web (using the HTTP protocol) and that only hyperlink activations on HTML pages are considered. The hyperlinks have to point at HTML pages and some other MIME-types, e.g. some image types (section 2.2.2 and definition 1).[14]
There are two possibilities to start the registration mechanism. Applying the first possibility, the client has to visit a certain web page, e.g. the address of the navigation tool. This address can be pointed at by the main page of an Internet server. The user then has to decide whether to use the navigation tool or not. It is also possible though, to take the main page as the starting point. Every user would then be forced to be registered by the system.
Hyperlinks on the 'start page' don't point to the original data objects directly but their addresses are substituted by URL addresses pointing to the trace server (as can be seen in figure 3.2). It is possible to supply URL-requests with additional parameters (section 2.1.2). Every substituted URL-address is supplied with the original page address, which was contained in the original hyperlink, the URL address of the page where the hyperlink is located (i.e. the web page on which the hyperlink was activated) and an identification number for a specific client. For example the address

$$\mathtt{http://server-address/origpage} \tag{3.1}$$

can be modified to

$$\mathtt{http://agent-address?server-address/origpage+frompage+id.} \tag{3.2}$$

In this example 'server-address' is the address where the original web page is located, 'origpage' is the relative address of the data object on the server.

[14]Navigation by Java Scripts, Java applets or CGI etc. is not discussed. The presented technique may also work in these cases, the implementation is however far more complex.

In the transformed URL string below, 'agent-address' is the Internet (Web-) address of the agent (trace-server) and 'server-address/origpage' is the original URL address of the data object, 'frompage' is the URL address of the page where the hyperlink is located and 'id' is the identification number assigned to a specific client. The '?'-symbol separates server- (in this case: agent-)address and the additional parameters.
In the example the request would not reach the original server at first, but it is sent to the agent (trace server). Reading the additional parameters the 'trace server' knows, what data objects are requested by the client. It can fetch the objects from the Internet in the same way as it would originally have been done by the client. The data objects aren't sent to the client directly (which would be the case for a usual proxy server), but the trace server scans the contents of the data object (HTML documents) and searches for URL addresses[15] contained in hyperlinks (section 2.1.4).[16] The addresses are then modified in the same way as it was done before for the first web page. In the first page modification step, an id-number was assigned to a client which is sent with the additional parameters in the query string. The id-number can now be re-used for the next modification step. By this means a specific client is characterized by its id-number and its individual navigation path can be registered.

It can be seen that by this means all requests of a client are redirected to

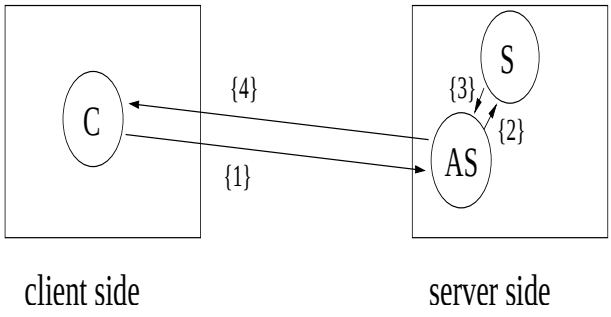


Figure 3.6: Redirection of Internet connections through an agent server component (trace server). A request is sent to the AS component first {1}. The data object is fetched from the server S ({2},{3}) and sent to the client {4}.

the agent server as shown in figure 3.6. Similar to figure 3.1 'C' in figure 3.6

[15]Only URL addresses pointing at HTML pages and some other MIME-types, like GIF or JPEG images are considered. The hyperlinks have to be marked by the HTML '< A >' tag (definition 2, [46]).

[16]This was also a main aspect in section 3.4.1

denotes the browser process on the client's computer, 'S' denotes the Internet server process and 'AS' is the agent server process, called 'trace server' in figure 3.2. The trace server component 'AS' can by this means register the navigation profile of a specific client.
For the user nothing in fact changes, since the modifications of the URL-addresses don't change the appearance of a web page. However the response speed may slow down because the way through the Internet to data objects is apparently longer (figure 3.2). This is the reason why the agent server and the considered Internet servers should be close to each other, i.e. there should be a fast Internet connection between them.
It is obvious that this procedure makes the development of an extended proxy server necessary. This tool, previously called trace server, will be described in detail in section 3.5.1.

3.4.3 Comparison of server-side registration methods

In this section the redirection technique will be compared to the other registration techniques presented in section 2.4.2 and section 3.4.2. Figure 3.7 gives a survey of the different methods and their properties. Methods for navigation registration on the server side that have been described are the 'redirection'-method (section 3.4.2), denoted as 'redir.' in the fourth row in figure 3.7, the 'access-log-file'-method, the 'ping'-method and the 'proxy'-method (section 2.4.2). The different methods have to be compared according to six attributes.
A first attribute is the ability to register requests for data objects of a client in the local Web area (column 'local data' in figure 3.7), which is desirable and possible for all registration methods. It can however be assumed that in many cases a client will leave the considered Web area and move to data objects on external Internet servers. The registration of navigation in the external Web environment is not desirable for personal data protection reasons[17], but it may be meaningful to register the navigation steps a client makes to external data objects.[18] The set of external data objects that are pointed at by hyperlinks from the considered web area will be denoted as 'border elements' of the web area. The second column in figure 3.7 shows if a registration method provides the possibility to register a client's requests of border elements. A third attribute is the possibility to register revisits

[17]It is also difficult, if not impossible to put it into practice using a similar architecture (section 3.2)
[18]i.e. on which path he leaves the considered web area

	local data	border elements	revisiting	navigation decisions	IP ambiguity	distributed area
access	+	-	-	-	-	(-)
ping	+	-	(+)	(+)	-	+
proxy	+	-	-	-	-	(+)
redir.	+	+	(-)	(+)	+	+

Figure 3.7: Comparison of different registration methods (see text). '+' denotes the presence of an attribute. '(+)' denotes the presence with limitations.

to previously visited data objects, as described in the previous section ('revis.'); a fourth attribute is the possibility to register between which data objects a navigation takes place ('navigation decisions'); a fifth attribute is the possibility to separate the navigation profiles of different users having the same IP-address ('IP ambiguity' in figure 3.7). Finally the possibility to register navigation decisions in a distributed Web area has to be compared ('distributed area').
In the previous section a possible ambiguity concerning the real navigation path in the case of multiple links to a currently visited object from different previously visited data objects was shown (figure 3.5). It is obvious that this ambiguity can't be solved by applying the access-log or the proxy server method given that caching strategies are applied in the connection between client and server. As far as the ping-method is concerned, it also depends on the applied caching strategies. These three registration methods make it therefore necessary to apply action inferring techniques as described in section 3.4.2 that provide only an estimation of the actual navigation path. The redirection technique provides the possibility to register not only the URL-addresses of requested data objects but also the data objects on which the requests were activated, which solves this ambiguity (column 'navigation decisions' in figure 3.7). If the browser's back button is activated or if the same hyperlink is activated again, these navigation steps won't be registered

by this method.
The problem that different clients may have the same IP-address if using the same provider is solved by the redirection method, too, because of the new strategy to identify clients (section 3.4.2). The IP-address identification that has to be used by the access-log, the ping and the proxy server method doesn't provide this information (column 'IP ambiguity'). It is important to note that the identification applied by the redirection technique only lasts for one session. If a user revisits the same site after a while he will be assigned a new id-number, unless he still uses a modified web page. If a client leaves the considered Web area to an external server (beyond the border objects) and returns into the web area, he will also be assigned a new identification number. These properties are disadvantages of the registration method compared to IP-identification. The system registers two navigation profiles and assumes that they come from different clients, instead of one profile from one client.
The revisiting problem can't be solved by the access-log and the proxy method. When a client visits data objects a second time, he won't be registered. In the ping method on the other hand the small, invisible page elements that are likely not to be cached may provide revisiting information depending on the caching strategies in the Internet. The redirection method registers revisits, when the navigation step to the previously visited data object is different for the same reason as described for the path ambiguity information above. Revisiting information is however not absolutely desirable, because this information makes additional network connections necessary. Thus, the redirection method provides more information than most of the other methods at the costs of an increased number of network connections (column 'revisiting' in figure 3.7).
When a client leaves the Web area, this information is not registered by the access-log, the ping and the proxy server method. As shown in section 3.4.2 this information can be provided by the redirection method. It would even be possible to 'follow' a client into the external Web area provided that the navigation is only pursued by hyperlinks. But it seems to be sensible to send the original (not modified) data object to the client, if it comes from an external source. Such a navigation decision would stop the navigation support tool. This (final) navigation step is however registered (column 'border elements' in figure 3.7).
A further important issue is the registration of user profiles in a distributed Web environment, comprising a number of servers. With the access-log method, each server stores the requests on his own. In order to reconstruct the full navigation path in the Web area, the different access-log files would have to be combined and analyzed. This process isn't necessary for the proxy

server, the ping and the redirection method since all navigation decisions can be registered by one server only, which simplifies data exploitation (column 'distributed area' in figure 3.7). Discussing the proxy method it was assumed, that there is only one proxy server in the connection between client and server necessary. It may be the case that the Web area is distributed in a way that this isn't sufficient. In this case several proxy servers are necessary and the respective registered data have to be combined.
It was discussed in section 3.3.2 that the complete information about a client's navigation is only known on the client side. With the four methods described above the navigation decisions in the considered Web area can be traced (depending on the applied method). This is in most cases only a small part of the whole navigation process (in the entire WWW). But as described in section 3.1 it was the objective to develop a navigation support tool for a restricted WEb area. For many navigation support applications like visualization of the local hyperlink structure or a history visualization of navigation decisions in the Web area it is sufficient to know the path in the considered Web area. For the development of estimation techniques for the relevance of data objects as described in the next chapters, it can however be important to know a client's navigation decisions in external areas of the Web as well. These decisions could contain additional information about a user's interests. This information is however not available for principal reasons using this architecture (section 3.3.2)[19] and therefore cannot be exploited.
We have assumed so far that navigation takes place only by activation of hyperlinks. It may also be possible that a user uses a search engine from time to time [51], [18]. In this case, the registration would end and after having used the search engine the client would be assigned a new id-number. The effect is, that two short navigation paths are registered and assumed to come from different clients instead of one long profile from one client, which is a mistake of the registration system.
This problem can be solved by providing a local search engine that works in combination with the agent software.[20] Query results, usually presented as ordered lists containing hyperlinks to relevant data objects, are modified in the same way as done by the agent. The pages are parsed and URL-addresses are substituted by the agent's address with the described additional parameters. By this means this problem can be solved and the navigation process of one client in the considered area can be fully traced.[21]
As far as the technical realization of the different registration methods is con-

[19] At least the navigation steps before the entrance into the web area can't be traced.
[20] The communication through a global search engine could be traced, too. This is again problematic because of data protection reasons.
[21] The client has got to use only the local search engine.

cerned it must be noticed that the modification step is in practice very complex because of the number of different formats in the Internet as described in section 2.1.4. As shown in section 3.5.3 it has so far only been implemented for HTML-data. In the case of other (script-)languages like Java, Java-Script etc., further considerations have to be made. Yet this problem is not of a principal nature since it is possible to extract partial functionalities of the respective language interpreters or compilers. The same considerations have to be applied for the development of spider software as described in section 3.4.1, where hyperlinks on web pages are extracted in order to find Web data objects.
It can be summarized, that the redirection method provides more information about a client's navigation behavior than the other methods at the cost of additional network communication and a more complex software structure. The information that is registered on the server side using this method are sequences of different[22] navigation steps between data objects in the considered web area. Each sequence characterizes one navigation session of a client. The navigation steps may only be a subset of the actual set of navigation decisions.
These considerations are a decisive basis for the subsequent chapter 4, since pattern recognition techniques that may be applied, strongly depend on the underlying data types.

3.4.4 Visualization of navigation support information

There are many possibilities for a navigation support tool to present support information as discussed in section 2.3. The techniques that can be applied for this visualization strongly depend on the software architecture (section 3.2). If a software or a browser extension is running on the client side there are no restrictions concerning the visualization. This is the case for the systems [28] and [114] applying the architecture 2 or 4 in figure 3.1. However in the system presented in [3] (which applies the 3. architecture in figure 3.1) only modifications on current web pages in the client's browser window or a generation of new web pages is possible. In [3] e.g. hyperlinks on the current web page are highlighted, if they were estimated to be relevant by the support system. It is obvious that complex visualizations like local hyperlink views or adaptive site maps can hardly be presented when this method is applied. The web pages would become very large and probably very confusing. The generation of new (dynamic) web pages showing support information would require additional navigation steps by a client and thereby increase disorien-

[22] depending on the caching procedures

tation. It is a common practice to use different window(s) in addition to the main browser window in order to present support information [68], which requires an architecture similar to 2 or 4 and thus makes a software installation on the client side necessary. As described in section 3.3.2 it was in our case the objective, not to make a software installation on the client side necessary for usability reasons.
The solution presented here makes use of modern Internet browser capabilities described in section 2.1.4 that are used by a vast majority of the Internet users. The browsers have to be capable of interpreting Java code. This language makes it possible to offer a user complex information visualization as it is necessary in our case. Basic capabilities are the creation of new windows apart from the web page in the browser window and communication between different windows. The visualization code doesn't have to be installed on the user's computer by the user but it is downloaded in the same way as a common HTML-page and it is automatically installed on the computer by the browser (section 2.1.4).
The code necessary for the presentation and visualization of navigation support can be loaded from a server every time the user visits a new page. The required software actually doesn't change for different support views but only the presented information will have to be updated. The usual navigation process from one data object to another would remove the old data object, including the visualization code and replace it by the new data object. This problem makes control of the navigation process by the temporarily installed software necessary as described in the following section.

3.4.5 Control of a navigation process

In this section it is the objective to describe a technique developed to make a refined presentation of support information on the user side possible. The user will not be forced to install an additional software on his computer himself. Instead, a Java applet is sent to the user and is automatically installed by the browser software to provide visualization functionalities. In order to avoid loading this Java code every time a user visits a new web page, the presentation code is installed persistently throughout the navigation process on the user's computer.
A second objective is to automatically update the navigation support presentation. Usually a client sends a request to a server in order to get a data object. With more refined methods it is possible to send data from server to a client without initiation by the client. In terms of agent technology this process is important to accomplish autonomy of the agent. The system shall be provided with the ability to present the user information if it decides

this to be relevant for the client. The possibility of an initiation of the data presentation on the client's screen by the agent on the server side is a first necessary step towards this autonomy.
The idea described in the following depends decisively on abilities of modern browser software. One important issue is the frame concept as described in section 2.1.4. Another attribute of the browser software that must be available, is support for the Java-language. A Java applet will run in a 'sand-box' on the client's computer that is controlled by the browser software (section 2.1.4).

The first page that is sent to the client when the navigation support starts consists of two frame components. One of these components is needed for presentation purposes, it shows (in the usual case) the current web page that is visited by the client. The other frame component not being visible (it has minimal size) contains the Java applet. This Java applet has two tasks. The applet provides the functions needed to visualize the support information in a second window. And it controls the presentation in the other frame component, in the common case the currently visited web page. This second task is organized by communication with the agent server.
If a user activates a hyperlink on the web page in the visible frame component a HTTP request is sent to the agent server, because of the previously described link modification (section 3.4.2). This request is shown in figure 3.8

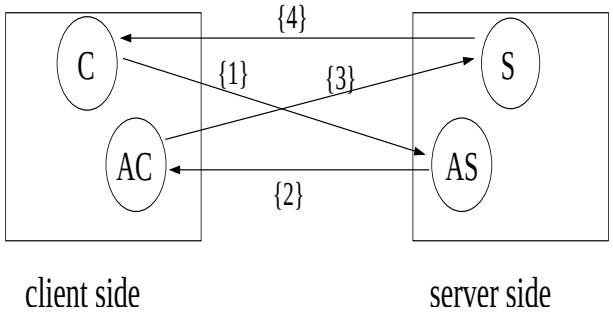


Figure 3.8: Communication sequence between client 'C' and server 'S' and the two agent components 'AC' and 'AS' to achieve a persistent activation of the applet code throughout navigation in the considered Web area.

as arrow 1 from the client process 'C' to the agent server process 'AS'. The agent component 'AS' however sends the request information to the agent component on the client side, i.e. the Java applet ('AC') in the invisible frame component of the client's browser (arrow 2). With this information the applet is able to initialize the presentation of the requested data object

by using the Java command [26] (arrow 3):

$$\mathsf{showDocument}(\mathsf{url}, \mathsf{string}), \tag{3.3}$$

The string 'URL' is the URL-address of the requested data object and 'string' is a presentation directive, forcing the browser to load the data object into the other frame component after it has been sent from the server to the browser process (arrow 4). The initialization of the download (arrow 3) will be refered to as 'download' or 'final request' in the following.

By this means the applet running in the invisible frame component of the

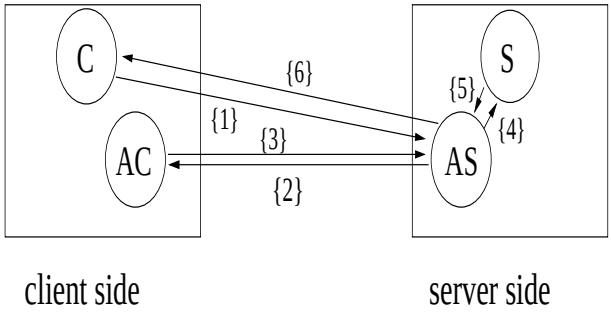


Figure 3.9: Communication sequence between client 'C' and server 'S' and the two agent components 'AC' and 'AS' to achieve a persistent activation of the applet in combination with client tracing.

client's browser is, in connection with the agent server, able to initiate loading of data objects into the visible frame component. The applet itself thus won't be removed when a new data object is loaded. It is stored throughout the whole navigation process and can therefore be used for support information visualization during the navigation session of a client in the considered Web area.

A second connection between the client's agent component AC and the agent server AS, not shown in figure 3.8, can be used to update the support presentation. This connection makes an initiation of the presentation by the agent possible as described above. It is 'introduced' to the client system by the client's agent component, communicating with the 'AS' component on the server side.

In combination with the redirection procedure described in section 3.4.2 (figure 3.6) the Internet communication has now the structure shown in figure 3.9. In contrast to figure 3.8 the connections {3} and {4} are redirected through the AS component, represented by the connections {3}, {4}, {5} and {6} in figure 3.9.

The different components on the client side (AC) and on the server side (AS) necessary to realize the communication will be described in detail in the following sections.
The (software) requirements on the client side necessary for this communication are more extensive than in the 1. and 3. architecture in figure 3.1. The browser has to be able to process the Java code and the Java support has to activated. It is possible for a user to deactivate Java in the preferences of the browser. In this case the navigation tool doesn't work because the 'AC' component can't be installed by the client's browser. The client then receives the message from the agent server that the activation of Java is necessary.
The requirements are much less extensive than in the general case applying the 2. and 4. architecture in figure 3.1 where an additional installation is necessary. The described system makes a much more refined presentation of navigation support data possible than it would be the case applying the 1. and 3. architecture in figure 3.1.

3.5 Components and realization

This section will describe the different components of the agent system and give details of the technical realization.

3.5.1 The 'trace server'

In the following, the software component on the server side, developed to realize the navigation support is described in detail. In contrast to figure 3.9 this component is in the described case not directly located on the server computer (i.e. one of the server computers since we consider a distributed Web area)[23] but it runs on a computer somewhere in the Internet close to the considered server computers as shown in figure 3.2. 'Closeness' between the agent machine and the server machines here means that there is a fast TCP/IP connection between them.
One first task is to analyze the considered Web structure as described in section 3.4.1 and to store this information in a database. This step can be pursued off-line and the resulting data will be assumed to be available in a database.
Two main tasks to be performed online by the agent component on the server side are the registration of the client's navigation, the redirection preparation described in section 3.4.2 and the communication necessary to control the client's navigation in section 3.4.5. These are the basic technical tasks

[23] Thus the right to execute programs on these computers is not required.

for the navigation support. Further questions are then how to exploit the registered information and how to generate the adaptive support information for the client. These questions will be discussed in section 3.6 and the following chapters.

The structure of the processes of the agent server component is shown in

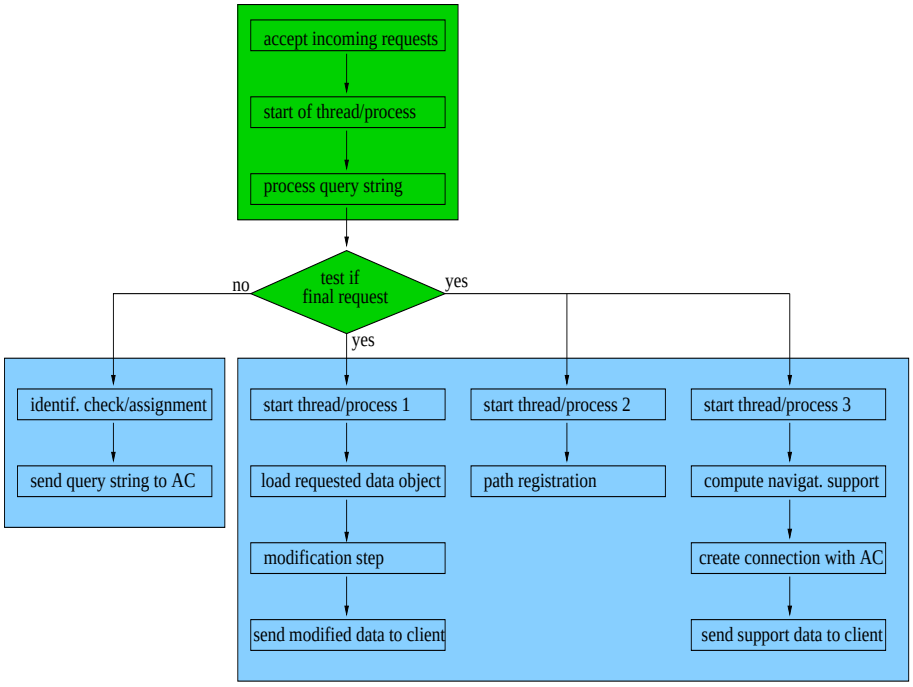


Figure 3.10: Sequence of procedures of the agent component on the server side (trace server).

figure 3.10. The figure has a strong connection to figure 3.9 since the server agent component 'AS' operates in combination with the client's agent component.

The first step to be performed by the AS-component is to wait for incoming requests from a client. Those requests are redirected HTTP-requests as described in section 3.4.2 and shown in figure 3.9 (first arrow) or download (final) requests from the AC component (third arrow). A new thread or process is then started to serve this specific client. This is necessary to be able to serve multiple clients at the same time. The incoming query string is then analyzed. According to the type of the request either the procedure block on the left in figure 3.10 or on the right is activated. A redirected HTTP request from the browser process 'C' (arrow 1 in figure 3.8 and 3.9) activates procedure block on the left. It is tested if this is the first request from a

specific client. If this is the case a new identification number is assigned to the client, if not, the former id number is reused. The signal is then sent back to the AC component, supplied with the respective identity information.
A download request ('final request' in figure 3.10) initiated by the AC component activates the procedure block on the right. Three different tasks have to be performed that are independent from each other and can therefore be pursued in a parallel manner, e.g. by different threads. One task is the web site modification. The requested page has to be downloaded from the Internet. The page is then modified as described in section 3.4.2 and is then sent to the client's browser process. A second task is the registration of the client's navigation path. Knowing the identification number from the incoming query string, the navigation information can be added to the information about a client already registered by the server. For this purpose a connection to a database (i.e. component 'knowledge' in figure 3.2) is established and the data are written to this database. The third task is the navigation support generation. Depending on the current client's position in the web space, his former navigation behavior and the navigation profiles of other users as registered in the 'knowledge' database, and the knowledge about the considered web space, that is also stored in the 'knowledge' database, the navigation support is computed. A simple support presentation will be shown in section 3.6, more advanced methods will be introduced in the following chapters. The support information is then sent to client's AC component.

3.5.2 The client's applet

This section deals with the client's agent component 'AC', the counterpart of the trace server on the server side. It was described in section 3.4.5 that at the beginning of the navigation support a Java applet is sent from the trace server to the client and is installed by the browser software in a frame component of the browser that is not visible to the client. According to section 3.4.5 (figure 3.9) the AC component has mainly two functions. One is to accept download requests from the server component and to initiate the download process of the browser software (section 3.4.5). The second task is to accept navigation support information and to visualize this information. Examples for this visualization will be given in section section 3.6.

The structure of the client's agent component can be seen in figure 3.11. The first step is the initialization of the applet software, e.g. the creation of connections to the trace server. The AC component then creates a new window responsible for the support visualization. The two main tasks are independent from each other and can be run in a parallel manner e.g. realized by different threads as shown in the figure. One thread listens for

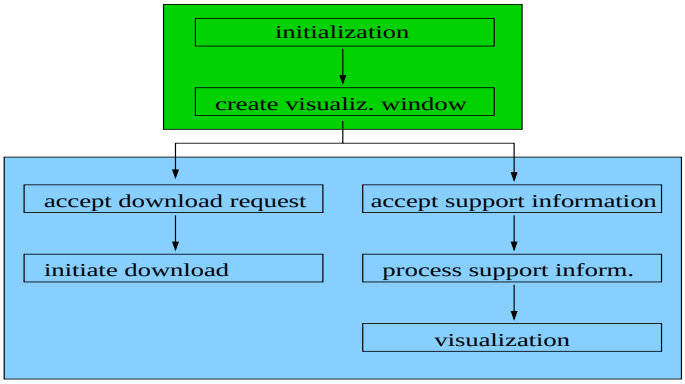


Figure 3.11: Procedures of the client's agent component.

(download) requests coming from the trace server (figure 3.9, connection {2}) and initiates this download (section 3.4.5). The other thread waits for support information. This information has to be processed for visualization purposes. However the main layout information has already been computed by the server. Then the visualization in the external window has to be controlled.
It can be seen that the client's agent component is (potentially) a very small program. Most of the tasks necessary for a support presentation are pursued on the server side. The advantage is a fast transport of the program to a client and a fast (automatic) installation. However all of the objectives described in section 3.1 that would not be possible using the first or the third architecture in figure 3.1 can be realized with the restrictions discussed in section 3.3.2.

3.5.3 Technical realization and problems

The described software agent was developed in Java and C^{++}. The main component, the agent component on the server side (trace server, section 3.5.1) was developed in Java. One component, the module to compute local maps and the matching and prediction component is realized as an own process, that was developed in C^{++} and communicates with the trace server by socket connections.
The client's applet was developed in Java, here it was necessary to consider the software constellation on the client side. Java is supported by most browsers and the Java applet was tested for Netscape (version 4.7) and the Internet Explorer on Solaris, Linux and Windows machines. Even with Java, this portability is not trivial and some details in the socket communication

had to be adjusted. One main component of the system, the URL parser for web pages, that is the core module for the redirection technique, was developed for HTML pages (including frames). Hyperlinks have to be marked by the '< A >' tag. Only hyperlinks to HTML pages and some basic MIME types are considered (like GIF, JPEG and other image types). The developed module is a prototype. Due to the vast amount of different rules used (even) in HTML, a stable version requires much effort (and costs) especially in order to test the software. Other languages (section 2.1.4) have not been considered. The effort to realize a parser that takes into account all languages is huge.[24] It may however be possible to make use of the original parser software provided by the Mozilla sources [106]. Since search engines depend on the same functionality as described in section 2.4.1, it may also be possible to apply modules that were already developed in this context.

3.6 Visualization examples

The software was applied on the web site of the Cognitive Systems Group, University Kiel in 2000 and 2001 (http://www.ks.informatik.uni-kiel.de). Figure 3.12 shows the original home-page of the research group, visualized by Netscape Navigator 4.7. The (original) web page contains two frames. The frame component on the left has, near the top, a hyperlink with the anchor text 'Navigation Tool'. An activation of this link creates a connection to the trace server. Figure 3.13 shows the constellation of windows and frames after this hyperlink has been activated. On the left of the two original frames, a new frame has been added, that provides some buttons to control the presentation (control frame). The presentation window is an external window (on the left in figure 3.13). At this state, it only shows one data object, the (URL-address of the) current web page in the browser window. If the user activates a hyperlink in the browser window, in this example the 'staff'-hyperlink, the result can be seen in figure 3.14. The next data object (the URL-address of the staff page) was automatically added in the visualization window, together with the hyperlink (the line between the objects). The two additional objects are the two frame components of the first web page (this information should possibly be deleted in future versions). Figure 3.15 shows a possible history presentation after a number of navigation decisions. The current position of the client in the web area (which is shown in the main browser frame), is the green rectangle in the structure window.
The structure that can be seen in the external window reflects the exact knowledge, the agent component on the server side has about the client's

[24] This is one main disadvantage of the redirection strategy.

navigation path. The graph information that is presented is sent to the client from the AS component. One problem becomes clear, when e.g. the browser's 'back' button is activated in the state in figure 3.12, which loads the previous web page into the browser window. The system still highlights the 'staff' page (green color) as the current position in web space, which is obviously wrong. If, after this transition, a navigation step to a new (not yet visited) web page is done, the system will know (and visualize) the current (true) position of the user. This problem results from the caching problem described in section 3.4.2 and 3.4.3. The browser stores the first web page after the first visit and doesn't reload it (from the external source), when it is requested again. The page is reloaded from the cache. Thus the AS component doesn't acquire this information. The described problem is one main reason, why user behavior will be modeled by sets of navigation steps and not sequences in the next chapter in section 4.1.
It can be seen in figure 3.15 that the layout of the navigation history can become quite complex (section 2.4.3). The applied algorithm for the graph layout can greatly be improved, in order to make the structure easy to understand. One main difficulty is not to change the layout every time, when the graph structure changes (which is still the case). When a new navigation step has to be added, the layout algorithm should take into account the previous layout since the user may already have got used to it.
Two visualization examples of the local hyperlink structure can be seen in figure 3.16 and 3.17, which are created by the system, when the *surround* button in the control frame is pressed. Figure 3.16 shows the set of web pages (and other web objects) that have the same depth[25] as the currently visited web page, which is the 'staff' page. It is marked by a different color (green) in the structure window. The structural knowledge was computed by the trace server (more precisely the navigation support component in figure 3.10), using the knowledge about the client's current position in web space and knowledge about the hyperlink structure of the Web area, which was previously extracted (off-line) by the method described in section 3.4.1 and stored in a database.[26] Figure 3.17 shows a similar structure view from the same position in the web space (the current position is again the green rectangle). In this case the structure view shows the current and the previous navigation possibilities. This structure view depends on the hyperlink structure, the current position of the user and the navigation history.

[25] i.e. the number of hyperlink transitions from the main page.

[26] This particular visualization could also have been provided by a system without previously stored structural knowledge, since it is only necessary to parse the previously visited web page for hyperlinks, which can also be done at the client side. The presentation is only used to demonstrate, how the system works.

The presented structure visualizations may e.g. be helpful, if a user wants to 'scan' a list of web objects in the web space, that are pointed at by a single web page. The structure views give him an idea of his current position in web space.
Many different structure views are thinkable that have not been implemented yet. The same problem as in the case of navigation history visualization however easily occurs, i.e. the layout of the graph, which is in this example even more complex due to the higher number of web objects to be shown. If the hyperlink structure is presented up to a higher depth, the number of data objects to be shown is likely to increase exponentially for many web sites.
Both presented visualizations are adaptive with respect to the user's behavior. The first is adaptive with respect to former navigation steps of a user. The second is adaptive with respect to a user's current position.

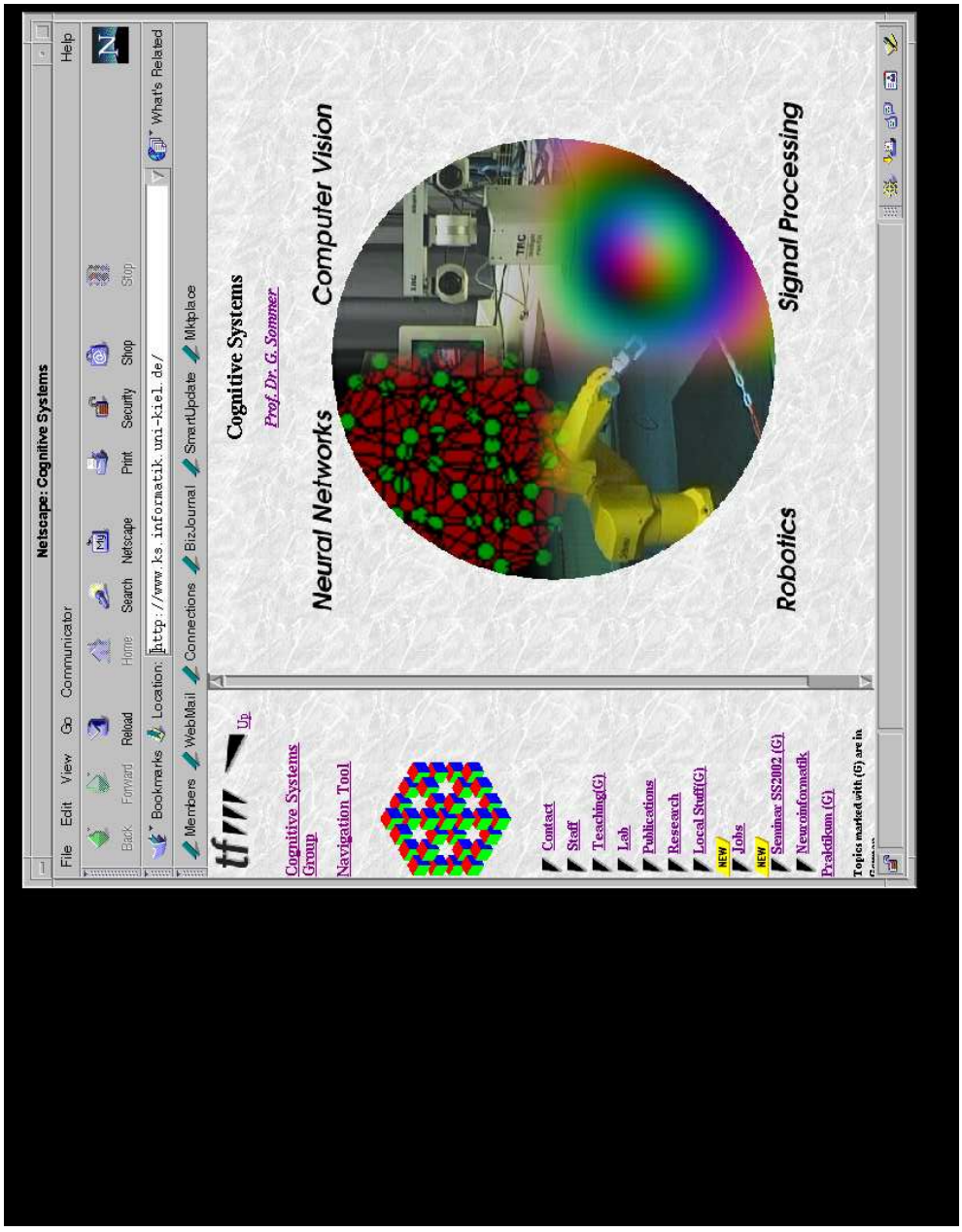


Figure 3.12: Original web page of the Cognitive Systems group, University Kiel. The second hyperlink from the top in the left frame component ('Navigation Tool') activates the navigation support.

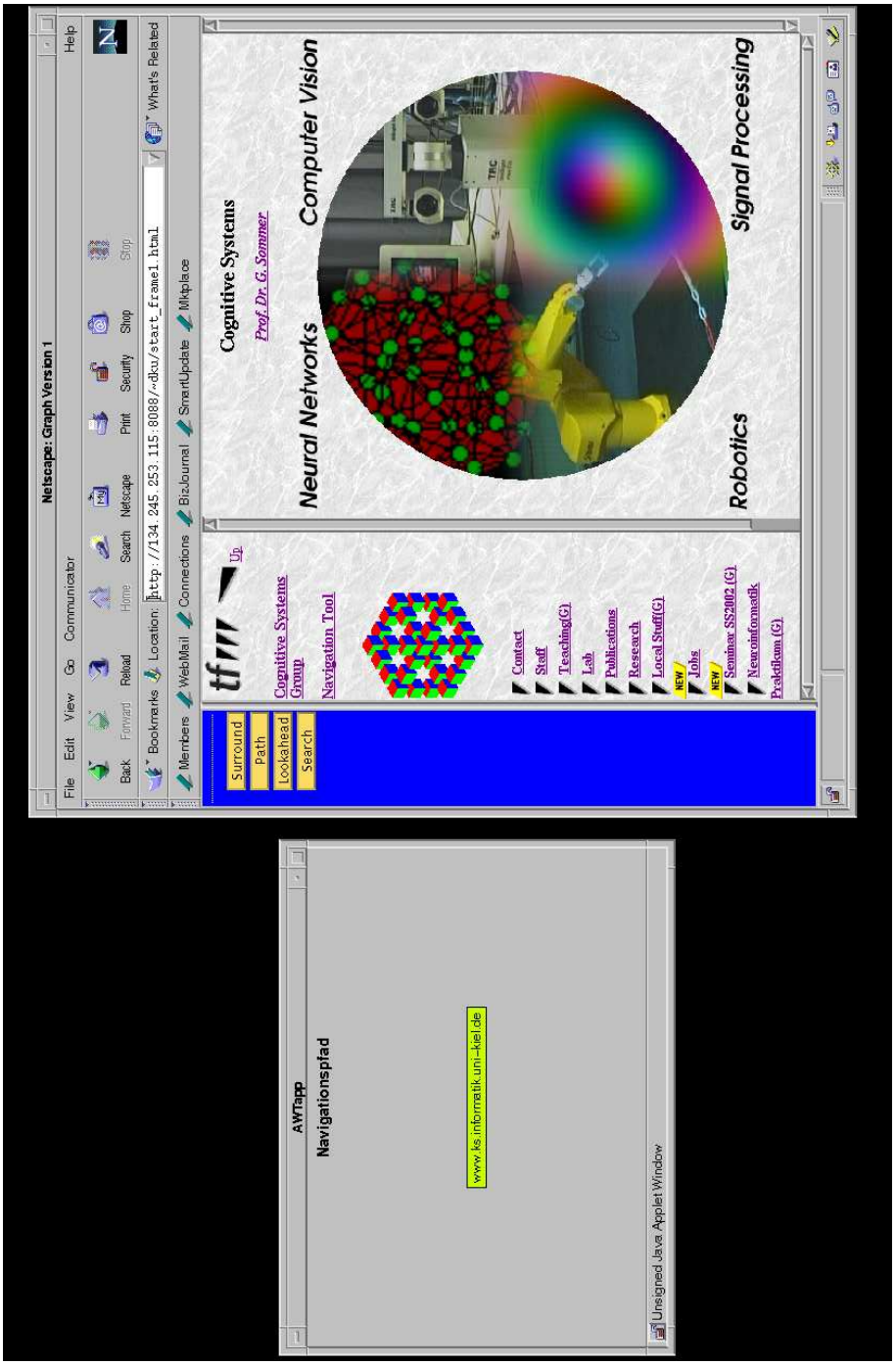


Figure 3.13: The window constellation after the activation of the support tool. The window on the left shows a rectangle with the URL of the current web page in the main browser window. There is an additional control frame in the browser window (blue). The 'staff' button that will be activated in the next step is the fourth hyperlink from the top in the left frame component of the original web page.

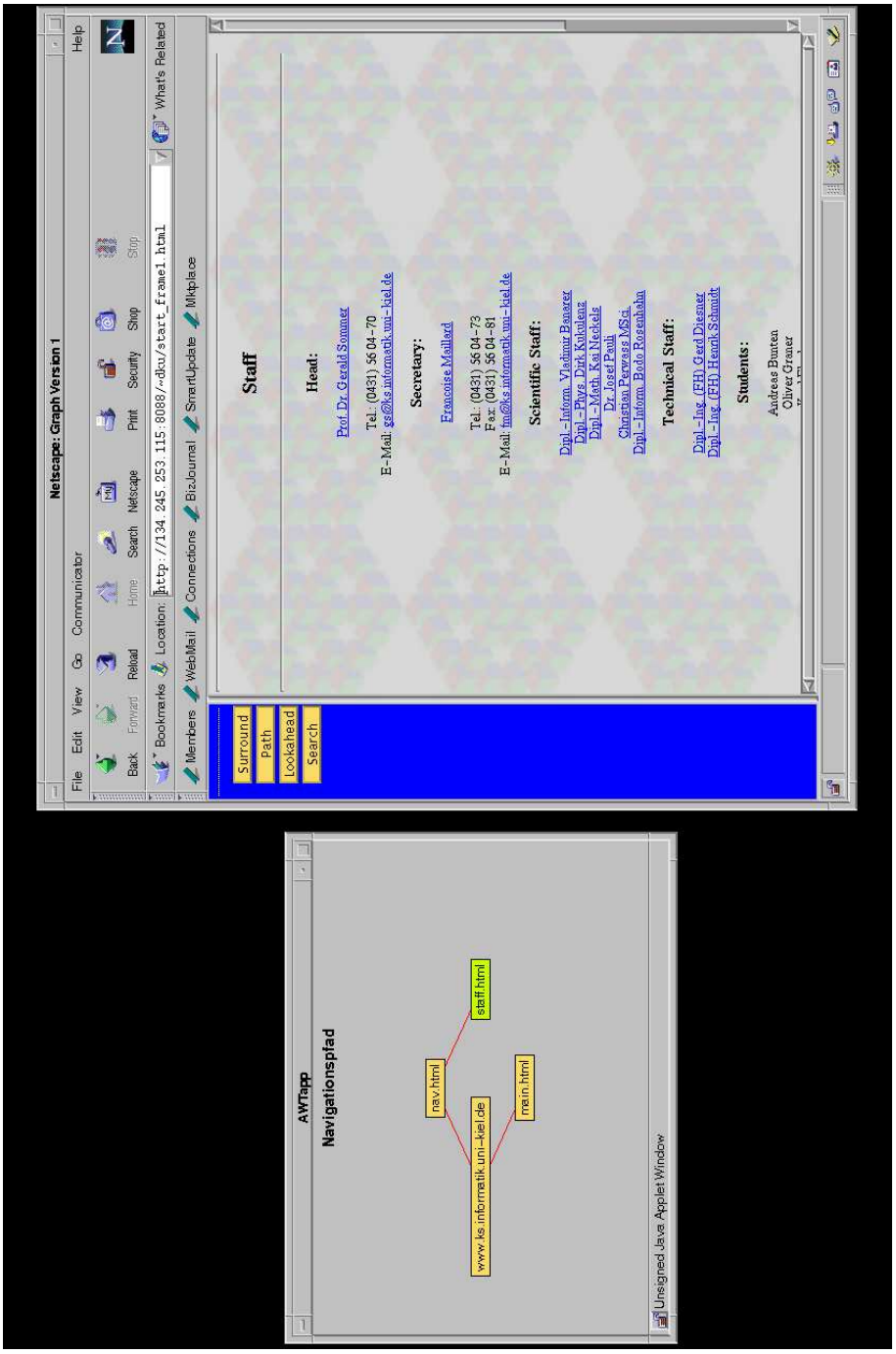


Figure 3.14: Navigation path visualization after the first navigation decision, an activation of the 'staff' button. The window on the left shows the navigation history (with two frame components 'nav.html' and 'main.html' of the previous page). The green rectangle shows the URL of the current page in the browser window.

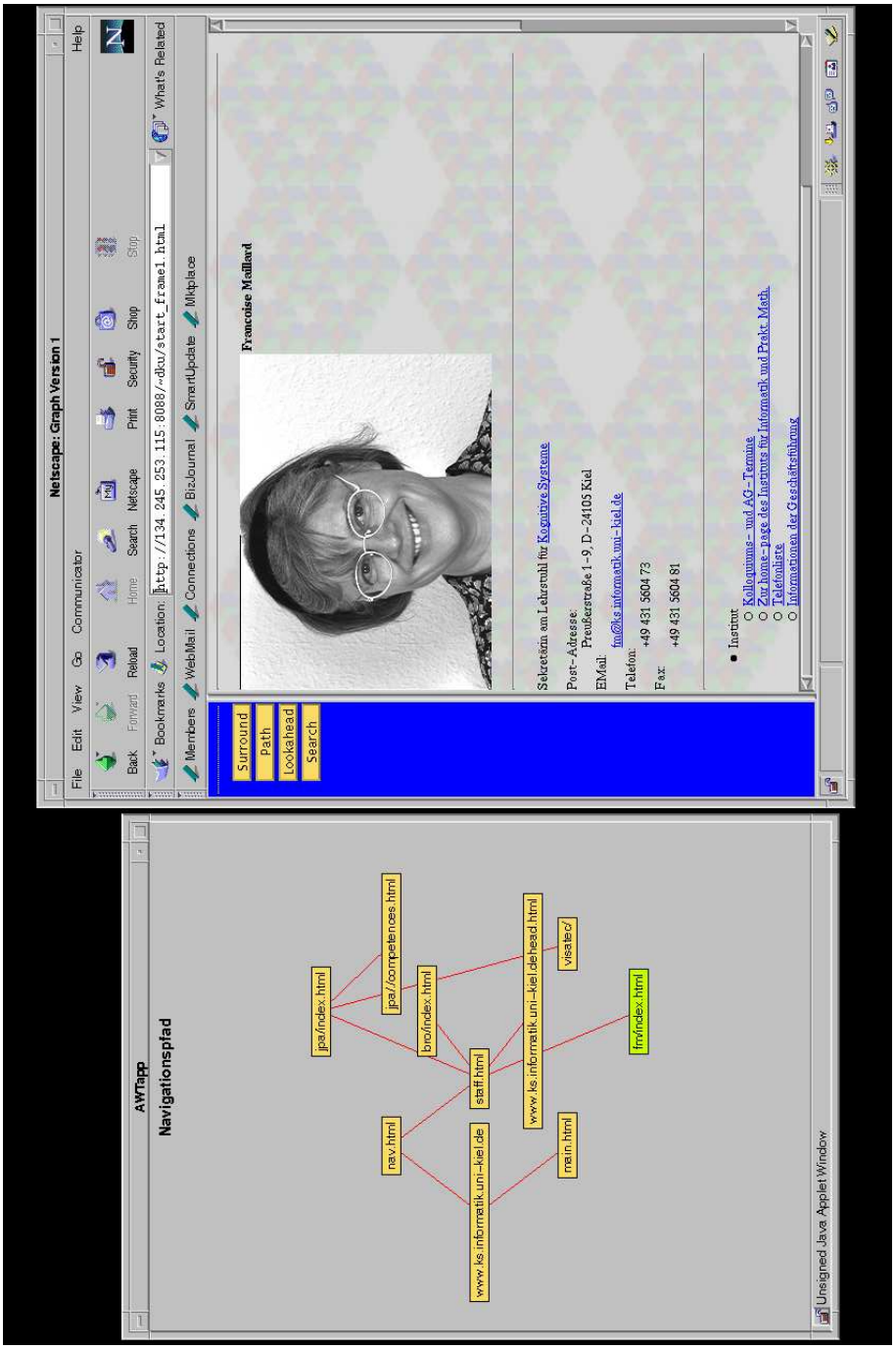


Figure 3.15: A possible visualization of the navigation history after a number of navigation steps.

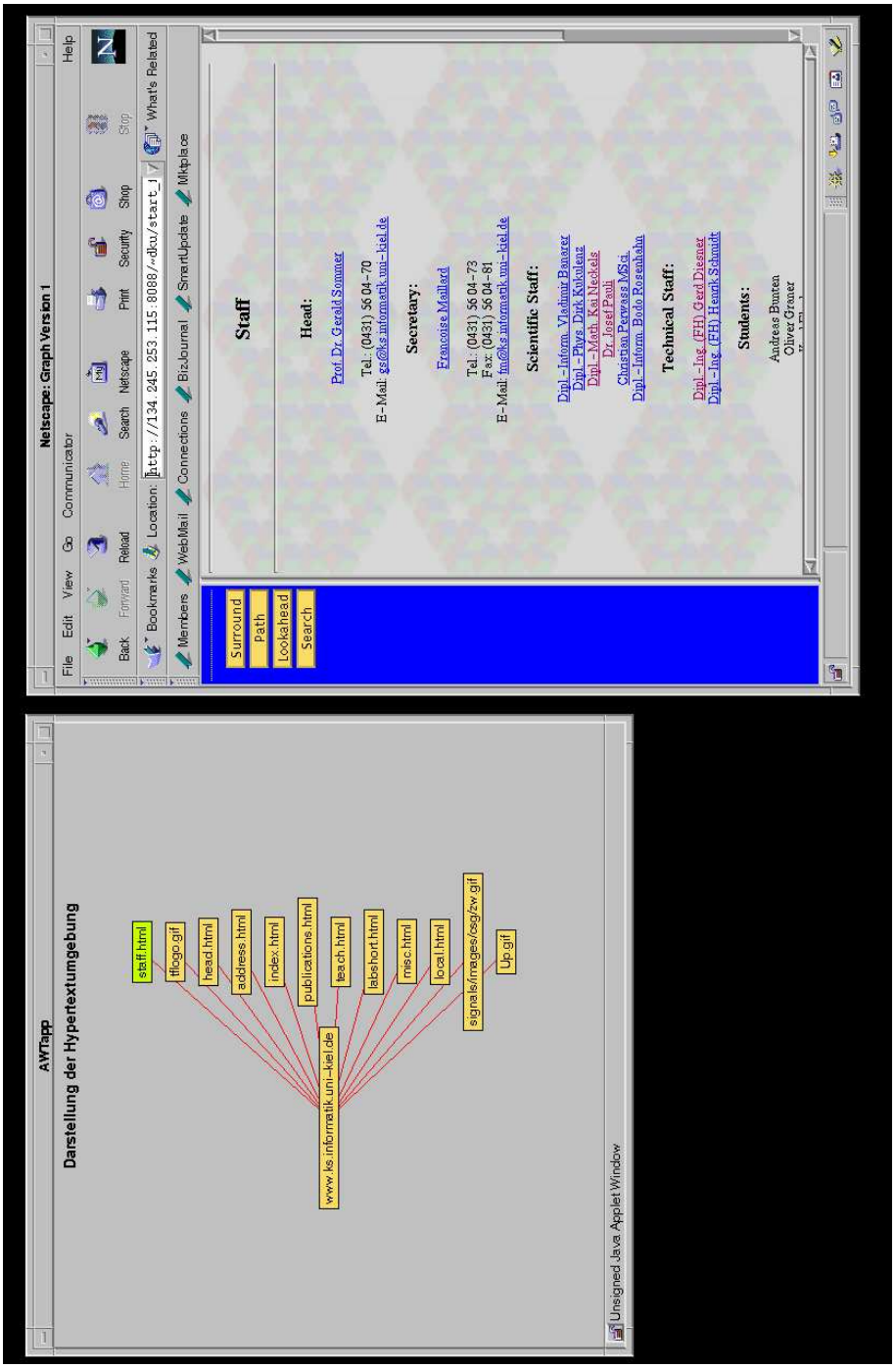


Figure 3.16: View of the local hyperlink structure, showing the current position (green rectangle) and hyperlinks at the same depth with respect to the main page after an activation of the 'surround' button in the control frame.

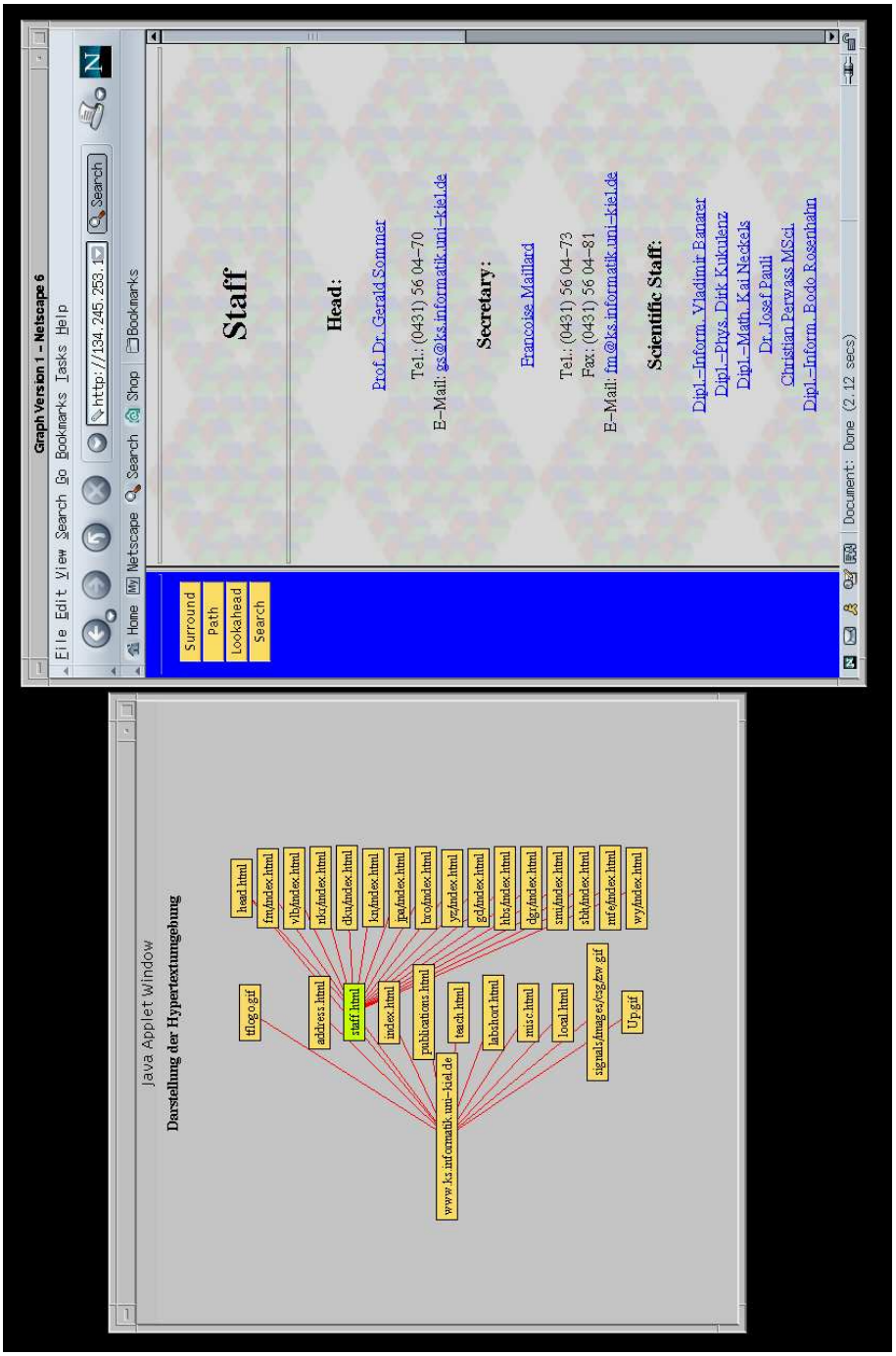


Figure 3.17: A similar structure presentation as above from the same position in hyper space. In this case the structure view shows the current and the previous navigation possibilities with regard to the client's navigation history.

3.7 Summary and discussion: Internet Agent

In this chapter a software system has been presented that makes it possible to realize a navigation support for users in the WWW. It was the objective to provide a support for navigation in a well-defined, restricted Web area consisting of a number of web servers, which are known to the system. At first a new architecture was presented in section 3.3.1 that is a mixture between software agents running on the client side and systems that run on the server side. The described software system benefits from both architectures. The capabilities of the described system to visualize information are (theoretically) similar to a system working on the client side. However a (partial)[27] registration of the client's navigation process on the server side makes collaborative filtering possible which is one main advantage of navigation support systems working on the server side. Collaborative filtering can be made use of when generating a navigation support, as will be shown in the following chapter.
The presented system was realized by making use of two main ideas. One idea is to redirect all URL requests from a client through the agent's server component as described in section 3.4.2. By this means the navigation registration on the server side is made possible. A second idea is to apply the ability of common Internet browsers to run 'Java applets' in a 'sand-box' (section 2.1.4). The developed applet (i.e. the client's agent component) is loaded from the server and installed temporarily; it only runs throughout the client's navigation process and is removed afterwards. It was shown in section 3.4.5 how this 'persistent'[28] storage of the client's agent component throughout the navigation process can be realized by controlling the navigation process. This control makes additional communication between the client's applet and the server's agent component necessary.
A technical realization of the system was presented, too, showing visualizations of support information in section 3.6.
One important attribute of the system is the possibility of an autonomous presentation. The client doesn't have to reload the support information as it would be the case if the information was stored on common web pages. The agent itself is able to initiate the presentation of information. Another advantage of the system is its usability for most of the Internet users without the requirement of an (active) software installation in contrast to many other systems (section 2.3). This is an important aspect since it is in most cases not realistic to expect the visitor of a web site to install a software on his

[27] i.e. navigation in the considered Web area
[28] with respect to the navigation process

computer. Such a requirement will only become accepted by users when a large number of other Internet servers require the same software or plug-in, as it is e.g. the case for 'flash' [110]. Another attribute of the system is the possibility to provide a navigation support covering a number of servers. The data of these servers don't have to be modified which makes it easy to install and to supervise the system.
One main aspect with regard to the following chapters is the information about the navigation of a client that can be registered. As discussed in section 3.4.3 and shown in section 3.6 the system makes it possible to register the sequence of different navigation steps of a client.[29] This information will we made use of when developing collaborative filtering techniques in the following chapters.
One disadvantage of the described system is the requirement of Java on the client side, which is supported by most browsers, but which may still contain security risks for the client. Another disadvantage is a more complicated communication between client and server. If a firewall is activated on the client side, the establishment of some connections may not be possible. More advanced techniques like HTTP-tunneling might have to be applied here.
One main drawback of similar systems like the one presented here is the restriction of the support to a well-defined Web area. It would be desirable to offer the navigation support in the whole Internet, which is not possible when the described architecture is applied. However the web area may be extended, if similar agents work in different web areas that communicate and exchange the respective user and support data. Such a multi agent system would however be far more complex.
In the next chapter it is examined, how the data about the user behavior that is registered by the presented system can be used for collaborative filtering purposes. Later, the registered data type is used to develop a prediction method for future user behavior. At first a mathematical model has to be developed on order to work with the respective data.

[29] in the worst case, i.e. every request is cached

Chapter 4

Pattern Recognition in Graph Spaces

This chapter describes the theoretical framework that will be applied to learn distributions of graphs in chapter 5. In the previous chapter a software system was presented that makes it possible to register the navigation behavior of clients in a well-defined (restricted) Web area (section 2.1.1) on the server side. In this chapter a mathematical model is presented to characterize the registered navigation behavior of users. The concept of a distribution of user profiles with respect to the model will be introduced. A collaborative filtering technique is then presented to learn the distribution of user profiles based on a set of profiles that was registered by the system.

4.1 Model definition and notations

It was shown in section 3.4.3 that in the case that all requests are cached in the connection between client and server the data, that is registered by the system presented in chapter 3, are sequences of different navigation (browsing) steps between data objects. This is the minimal information about a user's navigation path the system registers. In the case of different caching strategies (section 3.4.2), more information may be available. As previously described, these data can be used to estimate the actual navigation path, applying action inferring techniques [2], [53],(section 3.4.2), that take into account the web site's hyperlink structure. These methods may provide estimations of the 'real' sequences of navigation steps.
Because this method may not be exact as described in section 3.4.2, in this thesis the *sets* of navigation steps is the information used to model the clients' navigation behavior. These sets do not contain reoccurences of navigation

decisions and the information about the actual sequence of navigation steps is lost. Different caching strategies or action inferring techniques that may result in different registered navigation decision sequences usually imply a registration of the same set of navigation decisions (section 3.4.2). If e.g. the real sequence of requested data objects is $\mathtt{abcbd}$, the registered sequence e.g. in the access-log file may e.g. be $\mathtt{abcd}$, if the browser caches every data object. Depending on the action inferring strategy and the hyperlink structure, different estimated sequences are possible. The set of (undirected) navigation decisions is $\{(\mathtt{a},\mathtt{b}),(\mathtt{b},\mathtt{c}),(\mathtt{b},\mathtt{d})\}$, which isn't affected by the browser's caching strategy. Therefore, taking sets of navigation decisions as the basic data type to model user behavior, indifference according to caching techniques in the Internet is improved and the application of action inferring is not necessary. The sequence information is lost, but it may in many cases be reconstructed, as in the case of a sequence of data objects $\mathtt{abcd}$, which results in a registered set of navigation decisions $\{(\mathtt{a},\mathtt{b}),(\mathtt{b},\mathtt{c}),(\mathtt{c},\mathtt{d})\}$.
The registered part of a navigation as defined in section 2.2.2 takes place in a certain time interval, which will be called a *user session time(-interval).*

Definition 4 *A user session time(-interval) is the time interval needed for a client's actions in a well-defined Web area between the entrance time and the time when the area is left, as registered by a software system. It has to be smaller than a finite time interval, the 'maximal dwell time', 'm.d.' time. (In the other case it is the m.d. time).*
In the case that the client doesn't explicitly leave the area, it is the time interval beginning with the entrance time and ending with the time of the last action in the 'm.d.' time interval.

This definition takes into account that the presented software system registers the entrance of a client into the Web area by means of the 'id' parameter contained in the query string of a redirected HTTP request (section 3.5.1). The exit point and time can be registered as described in section 3.4.3. Yet, the client doesn't have to leave the Web area explicitly but he may just stop requesting new data objects. The system will then, after the m.d.-time, decide the user session to be 'complete'.
A user session can now be defined as follows:

Definition 5 *A user session, also refered to as user profile or user pattern is the set or sequence of a user's actions as registered by a software system that take place in the user session time interval.*

In the common case it is not possible for the described software system to identify a client after he has left the considered Web area and returns to

it (section 3.4.3). The user model can therefore only take into account one session of a specific user. Different methods would have to be applied to provide this revisiting information like Cookies, password identification etc., that have not been considered here.[1] Each user session is therefore used to characterize a different user, although a number of user sessions may originate from the same person.

4.2 Distortion concept

This section describes the basic assumption upon which the developed intelligent system is based. A fundamental question when analyzing user profiles are the reasons, why profiles differ from each other. At least three factors will have an influence, as shown in figure 4.1. The first factor is a user's goal or intention, i.e. his information need. There exist at least three different kinds of goals like exploring an Web area, the search for concrete data objects or the answer to a complex question by means of a navigation between data objects as described in section 2.2.2. Other factors, that have an influence on a user profile, are a user's knowledge of the specific problem domain and his experience with respect to the search process. These factors together will be refered to as *user setting*.

As presented in [18], different user settings result in a different behavior of

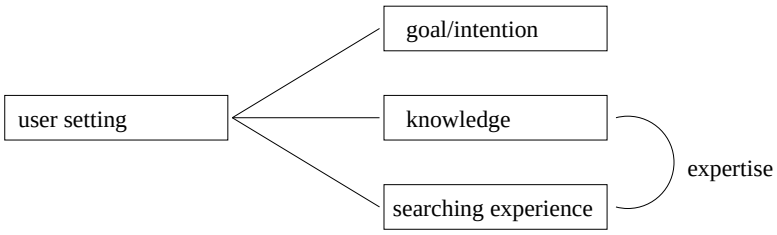


Figure 4.1: The behavior of a user depends on his goal, the knowledge about the domain and his searching experience (expertise). These factors together will be denoted as *user setting*.

users concerning the use of a hyper-media database. A first striking difference can be seen in the behavior of experts and novices with respect to the contents of the data. Similar differences are also presented in [51] for searching behavior in the Web. An example for this difference is the observation,

[1] Drawbacks of these methods are that they are too complicated to use and their applicability (e.g. cookies are in many cases not activated in the client's browser software).

that experts tended to browse fewer topics in more depth than did novices. They were not interested in changing topics until they completed review of the current topic. Novices tended to rely more than experts on referential links. It is one important aspect in the behavior of novices that they seemed to concentrate on understanding the basic structure of knowledge, and often switched topics to do so.

Figure 4.2 shows these results in the general case. Different users who have

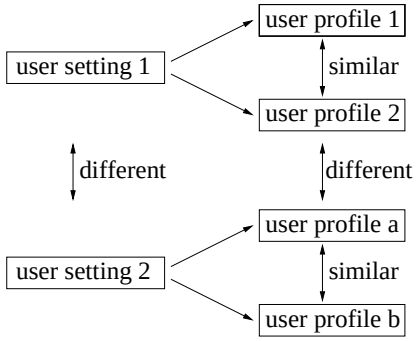


Figure 4.2: The same intention (user settings) of different users (often) results in similar (types of) user profiles. Experiments have been presented, where two different user settings result in two sets of user profiles with different characteristics (text).

the same user setting are likely to show a searching behavior that has similar characteristics. A different setting is likely to result in a set of user profiles that have different characteristics than the profiles in the first set (in a statistical sense).

The described examples in [51] and [18] thus lead to the assumption, that e.g. a user's 'degree of knowledge' about the data objects can be estimated just by observing how he uses the hyperlink structure. A similar assumption is shown in figure 4.3. If two users have the same or a similar user setting,

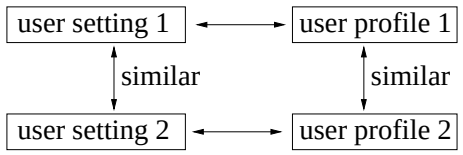


Figure 4.3: Assumed correlation between a user's user setting and the registered profile.

the observed user behavior is likely to have similar characteristics. On the

other hand, user profiles with similar characteristics are likely to result from similar user settings. It is obvious that such conclusions are only meaningful, when the knowledge about a user's behavior is extensive enough. If the observed behavior of users is used to achieve a segmentation in the 'set of user settings', the quality of this segmentation increases with the knowledge about the behavior.
From a theoretic point of view it is difficult to work in the space of user settings. From a practical point of view it is also difficult to acquire information about user settings.[2] Therefore, in the following only the 'space' of the observed user behavior is considered. Main problems will be to find metrics and learning algorithms in this space. A segmentation quality in the space of user settings is only accessed with respect to a prediction quality. It is assumed that an intention is estimated correctly if the observed behavior of a user can be predicted. By this means the 'space of intentions' is not actually needed in the practical considerations, it is only an abstract construction to understand the underlying processes.
One question to consider is why the observed user behavior of different users having the same user setting (or the observed user behavior of one user at different times i.e. sessions) may be different. The results in [18] and [51] only consider common characteristics in the set of behavior observations of experts and novices but they don't consider differences in the observations belonging to one of the groups themselves.
One possibility for such a difference in the users' behavior is (in the case of search in an Web area) the starting point, i.e. the first object that is loaded by a client coming from the considered Web area. This first object may depend on how the user reaches the Web area. One possibility is that he uses a hyperlink from an external place pointing into the considered Web area. In this case, the first object may depend on the data objects the user requested before, i.e. his previous positions in web space. A second possibility is that a search engine is used. In this case, the starting point may depend on the search engine and the query string. Other possibilities for such differences are the applied search strategy of a user, his individual perception of the data objects etc. If two user sessions are observed at different times they may differ because of a change of the Web area in the meantime. Hyperlinks may have been added or missing, the data objects like web pages may have been changed, added or deleted.
Most of these factors that cause a deviation of observed user behavior are not known to a software system like the one presented in chapter 3 and can only be modeled in a statistical sense. The deviation in the observed user

[2]There exist methods like questionnaires, that are inconvenient for a user.

behavior will be refered to as *distortion* of user profiles in the following.

4.3 Formal introduction into graph spaces

The definition of a user session (definition 5) doesn't take into account, how and what kind of actions are registered by the software system. The presented system registers navigation decisions of users as described in section 3.4.2 that can be assigned to specific users by means of the method described in 3.4.2 and 3.5.1. Due to the discussion in section 4.1 and 3.4.2 only the set of actions (navigation decisions) but not the sequence information is considered. Therefore the utilized session information are the sets of clients' transitions between data objects. Such a set can be regarded as a labeled graph, consisting of a set of nodes, that represent the visited data objects, and a set of edges representing the transitions between these data objects. The following definitions concerning such graphs can be found in many books and articles dealing with graph structures, e.g. in [8] and [74].
Let $\mathsf{L_V}$ be a set of vertex labels and $\mathsf{L_E}$ be a set of edge labels and define the set D of all data elements:

Definition 6 *Let 'D' be the finite set of all data elements (or data objects) in the considered Web area with an own URL address.*

Examples for the data elements are web pages, images, sound files, etc. Some of the data objects, e.g. some images, may be part of a web page (and are thus loaded automatically by the browser). These images should not be part of D. The main definition needed to model user behavior is the definition of a graph structure.

Definition 7 *A graph is a 4-Tupel* $\mathsf{G} = (\mathsf{V}, \mathsf{E}, \mu, \nu)$*, where*

V *is a set of nodes and* $\mathsf{E} \subseteq \mathsf{V} \times \mathsf{V}$ *is a set of edges.*

$\mu : \mathsf{V} \to \mathsf{L_V}$ *is a function that assigns labels to the nodes.*

$\nu : \mathsf{E} \to \mathsf{L_E}$ *assigns labels to the edges.*

Unique indices may be assigned to the finite number of elements in D ($\mathtt{Indices}(\mathsf{D})$). The set of labels $\mathsf{L_V}$ of a graph is a subset of these indices, i.e. $\mathsf{L_V} \subset \mathtt{Indices}(\mathsf{D})$. The set of edge labels $\mathsf{L_E}$ may contain additional information about the transition, like the dwell time etc. The edges in this definition are directed. However for simplification reasons, the estimation procedures described in the following only consider undirected edges. For such

an undirected graph it is required that for all $(w_1, w_2) \in E$ it follows that $(w_2, w_1) \in E$, i.e. the adjacency matrix of the graph is symmetric.
Given a finite set D (which is the case in the presented model) the set of possible graphs based on D is also finite.

Definition 8 *Let* Γ *denote the set of all possible graphs with respect to* D. Γ *will also be refered to as the 'graph space'.*

The graph that contains neither nodes nor edges will be denoted as *empty* or *zero* graph:

Definition 9 *Let* $0 = (V_0, E_0, \mu_0, \nu_0)$ *denote the graph 'empty' or 'zero' in* Γ *with* $V_0 = \emptyset$ *(and thus* $E_0 = \emptyset$*).*

In order to measure the 'size' of a graph, the number of nodes is counted:

Definition 10 *Let* $G = (V, E, \mu, \nu) \in \Gamma$ *be a graph. Let* $|G| := |V|$ *be the 'size' of the graph* G.

Obviously there are many different methods to define this 'size' of a graph that also consider the edges. The subsequent definitions are required for the definitions of distance functions between graphs. A *subgraph* is a partial structure of a graph:

Definition 11 *Given a graph* $G = (V, E, \mu, \nu)$*, a subgraph S of G ('*$S \subseteq G$*') is a graph* $S = (V_S, E_S, \mu_S, \nu_S)$ *with*

$V_S \subseteq V$,
$E_S = E \cap (V_S \times V_S)$,
the functions μ_S *and* ν_S *are the restrictions of* μ *and* ν *to* V_S *and* E_S*, respectively:*

$$\mu_S(w) = \begin{cases} \mu(w) & \text{if } w \in V_S \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

$$\nu_S(e) = \begin{cases} \nu(e) & \text{if } e \in E_S \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

The definition of a *maximal common subgraph* of two graphs makes it possible to compare two graphs with respect to a structural similarity:

Definition 12 *Given two graphs* G_1 *and* G_2*. Let* $\mathtt{maxSub}(G_1, G_2)$ *be the maximal common subgraph of* G_1 *and* G_2*, i.e.* $\mathtt{maxSub} \subseteq G_1$ *and* $\mathtt{maxSub} \subseteq G_2$ *and there is no other subgraph S with* $S \subseteq G_1$ *and* $S \subseteq G_2$ *and* $|S| > |\mathtt{maxSub}|$.

In order to decompose a graph into two part, which is an important computation step for various graph matching algorithms [74] and which will be used to measure the prediction quality of the presented algorithm in chapter 6, the definition of a *graph union* or *graph sum* is necessary.

Definition 13 *The union (or sum)* $G = G_1 +_{E'} G_2$ *(where* $G = (V, E, \mu, \nu)$*) of two graphs* $G_1 = (V_1, E_1, \mu_1, \nu_1)$ *and* $G_2 = (V_2, E_2, \mu_2, \nu_2)$ *is defined if* $V_1 \cap V_2 = \emptyset$ *and* $E' \subseteq (V_1 \times V_2) \cup (V_2 \times V_1)$*. Let* ν' *be a labeling function* $\nu' : E' \to L_E$.
Then define:

$$
\begin{aligned}
V &:= V_1 \cup V_2 \\
E &:= E_1 \cup E_2 \cup E' \\
\mu(w) &:= \begin{cases} \mu_1(w) & \text{if } w \in V_1 \\ \mu_2(w) & \text{if } w \in V_2 \end{cases} \\
\nu(e) &:= \begin{cases} \nu_1(e) & \text{if } e \in E_1 \\ \nu_2(e) & \text{if } e \in E_2 \\ \nu'(e) & \text{if } e \in E' \end{cases}
\end{aligned}
$$

In order to decompose a graph G, two graphs G_1 and G_2 and an edge set E' have to be found such that $G = G_1 +_{E'} G_2$.
In order to determine, if two graphs have an identical structure with respect to the node labels and the edges, a *graph isomorphism* between the graphs has to be found.

Definition 14 *A graph isomorphism* $f : V \to V'$ *from a graph* $G = (V, E, \mu, \nu)$ *to a graph* $G' = (V', E', \mu', \nu')$ *is a bijective function with:*

$\mu(w) = \mu'(f(w))$ *for all* $w \in V$

For all $e = (w_1, w_2) \in E$ *there is an edge* $e' = (f(w_1), f(w_2)) \in E'$ *with* $\nu(e) = \nu'(e')$.

For all $e' = (w_1', w_2') \in E'$ *there is an edge* $e = (f^{-1}(w_1'), f(w_2')) \in E$ *with*
$\nu'(e') = \nu(e)$.

With the previous definition the concept of a subgraph can be defined in a way that takes only structural properties of the graphs into account.

Definition 15 *An injective function* $f : V \to V'$ *is a subgraph isomorphism from* G *to* G'*, if there exists a subgraph* $S \subseteq G'$*, so that f is a graph isomorphism from* G *to* S*.*

The preceding definitions will be used to define one class of distance measures between graphs (section 4.4.3). The subsequent definitions are used for another type of distance functions that make it possible to take into account 'distances' between the data elements, i.e. the nodes of the graphs. (The latter distances will subsequently be defined, too.) This second type of distance functions is based on the idea that a graph distance can be defined by transforming one graph into the other, assigning a cost to each operation and by finding a transformation sequence with minimal costs. Possible operations for such a transformation are the exchange of node or edge labels, the deletion or insertion of nodes or edges (a missing node or edge is denoted by the '$' symbol).

Definition 16 *An edit function* δ *of a graph* $G = (V, E, \mu, \nu)$ *is one of the following functions:*

- $\mu(w) \to l, w \in V, l \in L_V$*, here the label value* $\mu(w)$ *of a vertex is substituted by a label* l *in order to correct label errors,*
- $\nu(e) \to l', e \in E, l' \in L_E$*, correction of an edge label by substituting label* $\nu(e)$ *by* l'*,*
- $w \to \$, w \in V$*, deletion of a vertex (and all incident edges),*
- $\$ \to w, w \in V$*, insertion of a vertex,*
- $e \to \$, e \in E$*, deletion of an edge,*
- $\$ \to e = (w_1, w_2), w_1, w_2 \in V$*, insertion of an edge to correct additional edges.*

These edit operations also define 'distortions' as introduced in section 4.2. Different edit functions may have different probabilities. Such a probability (exactly: (1-) the probability) can also be regarded as the 'cost' $C(\delta)$ of an edit operation. The defined edit operations make it possible to transform one graph into any other graph. A graph upon which one edit function was applied is defined as follows:

Definition 17 *For a graph* $G = (V, E, \mu, \nu)$ *and an edit function* δ*, the graph* $\delta(G) = (V_\delta, E_\delta, \mu_\delta, \nu_\delta)$ *is the graph upon which one of the edit functions was*

used.

$$V_\delta = \begin{cases} V - \{w\} & \text{if } \delta = (w \to \$) \\ V \cup \{w\} & \text{if } \delta = (\$ \to w) \\ V & \text{otherwise} \end{cases}$$

$$E_\delta = \begin{cases} E \cup \{e\} & \text{if } \delta = (\$ \to e) \\ E - \{e\} & \text{if } \delta = (e \to \$) \\ E \cap (V_\delta \times V_\delta) & \text{otherwise} \end{cases}$$

$$\mu_\delta(w) = \begin{cases} l & \text{if } \delta = (\mu(w) \to l) \\ \mu(w) & \text{otherwise} \end{cases}$$

$$\nu_\delta(e) = \begin{cases} l' & \text{if } \delta = (\nu(e) \to l') \\ \nu(e) & \text{otherwise} \end{cases}$$

Such a graph will be refered to as *distorted graph* in the folowing. For a transformation of one graph into another usually a sequence of edit operations is required:

Definition 18 *For a sequence of edit functions* $\Delta = (\delta_1, \delta_2, ..\delta_k)$ *and a graph* G *define* $\Delta(G) = \delta_k(...\delta_2(\delta_1(G)))$.

The costs for such a transformation is the sum of the costs of the individual edit functions:

Definition 19 *The cost of an edit sequence* $C(.)$*, is the sum of the costs of the edit operations* $C(\Delta) = \sum_{i=1}^{k} C(\delta_i)$

The concept of a subgraph isomorphism and the concept of an edited graph can be combined into the concept of an *edited subgraph isomorphism.* This definition is a generalization of the previous subgraph isomorphism definitions since it is possible to apply this subgraph condition even if the subgraph isomorphism definition is not 'exactly' fulfilled.

Definition 20 *An error-correcting (ec) subgraph isomorphism f from* G *to* G' *is a tuple* $f = (\Delta, f_\Delta)$*, where* Δ *is a sequence of edit operations, so that there exists a subgraph isomorphism from* $\Delta(G)$ *to* G' *and* f_Δ *is such an isomorphism.*

The costs of such an isomorphism $C(f)$ are the costs of the associated edit sequence $C(\Delta)$.

4.4 Methods to define a distance between graphs

In section 4.2 the assumption was made, that similar observed user profiles are likely to result from similar questions (or user settings) of users. It is still very difficult to find a process to (automatically) estimate a similarity between questions (user settings). Additionally it is difficult for a software system to register questions (user settings) without increasing the expenditure for a user. Therefore the problem of defining a similarity measure in the space of questions is not considered here.
This section gives a survey of definitions of a similarity in the space of observations, which will then be used to estimate the similarity between the underlying question (user settings) in a subsequent chapter according to section 4.2.[3] Since the observations (registered by the described system in chapter 3) are graphs, it is the objective to define a distance between graph structures as defined in section 4.3.

4.4.1 Topological-index methods

A graph can be represented by an adjacency matrix. The manipulation of such matrixes is computationally expensive. In order to develop fast methods for graph comparison it is possible to consider the topological structure and to find so called *topological indices* [5]. Such indexes were e.g. applied in [42] for the comparison of Hypertext structures. A simple example for such a topological index is the sum of the out-degrees (the number of outgoing edges) of all nodes of a graph. It is obvious that there may be many different graphs that have this same topological index.
One main disadvantage of this topological index, which is also true for many other topological indexes, is the degeneracy, i.e. it is not possible to discriminate between large sets of graphs.

4.4.2 Set distance

A different approach is to take a difference measure between data elements represented by nodes (or labels) of two graphs into account. In a simple case such a distance is given by $\delta_{i,j}, i, j \in \mathbb{N}$, i.e. the distance is 0 if the objects i and j are identical and 1 otherwise. A difference between two graphs can then be defined by the number of identical node labels of the graphs. In order to compute a similar distance, it is also possible to define a vector space, where each possible node of a graph is one dimension of this space. The node set

[3] User settings are assumed to be well-estimated (indirectly), if the respective user behavior can be predicted.

of a graph can then be regarded as a vector in this space with a 1 entry, if the possible node is contained in the graph's node set and 0 otherwise. A distance between two graphs can then be defined as the scalar product or the cosine between their respective node vectors. It is obvious that the order of dimensions in the vector space is irrelevant and that these distances don't take account of topological properties (the edge structure) of the graphs.

4.4.3 Shape metrices

The distance functions described in this section combine on the one hand knowledge about topological properties of graphs and on the other hand properties of the node or edge elements. The distance functions are based on the size of the largest common subgraph as given by definition 12. Given two (non-empty) graphs G_1 and G_2 it is shown in [17] that the following distance function has the mathematical properties of a metrics:

$$d_a(.,.) : \Gamma \times \Gamma \longrightarrow \mathbb{R}^{\geq 0}$$
$$(G_1, G_2) \longmapsto \begin{cases} 1 - \frac{|maxSub(G_1,G_2)|}{max(|G_1|,|G_2|)} & G_1 \neq 0 \text{ and } G_2 \neq 0 \\ 0 & \text{else} \end{cases}$$

In [111] a similar distance function between graphs was defined that was also proved to be a metrics.

$$d_b(.,.) : \Gamma \times \Gamma \longrightarrow \mathbb{R}^{\geq 0}$$
$$(G_1, G_2) \longmapsto \begin{cases} 1 - \frac{|maxSub(G_1,G_2)|}{|G_1|+|G_2|-|maxSub(G_1,G_2)|} & G_1 \neq 0 \text{ and } G_2 \neq 0 \\ 0 & \text{else} \end{cases}$$

The reason to consider this metrics is that the size of the graph union $|G_1 \cup G_2| = |G_1| + |G_2| - |maxSub(G_1, G_2)|$, i.e. the denominator in the formula, distinguishes variations in the size of the smaller of the two graphs. *If only the size of the larger graph is used to represent problem size, the distance between graphs will remain unchanged even if the smaller graph changes its size* [111].
In [34] a graph distance was proposed that is based on the maximum common subgraph and the minimum common super-graph (minSup) of two graphs G_1 and G_2:

$$d_c(.,.) : \Gamma \times \Gamma \longrightarrow \mathbb{R}^{\geq 0}$$
$$(G_1, G_2) \longmapsto |minSup(G_1, G_2)| - |maxSub(G_1, G_2)|$$

This metrics doesn't only take into consideration superfluous but also missing structural information in the input graphs.

Algorithms to compute such distances are described in detail in appendix A. These distance functions provide distance measures in a set of graphs without making distance definitions between nodes necessary. Such distances, which are difficult to find in many problems, are required for the following distance functions.

4.4.4 String-edit distance function

The shape metrices described above make it only possible to compare a structural similarity of graphs, if they have common substructures where the node labels are identical. It may also be important to compare navigation profiles that are registered in different web sites where the nodes are different (as was done by [18] manually), which isn't possible using shape metrices. In order to develop such a distance function, it is necessary to take a distance function between nodes (and edges) into account that was denoted as $C(\delta)$ in section 4.3, definition 19, where δ is a substitution of node (or edge) labels according to definition 16. A distance measure can then be defined based on a solution of the well known problem of error-correcting subgraph isomorphism detection which is important e.g. in the field of computer vision [74]. Various approaches to this problem have been presented. One common approach is based on the A^* algorithm [79]. Heuristic methods were presented to reduce the search space [116], [98], [102]. Continuous approaches are polynomially bounded in the number of computation steps but may miss the optimal match in some cases [56], [25].
The problem is to find an error correcting (e.c.) subgraph isomorphism $f = (f, \Delta)$, with $\Delta = \delta_1, ...\delta_n$ that has minimal costs $C(\Delta) = \sum_{i=1,..n} C(\delta_i)$:

Definition 21 *Let* G_1 *and* G_2 *be two graphs, let* $d(G_1, G_2)$ *be the e.c. subgraph-isomorphism distance:*
$d_d(G_1, G_2) :=$
$\mathtt{min}_\Delta\{\, C(\Delta) \mid$ *there exists an e.-c.-subgraph-isomorphism* f_Δ *from* G_1 *to* $G_2\}$

Definition 21 doesn't provide a symmetric distance measure. In order to provide a symmetric distance, it is possible to take the minimum of $d(G_1, G_2)$ and $d(G_2, G_1)$.

Definition 22 *Let* G_1 *and* G_2 *be two graphs, let* $d(.,.)$ *be the e.c. distance measure. Then let* $sd(.,.)$ *be the symmetric e.c. distance measure:*
$sd(G_1, G_2) := \min\{d(G_1, G_2), d(G_2, G_1)\}$

4.4.5 Contents-based distance function

The problem with the previous graph distance is to find adequate node distances (or edit costs). A first approach is to take node and edge distance (substitution costs) that are equal for all nodes and edges respectively. In [16] it was proved that the distances can e.g. be defined in a way that the graph edit distance problem is equal to the maximal common subgraph problem described above (with graph edit costs being equal for all nodes and edges). However in order to achieve an adequate segmentation of the set of graphs it may be necessary to consider more advanced node distances. In our case the actual objects represented by the graph nodes or labels are data objects in the Internet. The definition of distances in such a space of multimedia objects is very complex since it may contain the problem of finding distance measures between text, images, sound files and design features, i.e. how the components are assembled in a web page. In the case of textual data, which are the data that are also considered by Internet search engines, the problem falls into the field of information retrieval. Many distance measures have been developed in the past for textual data [96]. In many procedures the texts are processed into vectors of contained words. The dimension of the underlying space of these vectors is the number of all possible words (that are considered by the system). A distance can then e.g. be defined by the cosine between two word vectors. This means that this vector model reduces a text to the set of contained words and thereby ignores the topology of words in a text.[4]
Another strategy to define edit costs is to learn them from examples, as presented in [90].

4.5 Characterization of a graph distribution

In section 4.2 reasons were provided why the observed navigation behavior of users is likely to be different, even if the intention (setting) of users is the same. The underlying processes causing such distortions like different entry points, a change in the internet environment or different searching strategies can not be observed by our software system. It is therefore necessary to regard the observation of profiles as a stochastic process $(\mathsf{X}_\mathsf{t} : \Omega \longrightarrow \Gamma)_{\mathsf{t}\in\mathbb{R}}$, where X_t, $\mathsf{t} \in \mathbb{R}$ is a set of random variables. A realization of a random variable $\mathsf{X}_\mathsf{t}(\omega) \in \Gamma$, $\omega \in \Omega$ is a navigation graph as introduced in definition 7.

[4] A similar reduction of the problem is possible in the case of navigation profiles as discussed in section 4.4.2 with the described disadvantages.

It is the objective to classify a new profile according to a set of former profiles supplied by users. For this purpose it is helpful to know the distribution of the graph profiles, i.e. the distribution of X_t (it is assumed here that all X_t, $t \in \mathbb{R}$ are equally distributed and that the stochastic process is stationary). In many cases in stochastics, the values of random variables are elements of $\mathbb{R}$ or $\mathbb{R}^n$. Distributions can then often be described by smooth density functions with integral 1, e.g. a Gaussian distribution.
In the presented case, where the values are graphs, other methods to characterize a distribution have to be found.

4.5.1 Discrete characterization

A first method to characterize a distribution of graphs is a discrete characterization. The set Γ of all possible graphs is finite since the set of data objects D is finite. The distribution of the stochastic process can be estimated by the relative frequency of navigation profiles. Methods to describe such a distribution of sets or sequences in the case of very large databases were presented in the field of data mining [1], [24]. In these cases only those data elements with a high frequency are extracted (that represent 'the important part' of the discrete distribution).

4.5.2 Topological characterization

The discrete distribution characterization doesn't take into account a possible similarity between graph profiles. Yet, many navigation profiles will only slightly be distorted because of a similar intention of users as described in section 4.2. A distance measure between graphs as presented in section 4.4 makes it possible to provide a distribution characterization with knowledge about the topology of the graph space. This topological knowledge can be used to simplify the distribution characterization without losing relevant information, which can also be understood as a learning process.
A first idea is to find groups for similar profiles that are likely to result from similar intentions. Provided that there is a finite number of graphs $G_1, ...G_m$, $m \in \mathbb{N}$ and a finite number of n different groups ($n \leq m$), such a distribution characterization can be formalized by a function that assigns every graph profile to its group:

$$\begin{aligned} \mathsf{Charac1}: \quad \{G_1, ...G_m\} &\longrightarrow \{1, ..n\} \\ G &\longmapsto k \in \{1, ...n\} \end{aligned}$$

Another method to characterize such a distribution is to take into account some characteristics about the inner structure of the clusters. Possible char-

acteristics are the center elements of the clusters, the deviations of elements within a cluster and the absolute probability of a cluster, which can be estimated by the relative number of elements (the number of elements in the cluster divided by the number of all elements). A possible formalization of such a distribution characterization is then a set of these cluster characteristics:

$$\mathsf{Charac2} := \bigcup_{i=1,..n} \{(\mu_i, \sigma_i, \frac{A_i}{A})\},$$

where $(\mu_i, \sigma_i, \frac{A_i}{A})$ is the tuple of inner cluster characteristics of cluster i, with μ_i being the center graph of cluster i, σ_i is a measure for the distribution within the cluster, e.g. the mean value of the distances of the elements in the cluster i from the center element μ_i, A_i is the number of elements in the cluster, and A is the number of all graph elements. The center values μ_i can be found from *Charac1* by determining the element in the cluster with the smallest sum of the distances to all the other elements in the same cluster (section 4.6.2).
In the following chapters a simplification of the previous graph distribution characterization is used by taking only the center points into account.

$$\mathsf{Charac3} := \bigcup_{i=1,..n} \{\mu_i\}.$$

The latter distribution characterization can be understood as a characterization by cluster representatives.

4.6 Estimation of graph distributions

The previously defined metrics or distance functions can now be applied to estimate a graph distribution as characterized by one of the methods in section 4.5.2. The navigation graphs can be clustered using common clustering techniques like nearest neighborhood clustering as described in [33] and by using one of the distance functions given in section 4.4. Further investigations concerning the shape of the inner cluster distributions according to *Charac2* in section 4.5.2 can then be made. The steps that have to be performed for a distribution estimation are shown in figure 4.4.

4.6.1 Clustering techniques

The applied clustering technique, i.e. the third step in figure 4.4, is the *single linkage* or *nearest neighborhood* technique, first described in [104], [41]

Graph distribution estimation

1. registration of profiles (section 3.4.2)
2. computation of the distance matrix (section 4.4)
3. cluster algorithm (section 4.6.1, appendix B)
4. estimation of the graph distribution (section 4.5)

Figure 4.4: Steps for a distribution estimation.

and [54]. It is described in more detail in appendix B. The rough idea of this algorithm is first to define a distance between groups as the distance of the closest pair of individuals (one individual in each group). Then the elements are sequentially combined to groups until one group, containing all elements, is obtained. In each combination step two elements (either the original elements or newly formed clusters) are chosen that have a minimal distance. These elements are then combined to form a new cluster element. The sequence of join operations can be visualized as a *dendrogram* [33]
One main problem still present is obviously to obtain the number of clusters that 'adequately' describes a problem. In the case of a distribution estimation it is the problem to find a (minimal) number of clusters that makes e.g. an optimal prediction possible (chapter 6). The number of clusters should in this case be minimal, because the estimated distribution will subsequently be used for a classification of new graph profiles, which is a computationally expensive step, linearly depending on the number of cluster centers (appendix A). A number of clusters, that is too small, may not optimally represent the distribution (which is shown in the experiments in section 6.4.3). A number that is too large may not increase the prediction quality.

4.6.2 Determination of a cluster center

In section 4.5.2 (`Charac3`) it is necessary to determine the center of clusters. In the case of an Euclidean vector space there are different methods known from statistics to find such a center point. Possible methods are e.g. to take the mean value or the expectancy value of the data points in a cluster. In the case of a graph space, the determination e.g. of a mean value is difficult. One

reason is the absence of necessary algebraic operations. In this work therefore another strategy is applied, that is, similar to the cluster procedure, based on the distance values between the graphs. One possibility is to take the (one) value as the center value that has minimal distance to all the other values. If a set of graphs $G_1, ...G_n$ ($n \in \mathbb{N}$) is given and a distance function $d : \Gamma \times \Gamma \longrightarrow \mathbb{R}^{\geq 0}$, define the center element $\bar{G}$:

$$\bar{G} := G \in \{G_1, ...G_n\} \text{ which satisfies: } \sum_{i=1,..n} d(G, G_i) \text{ is minimal.} \quad (4.1)$$

A second method is to reduce the 'costs' for smaller distances, by taking the square of the distance values. This is the value that will be applied in the subsequent experiments. With the same conditions as above, the mean value $\bar{G}$ is in this case:

$$\bar{G} := G \in \{G_1, ...G_n\} \text{ which satisfies: } \sum_{i=1,..n} d(G, G_i)^2 \text{ is minimal.} \quad (4.2)$$

It is obvious that the topology of a graph space is much more complex than e.g. in the case of an Euclidean vector space. It still has to be thoroughly examined under which conditions the above methods lead to meaningful results (which is not considered in this work).

4.6.3 Quality of a distribution estimation

In order to measure the quality of a distribution estimation, it may be helpful to determine the distance between a real distribution that is known in advance and an estimation of this distribution.
Let $G_1, ..G_n$ be the (observed) elements in Γ, $H_1, ..H_m$ ($m \leq n$) be the real cluster centers characterizing the graph distribution and $d(.,.)$ be the distance between two graphs according to one of the definitions in section 4.4. Let $\delta(G) := min\{d(G, H_j)|j = 1, ..m\}$ with $G \in \{G_1, ...G_n\}$.

Definition 23 *Given an estimation of the cluster centers* $\hat{H}_1, ..\hat{H}_m$*, let* $err := \sum_{i=1,..m} \delta(\hat{H}_i)$.

Obviously, *err* decreases, if the estimation result gets better, i.e. the estimated cluster centers move towards the real ones. This quality function is used to evaluate experiments concerning a distribution estimation in chapter 5.

4.7 Summary: Pattern Recognition in Graph Spaces

The system described in the previous chapter makes it possible to register and to store the sets of navigation steps of users on a central server. In this chapter a mathematical model was presented to deal with these data. The registered user behavior is modeled by graphs; the nodes represent requested data objects and the edges represent transitions between data objects (usually the activated hyperlinks). The problem of defining a distribution of a set of graphs was discussed. Instead of a discrete distribution characterization, topological information concerning the distance between graphs is taken into account, too. Different distance measures between graphs were presented for this purpose. With the help of these considerations, a procedure was presented to estimate the distribution of graphs, which is based on a clustering algorithm.

In the next chapter experiments are presented that show properties of the distribution estimation algorithm. In chapter 6 an algorithm for the prediction of user profiles is presented that is mainly based in the distribution estimation.

Chapter 5

Estimation of Graph Distributions

5.1 Task description

There are two main questions in order to evaluate an estimation of a graph distribution. The first question is whether the distribution characterization is adequate for a problem. In section 4.5.2 different possibilities for such a characterization were presented, where the actual characterization also depends on the applied distance function or metrics. It is obvious that e.g. the set distance in section 4.4.2 can't take topological information[1] into account and wouldn't be adequate for a problem where this knowledge is important. The main problem to examine is how to optimally discriminate between the graphs [42]. The second problem is how to evaluate a distribution estimation, given that a distribution characterization was fixed. In the following emphasis is laid on the second question, where in most cases the characterization `Charac3` in section 4.5.2 will be applied and the metrics '$\mathtt{d}_{\mathtt{a}}$' in section 4.4.3. In order to evaluate the procedure for a distribution estimation presented in section 4.6 it is the objective to show properties of the procedure under certain circumstances. The first property to be shown is a convergence of the estimation results to the real distribution for an increasing number of graph profiles. This is a basic statistical requirement for every estimation process. Further analysis concerns the dependence of a distribution estimation on the variance of graphs in clusters, the effect of topological information, and an analysis of 'non-stationary distributions', i.e. distributions that change in time.
These results are difficult if not impossible to obtain from observed navigation

[1] i.e. the information contained in the edges

profiles since the original distribution is not known in this case. Therefore in the following graph profiles will be used, that are generated randomly according to a well-known distribution that was fixed in advance. It is then possible to evaluate the estimation results by comparing them to the original distribution parameters. These experiments reveal main properties of the described estimation procedure.

5.2 Simulation procedure

In order to generate graph profiles of a known distribution, it is first necessary to define a space of graphs Γ, as introduced in definition 8, that is based on a set D of data elements. Instead of taking a set of data objects, we assign every data object an unambiguous number and regard D as a set of these numbers; i.e. $D = \{1, ..n\}$, $n \in \mathbb{N}$. A graph can now be generated by taking a number of elements from D randomly and by fixing a number of random edges between these nodes. It is also possible first to choose a number of edges at random and then to take the set of adjacent nodes as the node set of the graph. With regard to the case considered here, i.e. navigation in an Web area, only connected graphs will be considered. Provided that the size of the graphs, the number of edges, the nodes and the edges were chosen randomly, equally distributed, by this means connected graphs following an equal distribution in a graph space can be created. If the user profiles were equally distributed, nothing could (would have to) be learned by the system. Obviously, in a real scenario the user profiles are not equally distributed as discussed in section 4.2.
In order to generate data that are not equally distributed two assumptions concerning this distribution are made:

- There exist a finite number of local maxima in the distribution (i.e. graphs with a high likelihood), the 'center graphs'.
- The distribution has got the 'shape' of a sum of Gaussian distributions, graphs near the center graphs have a decreasing likelihood, depending on the distance to a center graph.

The distribution will be modeled by a kind of 'sum of Gaussian distributions' in graph space.[2] First a number $m \in \mathbb{N}$ of center graphs are computed. Then a probability value and a variance is assigned to each center graph. The variance value determines, how the likelihood decreases for more and more

[2]This picture is taken from the case of an Euclidean vector space.

Generation of graphs

Distribution generation:

fixing of the data space D
fixing of the distribution settings (number of clusters, the probability of the clusters, the variance)
generation of a distribution characterization (random cluster centers)

Profile generation:

set counter=0, fix number of profiles $n \in \mathbb{N}$,
set graphlist = $\emptyset$
while counter < n

choose a center graph with respect to the probabilities
choose a distortion value with regard to the variance value
distortion of (a copy of) the center graph
store the new graph in graphlist
counter := counter + 1

output graphlist

Figure 5.1: Generation of a number of n simulated graphs.

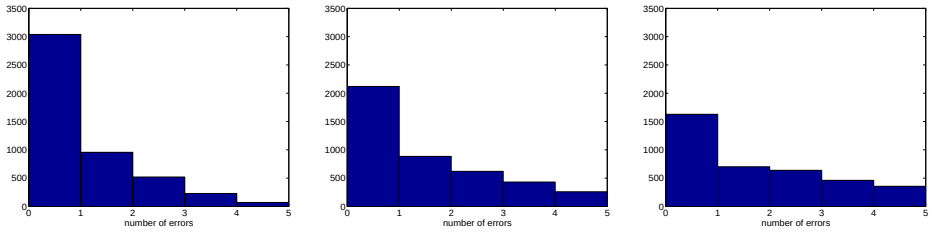
distant graphs.

The different computation steps can be seen in figure 5.1. The upper part of the figure shows the steps to fix a distribution and the lower part of the figure comprises the steps to generate a number of graphs following this distribution.

At first a center graph is chosen according to the probability values of the clusters. Then a distortion value (i.e. a number of distortions) is chosen depending on the variance of the respective clusters. This value follows a discrete (positive) Gaussian distribution:

$$p(x) = \frac{1}{\sigma^2} exp(-\frac{x^2}{\sigma^2}) \ , \ x \in \mathbb{N}^{\geq 0}. \tag{5.1}$$

Figure 5.2 shows histograms of randomly generated values following this discrete distribution for different variances σ^2. These variances will be refered to as *error variances* or *cluster variances* in the subsequent text. A copy of the center graph is distorted with respect to one of the edit functions in definition 16, the number of distortions is the distortion value. The new graph



(a) σ^2=2 (b) σ^2=3 (c) σ^2=4

Figure 5.2: Histograms showing simulated discrete (positive) Gauss distributions with zero mean and different variances σ^2.

obtained in this way is stored and the computation continues by choosing a new center graph in the first step. The graph computation is carried out n times, depending on the number of graphs to be computed.
Similar to figure 4.4 in section 4.6, figure 5.3 shows the steps necessary for a distribution estimation with randomly generated data. In the first step

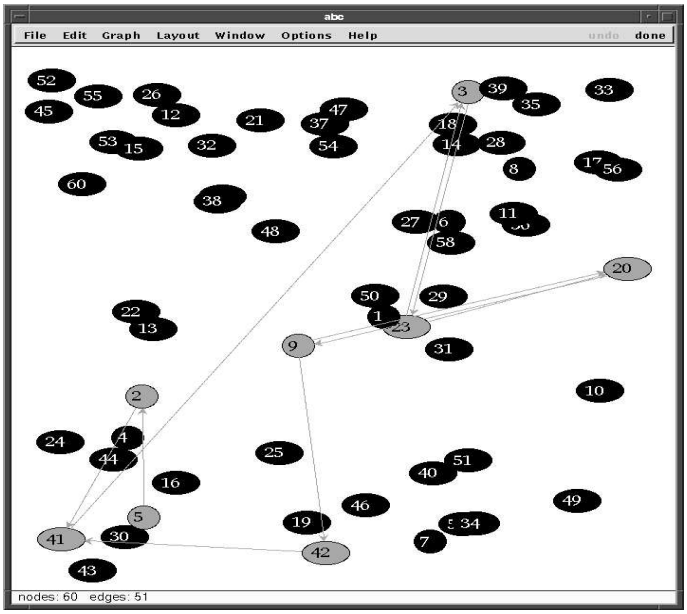
Graph distribution estimation

1. generation of profiles (algorithm in figure 5.1)
2. computation of the distance matrix (section 4.4)
3. cluster algorithm (section 4.6.1)
4. estimation of the graph distribution (section 4.5)
5. quality estimation by comparing estimated graph distribution with original distribution (section 4.6.3)

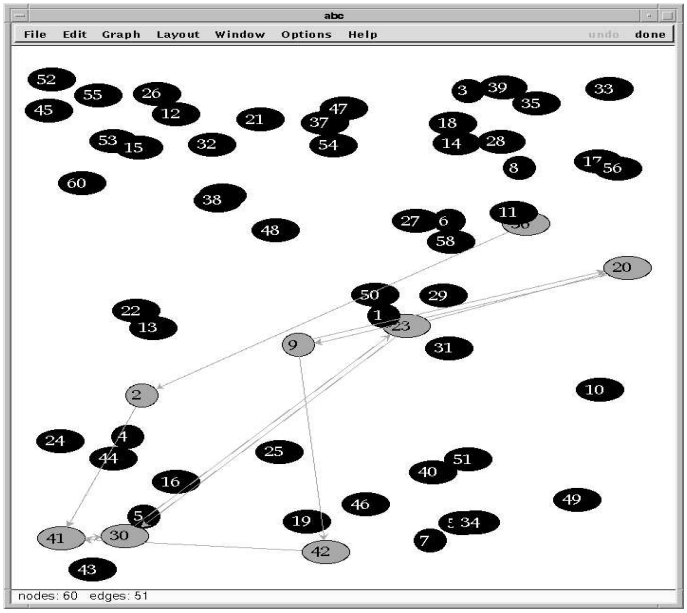
Figure 5.3: Graph distribution estimation with simulated graph data of a known distribution.

a number of graphs $G_1, ...G_m$, $m \in \mathbb{N}$ is generated, following a well-known distribution as described above. In a second step the distances between the generated graphs are computed as described in section 4.4. The result is a matrix M, containing the distance information between element i and j at the position $M(i, j)$. The distance matrix is then used for a clustering proce-

dure of the elements as described in section 4.6.1. The result of the clustering process is a function from the set of graph profiles $G_1, ...G_m$ to the set $\{1, ...c\}$, where $c \in \mathbb{N}$ is the number of clusters. The function assigns each element a cluster number according to the estimation. The cluster information is now used to estimate the distribution as described in section 4.6 with respect to the chosen characterization (section 4.5.2). The main advantage when using generated graphs to test the estimation procedure is the knowledge about the real distribution. In a final step therefore, the estimation results can be compared to the original distribution, i.e. the simulation parameters used for the generation of the graphs. Here, the quality function presented in section 4.6.3 is applied.
Figure 5.4 tries to visualize the distribution estimation problem. Figure 5.4(a) shows a set D of 60 nodes and a possible graph in the graph space Γ, based on D. This graph of 8 nodes is a possible cluster center. Figure 5.4(b) shows a distortion of this cluster center based on label distortions. One node (3) is e.g. deleted and another node (30) is inserted. Figure 5.5(a) shows the sum of five distorted graphs, which is the visualization of a possible cluster based on the center graph (although the cluster is the set of the five graphs, which is difficult to visualize). Figure 5.5(b) shows another cluster based on a second center graph. The problems of clustering and distribution estimation are based on the problem of discriminating between sets of graphs shown e.g. in figure 5.5(a) and 5.5(b) (the different cluster graphs are however merged in the figures).
Another possibility to show the problem of distribution estimation is presented in figure 5.6. Figure 5.6 shows three simulation examples. In each simulation 100 graphs were generated with a distribution consisting of two clusters. The variance of the clusters was changed in the different experiments by taking different variances for the discrete Gaussian function in the distortion step ('error variance'), as described above. Each figure shows a histogram of distances from one (randomly selected) profile to the other profiles. In the first figure the error variance is very small. The two clusters can easily be seen (and separated). About one half of the profiles is similar to the chosen profile, the other half is different. In the second figure a higher error variance was chosen and the two clusters seem to merge, but can still be separated, i.e. the percentage of misclassifications would still be small. In the third figure the error variance is high and the two clusters can no longer be distinguished.
When trying to create clusters it is thus evident that the number of distortions must not be too large, which is obviously also true for real profiles.

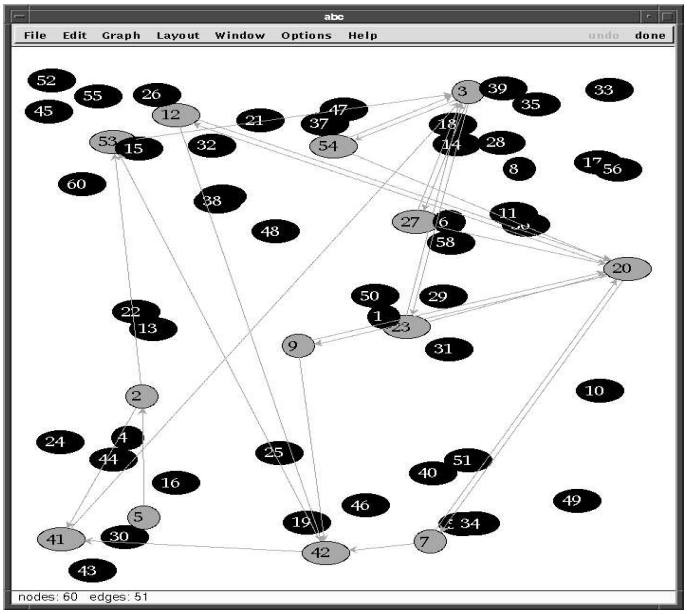


(a) A space (D) of data objects with 60 elements and one graph G of 8 nodes (a possible cluster center).

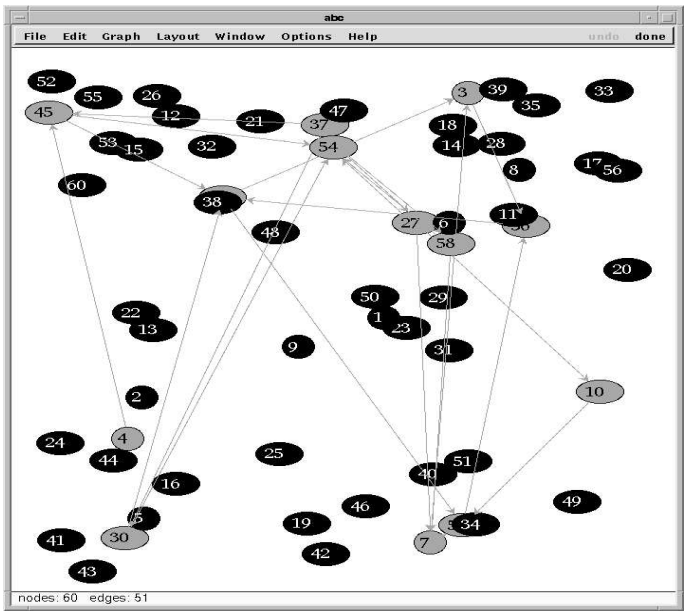


(b) A distortion of the graph G (label distortions).

Figure 5.4: A visualization of a set of data objects, a cluster center in the associated graph space and a distortion of the center element.

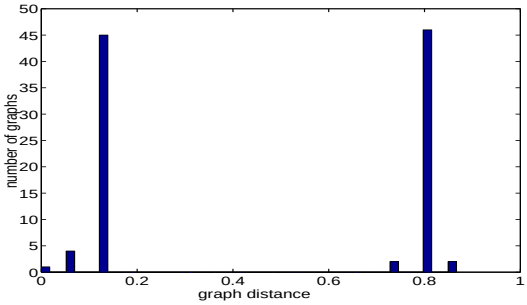


(a) The sum $\sum_{i=1,\ldots 5} G_i$ of 5 distorted graphs (based on G).

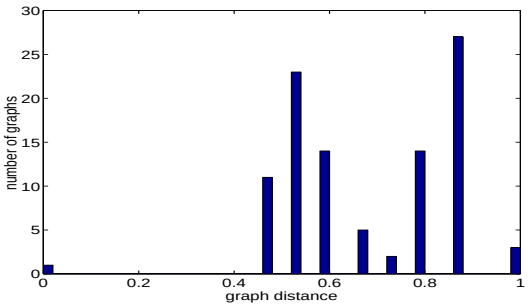


(b) The sum $\sum_{i=1,\ldots 5} G'_i$ of 5 distorted graphs (based on another center graph G').

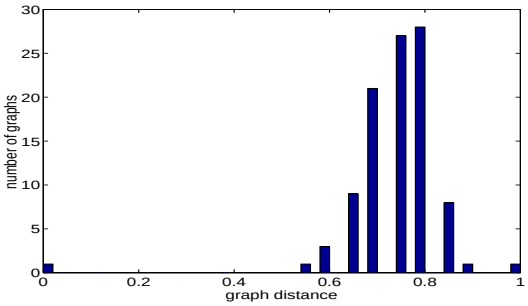
Figure 5.5: A visualization of the sum of five cluster elements based on two different cluster centers. The figures show the problems when visualizing graph clusters directly (for a precise visualization, the five elements should not be merged; they should be separated e.g. by a different color).



(a) In the simulation the error variance is 1 (label errors).



(b) error variance = 5



(c) error variance = 10

Figure 5.6: Histograms of distances from one (randomly selected) profile to a set of 100 profiles. The three histograms are based on three different profile sets, generated with different simulation parameters (i.e. error variances). Common to all simulations, the graphs consisted of 15 nodes, 20 edges and the distribution characterization consisted of 2 clusters. The clusters may only be found, if the cluster variance is sufficiently small.

5.3 Effect of the information used for segmentation

The input data that are used for a distribution estimation contain information about the distribution, depending e.g. on the number of profiles, the degree of the distortion of profiles and the contained topological information. The subsequent experiments try to show some statistical properties of the distribution estimation depending on the input data.

5.3.1 Dependence of distribution estimation on the number of profiles

A main aspect of a distribution estimation is the behavior of the estimation result for an increasing number of data. A theoretical basis for such a consideration in the field of statistics is the 'law of great numbers', i.e. the convergence of the relative frequency of the values of independent random variables to their probabilities, which means that the more observations (measurements) are available, the better is the estimation ('better' is meant in a statistical sense). A similar result is desirable in every estimation process. The question to examine here is if this property is also true for the described stochastic experiment with graph data.
In the experiment, the number of elements in the data set D is 30, the number of nodes in each graph is 25, with 30 edges. The original distribution characterization (`Charac3` in section 4.5.2) consists of 2 original (center) graphs of the same size, that are found randomly with equal distribution in the graph space. The number of identical simulations is 10; that means that the same simulation setting was used 10 times to generate random data and to compute the estimation error. Then the mean value was computed based on the 10 values.
Figure 5.7 shows the estimation error according to section 4.6.3 depending on the number of profiles used for the distribution estimation. Each value in the figure is the mean value of the estimation errors in the identical simulations. The graph metrics applied here for the clustering and the estimation quality measurement is the first subgraph metrics in section 4.4.3.
As can be seen, the estimation error decreases, when the number of graphs increases. This means that, the more graph observations are available, the better the distribution estimation is.

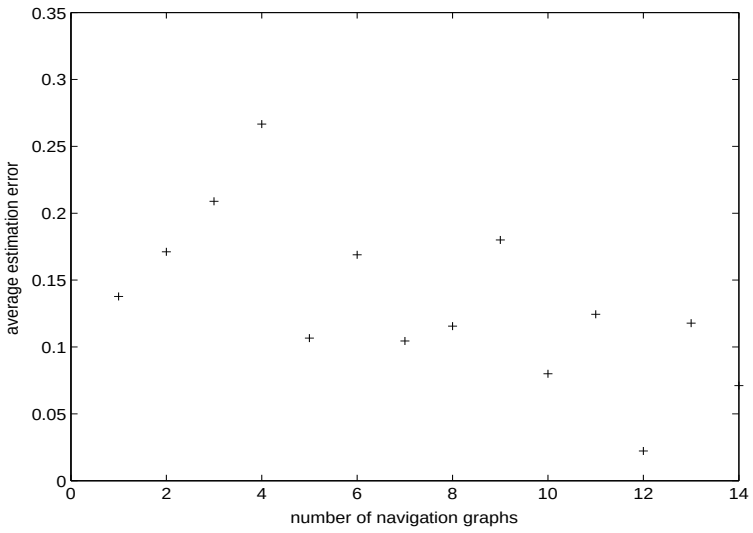


Figure 5.7: Dependence of the estimation error according to section 4.6.3 on the number of profiles. The original distribution consists of 2 centers.The values are the mean values of 10 identical simulations (text). The estimation error decreases, if more (graph) observations are available since the information about the distribution increases.

These results were expected with regard to the theoretical considerations above. The more information about the distribution is available for the estimation process, the better are the estimation results. Similar results will be obtained using a different quality measure in the next chapter.

5.3.2 Impact of distortions

In section 4.2 the term *user setting* was introduced to combine different main parameters that influence a user's searching behavior. Reasons were discussed, why navigation profiles may differ though users have the same or a similar user setting. These differences were denoted as *distortions*.
In this experiment the consequence of distortions on the quality of the estimation result is examined.

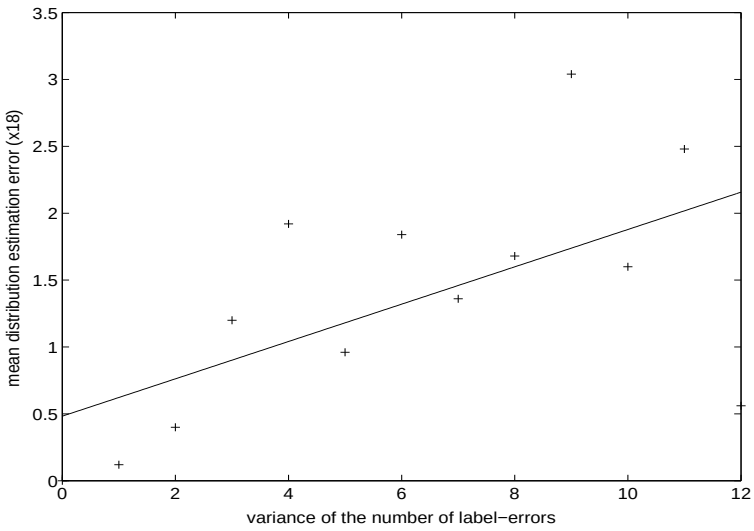


Figure 5.8: Dependence of the estimation error on the error variance of graph clusters with a linear (least square) approximation. An increasing variance of clusters leads to a loss of information about the graph distribution.

The simulation parameters are basically the same as in the previous experiment. The number of clusters is 2 according to `Charac3` in section 4.5.2. Every generated graph contains 25 nodes and 30 edges. The number of profiles is in all simulations constantly set to 10, the error variance of the profiles is changed from 0 to 12. In the experiment label distortions are considered (section 4.3).
Figure 5.8 shows the result of this experiment. The distribution estimation error is increasing for an increasing error variance of the graph profiles.
The reasons for these results can be seen in figure 5.6. The larger the error variance of profiles, the more difficult it is to discriminate between the clusters. An increasing error variance (distortion) results in a loss of information about the distribution. Similar results have been found for another quality measure in the next chapter.

5.3.3 Effect of topological information

One aspect that was important in section 4.4, when different distance measures between graphs were presented, is the importance of topological information, i.e. information contained in the edges of a graph. This question is examined for the metrics $\mathtt{d}_\mathtt{a}$ in section 4.4.3 (it has already been discussed that this information has no effect e.g. for the set metrics in section 4.4.2).

In the experiment the basic distribution contains two clusters, represented

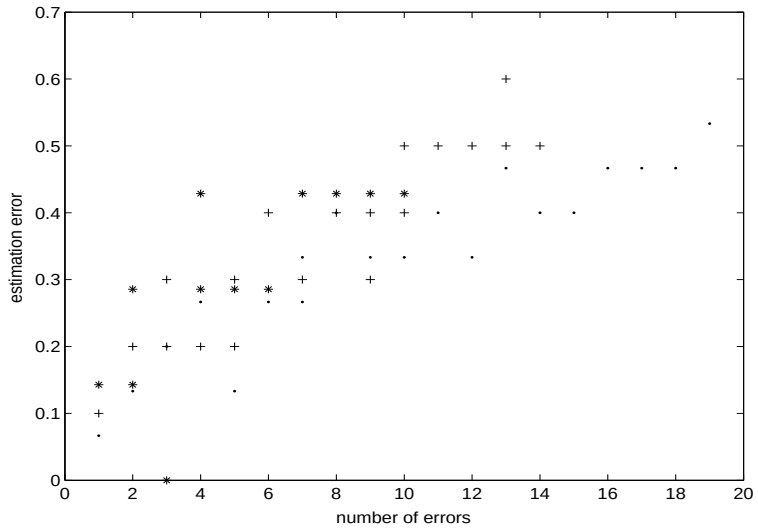


Figure 5.9: Dependence of the estimation error on topological information: edges are gradually deleted. In each experiment there is a different size of the generated graphs in Γ: '.' denotes the results for a graph size $|V_G|$=15; '+': $|V_G|$=9; '*': $|V_G|$=5. The smaller the graphs, the faster is the increase of the estimation error, when topological information is gradually deleted.

by two center graphs that are generated randomly in the graph space. The number of profiles is 5 in all simulations. The number of nodes is the same as the number of data objects in D, ($|D|$). The number of edges is (at the beginning, without distortions) $|D| + 5$. In the experiment the distortion function is, different from the experiments above, a deletion of edges. Edges are subsequently deleted and the effect on the estimation quality is measured. Figure 5.9 shows the results for 3 different graph sizes. The '*' symbol marks the simulation results for a size of all graphs $|V_G| = 5$. In the experiment the number of edge distortions (deletions) is increased from 0 to 10. The '+' symbol marks the simulation results for $|V_G| = 9$, the number of edge distortions (deletions) is increased from 0 to 15 and the symbol '.' marks the simulation results for $|V_G| = 15$, the number of edge distortions (deletions) is increased from 0 to 20.
The results show how the distribution estimation error increases, if the topological information is gradually deleted (similar to the previous experiment, where label distortions were considered). This increase is higher if the graph size is small as can be seen by the increase for different $|V_G|$ values. In the

estimation process the topological information thus seems to be more important for small graphs.

5.4 Non-stationary distributions

In section 4.5 the concept of a characterization of the distribution of graph profiles was introduced. In real-word scenarios these distributions will not remain constant, but they will change in time. With regard to the user setting in section 4.2 it is e.g. possible that the searching experience of Internet users increases, since more and more people get used to applying the Internet for information search. The distribution of information needs may also change from time to time depending on certain trends in science, economy etc. The knowledge about a distribution, that may actually be used in real applications is limited. In this thesis, distributions are characterized decisively by center graphs (section 4.5). In the next chapter it will be shown, that for prediction purposes, it is one main computation step to compare a new registered graph to these center graphs (classification, section 6.2). This comparison is computationally expensive and therefore the number of graphs used for a distribution characterization has to be limited as far as possible (though the number of graphs should on the other hand be maximal with regard to the result in a previous section. The information about the graph distribution increases, when the number of graph profiles for the estimation process increases).
In this experiment it is the aim to examine the effect of a distribution change on the quality of the distribution estimation. The maximal number of graphs that may be stored by the system is limited to 20 (in real cases this number will be larger). These 20 graphs are the profiles that are used for the distribution estimation. If a new graph is registered, the oldest graph in the list is deleted and the new graph is inserted (first-in-first-out storage). In the experiment a change in the distribution is assumed to occur instantly at a time '0'.[3] After each insertion of a graph (after the reference time '0') that follows the new distribution a new distribution estimation is performed. In the computation of the estimation error (section 4.6.3), the estimation result is compared to the old distribution (before the change), which is assumed to be available.
Figure 5.10 shows the estimation results, where both (old and new) distributions consist of two clusters (`Charac3` in section 4.5.2), the number of data elements is $|\mathsf{D}| = 20$, the number nodes in each graph is 10; there are

[3]There are different ways conceivable how a distribution may change. In real cases the change is likely to occur in a time period ('smooth' distribution change).

15 edges in each graph and a fix number of node label distortions of 5. The x-axis in the figure shows the number of graph exchanges (deletion of old graph, insertion of a graph, that follows the new distribution).

It can be seen that the estimation error increases and reaches a maximum

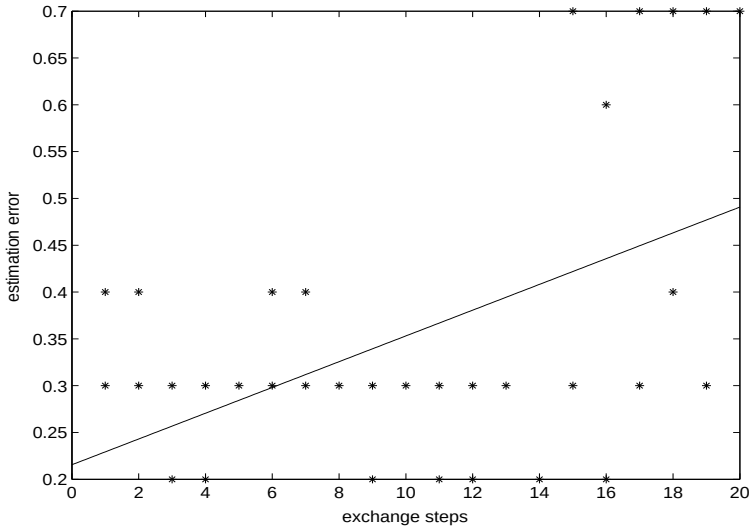


Figure 5.10: A distribution change occurs at the first registration step (exchange step '0'). Both distributions consist of 2 clusters. The estimation errors for both clusters can be seen (in contrast to definition 23, where the values are added). The estimation error increases while the elements in the graph profile set used for the distribution estimation are substituted by elements following a new distribution (the figure shows a linear least square approximation to the data points).

when all graph profiles are exchanged by profiles following the new distribution.

A similar analysis for real profiles may be necessary to find out, how often a distribution estimation has to be performed. On the one hand, this estimation is computationally expensive and should be performed as rarely as possible. On the other hand the learned distribution will no more represent the real distribution after some time due to a change in the distribution. Therefore a distribution estimation should be performed as often as possible. Similar considerations may help to deal with this complex problem for real data.

5.5 Summary: Estimation of Graph Distributions

In the previous chapter 4 a procedure to estimate distributions of graphs was presented. In this chapter it was the objective to evaluate such a distribution estimation. The basic idea is to fix a graph distribution (characterization), to generate graphs that follow this distribution, to estimate the graph distribution using the algorithm in the previous chapter and finally to compare the original graph distribution and the estimation results (according to definition 4.6.3).
Applying this procedure different properties of the graph distribution process have been shown. One result is that the distribution estimation improves when the number of graphs available for the estimation process increases. It has been shown how the estimation error depends on the number of label errors, and how the size of graphs takes influence on the importance of topological information (the information contained in edges) for the estimation process. Finally non-stationary distributions have been considered. The result of a change in a graph distribution has shortly been discussed.
These results show that basic statistical properties seem to be fulfilled for this estimation process, there are however many open questions, especially if changing distributions are concerned.
The described testing method can't be applied to real data since in this case the original distribution is not known in advance (as it was in this chapter). Therefore in the next chapter another method to test the distribution estimation method is presented that can also be applied to observed graph profiles.

Chapter 6

Classification and Prediction of User Profiles

It is the aim in this chapter to apply the theoretic model and the distribution estimation presented in previous chapters to predict future navigation decisions of a specific user. The prediction is based on the (navigation-)behavior of this user and a set of registered profiles of other users. This prediction can also be regarded as a relevance estimation of future navigation steps for the user, provided that the relevance of navigation decisions for users is strongly connected to the probability that users actually make these navigation decisions. This assumption is probably valid, if the users of the web site know the site already very well (and thus always make optimal decisions according to relevance). In real cases, the knowledge of users is insufficient and the equivalence of prediction and relevance estimation is only partially true.
The presented algorithm first uses registered or generated profiles to estimate the distribution of profiles. A new profile is then classified according to this distribution. The classification result can then be used for a prediction. In the chapter a quality measure for the prediction process will be presented that, in contrast to the estimation error measure in section 4.6.3, doesn't make it necessary to actually know the original distribution.

6.1 Prediction methods

The estimation of relevant data objects or navigation decisions is well-known from the field of search engine development [10]. Search engines usually apply methods developed in the field of information retrieval [96], [95]. The estimations are usually based on query words provided by a user. In this thesis, a relevance estimation is not based on search words but on navigation

behavior. A possible estimation method based on the observation of navigation behavior that applies information retrieval methods as described above, would be to find e.g. web pages that are similar to the ones, a user already requested [68], [69]. As far as the mere observation of users, without a contents analysis is concerned, methods have been developed in the field of data mining to find sets or sequences of objects with a high frequency [1], [24]. These data mining techniques can be understood as a kind of distribution estimation, where the 'positions' in the discrete distribution with the highest likelihood (frequency) are found. The data mining techniques can be used to predict future navigation steps, applying a similarity measure between a new observed profile and the observed profiles with a high frequency. However this estimation doesn't take similarities of profiles and thus topological information into account and is therefore susceptible to distortions. The estimation therefore leads to problems discussed in section 4.5.2.
A problem, similar to the previous prediction technique, was presented in [83]. The article presents a procedure for the generation of so-called 'index lists', i.e. web pages consisting of hyperlinks to a set of pages that cover a particular topic. The procedure is based on access-log data, that are transformed into sets of data objects, which were requested by the individual clients. The procedure first computes the co-occurence frequencies of data objects; that means the frequencies that two data objects are found together in one of the requested sets of clients. The respective matrix is then used for a cluster procedure, which results in clusters of data objects. After a first selection by a web-master, the clusters are used to generate the index lists. If a user requests a data object contained in one of the clusters, the respective index list may contain links to web pages that are also relevant for him (because they often deal with a similar topic).
Recently, Markov models were applied for a prediction of user profiles [2], [119], [99]. One main disadvantage of these models is that the order of the Markov process has to be fixed in advance. In practical cases the order is usually chosen to be very low (i.e. one or two, e.g. in [2]) because the computations would be too complex otherwise. On the other hand, a prediction is usually only done for the next element, i.e. the subsequent element with the highest (conditional) probability. Further predictions of subsequent elements or navigation steps are difficult to obtain. New method have recently been developed by [92] and [15] in order to make Markov models of variable length possible (*Variable Length Markov Chains*). These methods have not yet been applied in this specific problem domain. The VLMCs make it necessary too, to define a maximal depth of the respective Markov chains.
The prediction method presented here ([61]) doesn't make it necessary to fix the order of a stochastic process, since the profiles are completely compared

to each other with the described distance functions. It is possible in some cases to predict longer navigation sequences, if this knowledge is contained in the database. In contrast to common data mining procedures, the distances of profiles are taken into account, which makes it possible to take distortions into account and therefore to find a more adequate distribution estimation.

6.2 A new prediction algorithm

A survey of the sequence of computation steps of the new algorithm can be seen in figure 6.1. In a first (off-line) step, the registered profiles are used for a distribution estimation as described in section 4.6. It is important to note that this estimation step can in most applications and especially in our case be done off-line.

Prediction procedure

(offline)

Data acquisition

Distribution estimation

(online)

Registration of a new (partial) user profile

Computation of the distances to the estimated cluster centers

Classification of the new profile according to the estimated distribution and a classification function

Prediction of future navigation decisions according to the classification result

Figure 6.1: Algorithm for a prediction of new navigation decisions.

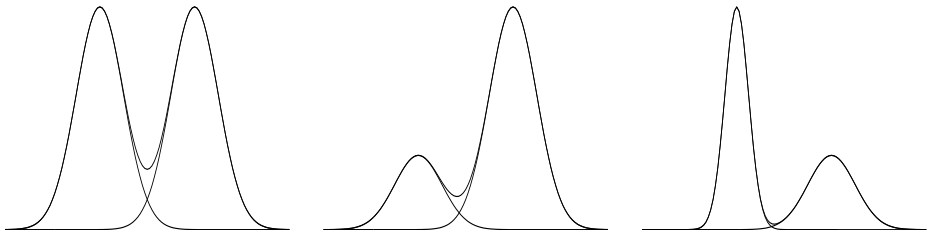
One important question is, how often this estimation should be performed. A problem concerning this question is that a fast change in the distribution can't be proved if there are not enough registered profiles. The observation of a sufficient number of profiles necessary to estimate a change in the distribution takes some time; it depends on how frequented a web site is. Depending on this frequency, the time intervals between distribution

estimations have to be chosen, which may e.g. be a week or even a month. Another possibility to choose the time intervals between distribution estimations is to update a distribution estimation every time, when the web site has changed itself, because the distribution of profiles is then likely to change, too.[1]
The online (prediction) step in figure 6.1 has to be performed in real-time, since it is based on a user's current navigation profile and the estimation results shall be presented to him immediately. As can be seen in figure 6.1, a new currently registered navigation profile is compared to the estimated cluster centers known from the distribution estimation step. According to this distance vector and with respect to the distribution characterization the new profile is classified. The classification may also take into account additional information like (an estimation for) the absolute likelihood of a cluster, the variance etc. This classification step will subsequently be described in detail. According to the classification, the respective cluster elements are now used for a prediction of new navigation steps or sequences. Different prediction techniques are conceivable.

Classification

It is difficult to visualize the classification problem in the space of graphs, therefore figure 6.2 shows the similar problem in the case of a random variable into the space of real numbers, $\mathbb{R}$. The integral over the respective density function should in this case be one. The first figure 6.2(a) shows two clusters with the same probability and the same variance. The cluster centers in this case are the (two) elements on the x-axis where the probability value has a (local) maximum. A classification of a new element in this case seems to be optimal if the closest cluster center is found and the element is assigned to the respective cluster.

[1] This practical problem was not considered here. It was however already made evident in section 5.4.



(a) A distribution consisting of two Gaussian functions.

(b) A distribution consisting of two Gaussian functions with different probability.

(c) A distribution consisting of two Gaussian functions with different variances.

Figure 6.2: Different shapes of distributions (density functions). The figures show the density functions of the individual clusters and the sum of the individual density functions.

In the second figure 6.2(b), the absolute probability (the integral of the respective density function) of the first cluster (on the left) is smaller than the probability of the second cluster (on the right). A classification according to the previous method could cause a misclassification in the area between the two clusters, because the likelihood[2] of the second cluster in this area is obviously larger. In the third figure 6.2(c), the variance of the first cluster is smaller than the variance of the second cluster although the absolute probability of both clusters is similar. A classification of an element in the area between the clusters should therefore prefer the second cluster.
In order to find adequate classification functions in the case of a graph space Γ, it is assumed that similar results prove to be true in the case of a random variable into Γ. Given the estimated cluster centers $\hat{H}_1, ..\hat{H}_m$, $m \in \mathbb{N}$ and the new profile G, in the first method

$$d1(G, \hat{H}_j) := d(G, \hat{H}_j) \tag{6.1}$$

has to be minimized in $j \in \{1, ..m\}$, where $d(G, \hat{H}_j)$ is a distance of G to the cluster center $\hat{H}_j$ as defined in section 4.4. A second method is to take the absolute probability of clusters into account, which can be estimated by the relative number of elements in the cluster. The minimization of

$$d2(G, \hat{H}_j) := d(G, \hat{H}_j)\frac{1}{1 + A_j/A} \tag{6.2}$$

[2]The likelihood in this case is an integral of the density function over an interval in the respective area.

in j takes this absolute probability into account, where A is the number of observed profiles, A_j is the number of profiles estimated to belong to cluster j. If the probability of the cluster j is large (as estimated by the relative frequency A_j/A), the distance $d2(G, \hat{H}_j)$ becomes smaller. Thus the probability is increased, that the graph element G is assigned to the cluster j.
There are many different classification functions conceivable which have not yet been considered.

Prediction

Once, a profile has been classified (assigned) to one of the clusters, the actual predictions can be derived from the elements in this cluster. In the simplest case, the cluster center is taken as a prediction, which is the case that will be applied here. It is also possible to take the center element and the elements near the center, according to a distance $\epsilon \in \mathbb{R}$, that defines a 'sphere' around the center element. These elements in this sphere can then be combined in order to generate a prediction:

$$K(\mu_j, \epsilon) := \sum_{X \in K_{\mu_j,\epsilon}} X \ , where \ K_{\mu_j,\epsilon} := \{X \in C_j | d(X, \mu_j) \leq \epsilon\} \tag{6.3}$$

The sum is based on the graph summation given by definition 13 (with $E' = \emptyset$). C_j is a cluster and μ_j is the cluster center; d(.,.) is a metrics according to section 4.4.3. In the following only the case $\epsilon = 0$ is considered.[3]

6.3 A measure for the prediction quality

The prediction results depend on the distribution estimation, the classification process and the prediction process. In section 4.6.3 a method was presented to estimate the quality of a distribution estimation. The method requires the knowledge of the original distribution, which is known only for simulated data. As far as real data are concerned, another strategy to measure the prediction quality has to be applied. The strategy to measure the prediction quality that is presented here is similar to a method presented in [118] (the data type in the article (i.e. sets) is different from the case considered here). The structure of this process can be seen in figure 6.3. A set of profiles is divided into two parts, a training set and a testing set. These profiles may be generated or extracted from observations.

[3] In terms of information retrieval [96] a large ϵ is likely to increase the recall, while a small ϵ will increase the precision.

Estimation of the prediction quality

Acquire profiles:
Generation of profiles (simulation) or
profile extraction (observations)

Splitting:
Decompose the profile set into training and testing set

Distribution estimation:
Use training set for a distribution estimation

Testing:
fix a decomposition size s, set recall=0
for each graph G in the testing set:
decompose $\mathsf{G} = \mathsf{H}_\mathsf{G} + \mathsf{T}_\mathsf{G}$ with regard to s, i.e. $|\mathsf{E}_{\mathsf{H}_\mathsf{G}}| = \mathsf{s}$
(random decomposition)
classify H_G
compute prediction $\mathsf{P}_{\mathsf{H}_\mathsf{G}}$
compute the distance $\mathsf{d}(\mathsf{P}_{\mathsf{H}_\mathsf{G}}, \mathsf{T}_\mathsf{G})$ between $\mathsf{P}_{\mathsf{H}_\mathsf{G}}$ and T_G
recall := recall + $1 - \mathsf{d}(\mathsf{P}_{\mathsf{H}_\mathsf{G}}, \mathsf{T}_\mathsf{G})$
Output $\frac{\text{recall}}{|\mathtt{testprofiles}|}$

Figure 6.3: Estimation of the prediction quality for simulated and real profiles based on a registered (or generated) set of profiles. $|\mathtt{testprofiles}|$ is the number of test-profiles.

The training set is used to estimate the distribution of the profiles as described in section 4.6. Each profile G in the testing set is used for one prediction process.
The graph is first divided into two parts, i.e. $\mathsf{G} = \mathsf{H} + \mathsf{T}$ where $\mathsf{H} = (\mathsf{V}_\mathsf{H}, \mathsf{E}_\mathsf{H}, \mu_\mathsf{H}, \nu_\mathsf{H})$ and $\mathsf{T} = (\mathsf{V}_\mathsf{T}, \mathsf{E}_\mathsf{T}, \mu_\mathsf{T}, \nu_\mathsf{T})$ are subgraphs of G and '+' defines the graph that results from the combination of the node and edge sets of the two graphs (definition 13).[4] The division of G is done randomly ac-

[4]The set of edges E' between the two subgraphs in definition 13 is not considered here.

cording to a previously fixed number of edges in H.[5] H is the graph used as the (partial) navigation profile that has to be analyzed. A prediction for this graph is computed according to section 6.2. The predicted navigation profile can then be compared to the part of the profile that actually occurred, i.e. the graph T. The distance between these two graphs according to a distance function defined in section 4.4 is then used as a quality measure for the prediction. The quality values of each graph in the testing set are summed up and taken as a value for the prediction quality. One quality value derived by this method will be called *recall* in the following:

$$\mathrm{recall} := \frac{1}{|\mathrm{testprofiles}|} \sum_{i=1,..|\mathrm{testprofiles}|} 1 - d(P_i, T_i), \tag{6.4}$$

where |testprofiles| is the number of profiles, $G_i, i \in \{1, ..|\mathrm{testprofiles}|\}$ is the testing set and T_i and P_i are the respective partial and predicted profile. The recall is a measure of how similar a set of predicted navigation profiles is to the profiles that were actually registered.
The term *recall* is frequently used in the field of information retrieval. It denotes e.g. the percentage of data elements, a search engine retrieves, that are relevant for a user, divided by the number of all relevant elements. In the described case it is not known, if the registered navigation path was really relevant for a user. As discussed at the beginning of chapter 6 this can only be assumed in the ideal case, that users are already familiar with the web site. The set of all relevant navigation profiles is not known and thus a normalization similar to the described recall definition in information retrieval is difficult.
This measure strongly depends on the registered data elements and must therefore be understood in a statistical sense. It is necessary to examine different data files in order to eliminate statistical deviations.

6.4 Experiments with simulated data

6.4.1 Different classification functions

In this experiment the effect of two different classification functions on the classification quality is examined, provided that the distribution is already known. The graph distribution (`Charac3` in section 4.5.2) contained 2 graph clusters C_1 and C_2 with probability $p(C_1) = \frac{1}{3}$ and $p(C_2) = \frac{2}{3}$. Each graph contained 25 nodes and 30 edges. The random graphs are generated

[5] The decomposition should usually be done in a way that both subgraphs are connected.

as described in section 5.2. For each generated graph, its original cluster was known. Thus the percentage of misclassifications could be determined, denoted as 'classification error'.
Fig. 6.4 shows the classification error based on the minimization of $\mathsf{d1}$ (•) and $\mathsf{d2}$ (+) in section 6.2 ($0.x \hat{=} (10 \times x)\%$ in the figure). In the experiment the variance of label errors σ^2 is changed (section 5.2). Each x-axis value represents one set of generated graph profiles.

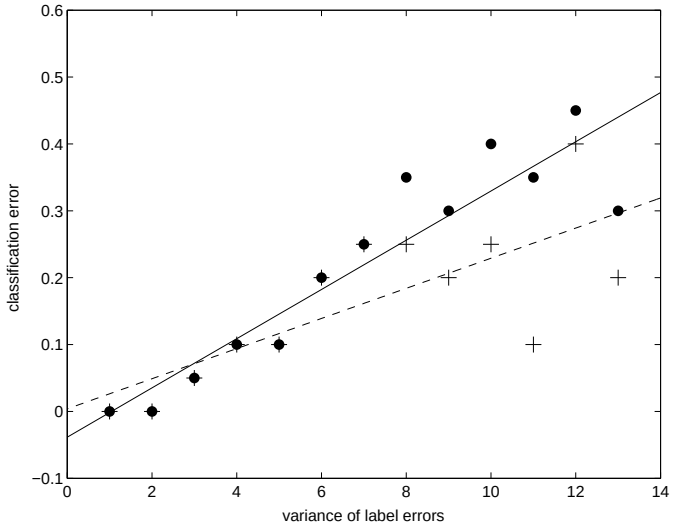


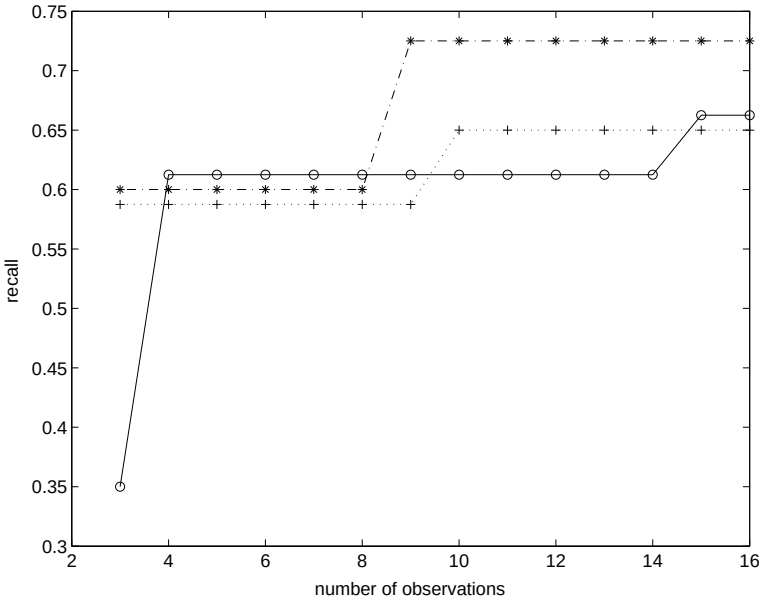
Figure 6.4: Classification experiment of user profiles by minimizing d1 (•) and d2 (+) in section 6.2. The distribution consists of two clusters with different probabilities. The two lines are linear (least square) approximations to the respective data points. The classification result improves, if the probabilities of the clusters are taken into account in the classification process.

As can be seen, the prediction based on minimization of $\mathsf{d2}$ shows better results for higher values of the label error. This result was expected, since more information about the shape of the distribution is used in the case of $\mathsf{d2}$. The experiment proves, that the applied classification function may have an impact on the classification error. Nevertheless in the following experiments the nearest-center classification (d1) is applied in order to find different properties of the estimation process.

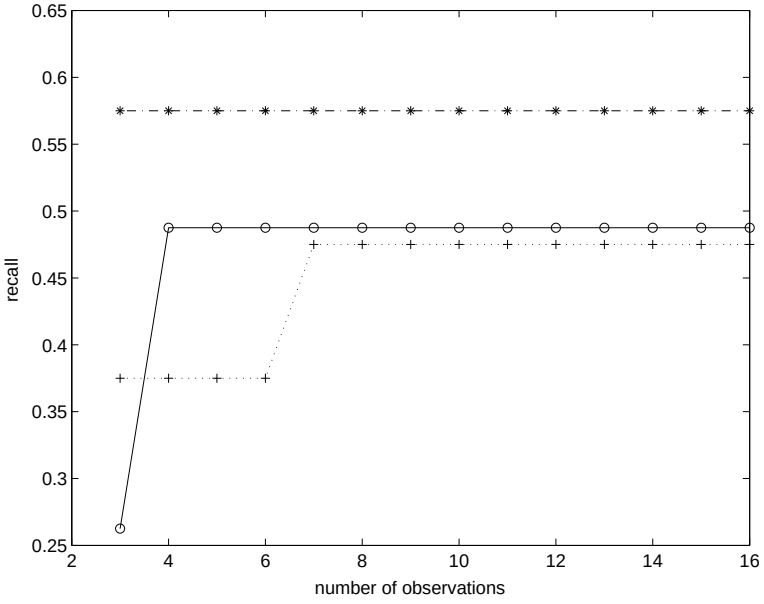
6.4.2 Convergence of the estimation

Dependence of the prediction quality on the number of graph profiles

An important aspect when analyzing the quality of a prediction process is the convergence of the prediction quality (as measured e.g. by equation 6.4) for an increasing number of observations (in this case generated graph profiles), similar to the results and the discussion in section 5.3.1. It is desirable that, the more observations are known in an estimation process, the better the estimation for the respective distribution of the data becomes. In this experiment this aspect was analyzed for different cluster variances (section 5.2).
The number of graph profiles used for the distribution estimation is changed in a range from 2 to 16. The set of profiles is 'growing' in this experiment, i.e. a larger set contains the smaller sets (in former experiments the sets were generated independently). The number of elements in D, is $|\mathsf{D}| = 50$. Each graph has 8 nodes and 13 edges. The number of edges in H is 5 (H is the first subgraph in the decomposition step in section 6.3). The distribution consists of two clusters and thus two center graphs according to `Charac3` in section 4.5.2. The error variance σ^2 according to equation 5.1 is changed in a range from 1 to 4. The applied error function is a random exchange of node labels (equally distributed in D).
Figures 6.5 and 6.6 show the results of the four simulation experiments. It can be seen that the prediction quality (recall) is increasing monotonously for small error variances (1 and 2). For larger error variances (σ^2=4), there is no obvious convergence, the recall value seems to change arbitrarily. When looking at the absolute values on the y-axis it can be seen in this experiment, that for smaller variances the recall is higher and becomes smaller when the variances increases.

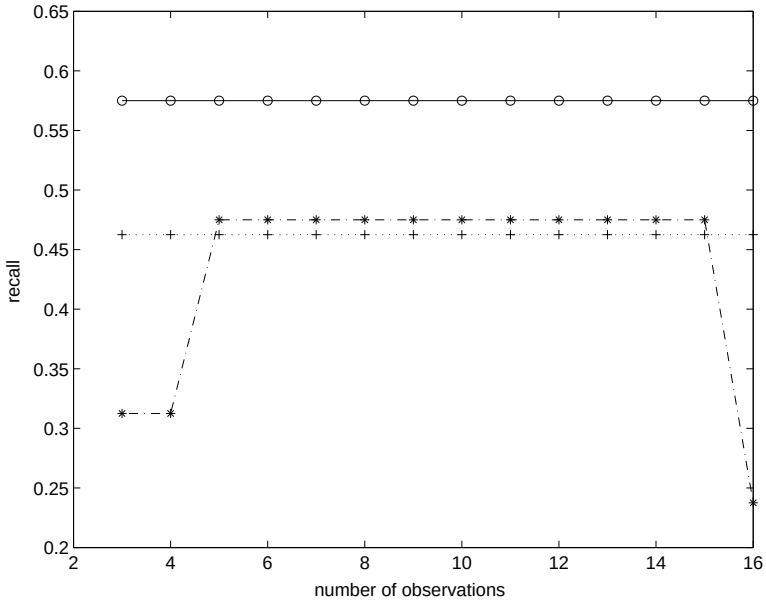


(a) variance $\sigma^2 = 1$

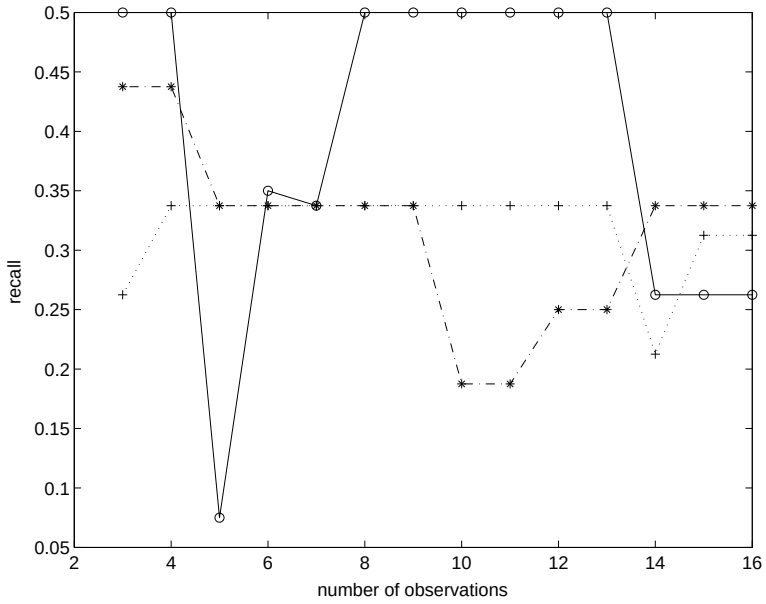


(b) variance $\sigma^2 = 2$

Figure 6.5: Prediction quality depending on an increasing number of graph profiles known in the estimation process. The variance of the clusters is changed (i.e. the error variance σ^2 in section 5.2). Each figure shows 3 simulations with identical simulation parameters. The prediction quality improves monotonously when the number of graph profiles is increased.



(a) variance $\sigma^2 = 3$



(b) variance $\sigma^2 = 4$

Figure 6.6: Similar graphs as in the previous figure. The error variances are $\sigma^2 = 3$ and $\sigma^2 = 4$. If the cluster variances are too high, there is no convergence.

The convergence of the prediction quality (recall), that was observed in this experiment for small variances, is the result that was actually expected with respect to the result in section 5.3.1. An increase of information in the estimation process (the number of observed graph profiles) leads to an increase of the prediction quality. The fact that there is no convergence for larger variances may result from the lack of information needed for the estimation process (similar to the discussed problem in section 5.2, figure 5.6). This effect may be important when applying the algorithm to real observations. It is only possible to apply the algorithm when the variance of observed clusters is small.

Dependence of the prediction quality on the graph decomposition needed for the prediction quality measurement

A main step when computing the prediction quality in section 6.3 is the decomposition of a testing graph G in two subgraphs $H = (V_H, E_H, \mu_H, \nu_H)$ and $T = (V_T, E_T, \mu_T, \nu_T)$. In this experiment, the dependence of the prediction quality on the number of edges in H ($|E_H|$) is examined.
The experimental setting is similar to the previous experiment. The original distribution consists of two clusters, the size of the graph space is $|D| = 30$, the generated graphs consist of 8 nodes and 13 edges. The error variance of the clusters in this experiment is $\sigma^2 = 1$. In the experiment different decompositions of the testing graphs were tested. The experiments measure the prediction quality for an increasing number of profiles, where the number of edges in the subgraph $H = (V_H, E_H, \mu_H, \nu_H)$ is changed in different experiments.
Figure 6.7 shows the results of these experiments for an edge number $|E_H| = 0$, $|E_H| = 1$, $|E_H| = 2$ and $|E_H| = 4$. The number of profiles is changes from 3 to 16.
In the figure the recall values in each graph are the averages over 9 identical experiments (i.e. the simulation parameters in these experiments are identical).
It can be seen that the prediction quality is minimal for $|E_H| = 0$ (lowest graph). The maximal graph is the experiment with $|E_H| = 1$, the prediction quality is then decreasing for $|E_H| = 2$ and $|E_H| = 4$.
The minimal prediction quality for $|E_H| = 0$ is the expected result, since no information about the graph is known in the classification step, the graph is classified randomly.

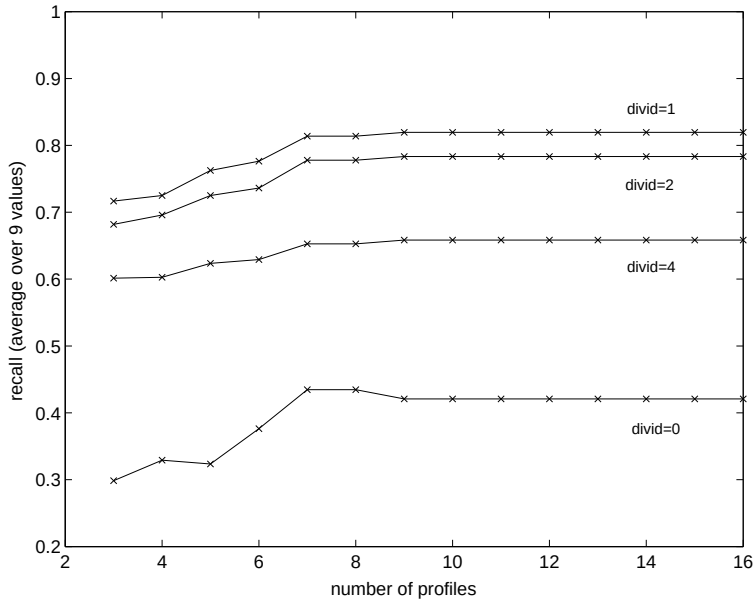


Figure 6.7: Convergence of the prediction quality (recall) for different numbers of edges in H ($divid = |E_H| = 0, 1, 2$ and 4). Each graph is the average over nine simulations. The recall is minimal for $E_H = 0$ and maximal for $E_H = 1$. One edge seems to be sufficient in this experiment to classify a graph (text).

The prediction quality is not constantly zero, probably due to statistical effects. The highest prediction quality for $|E_H| = 1$ leads to the assumption that the information about one edge is sufficient to classify a testing graph correctly, probably because the graphs in different clusters don't have many edges in common. The prediction quality is decreasing for higher values of $|E_H|$ probably because the second part of a testing graph (T) becomes smaller for larger H-graphs and thus the similarity of P and T decreases (as measured by the metrics d_a in section 4.4.3). Similar considerations for higher variances may lead to different results. Then more information may be necessary to correctly classify a graph, since the graphs may have more edges in common.

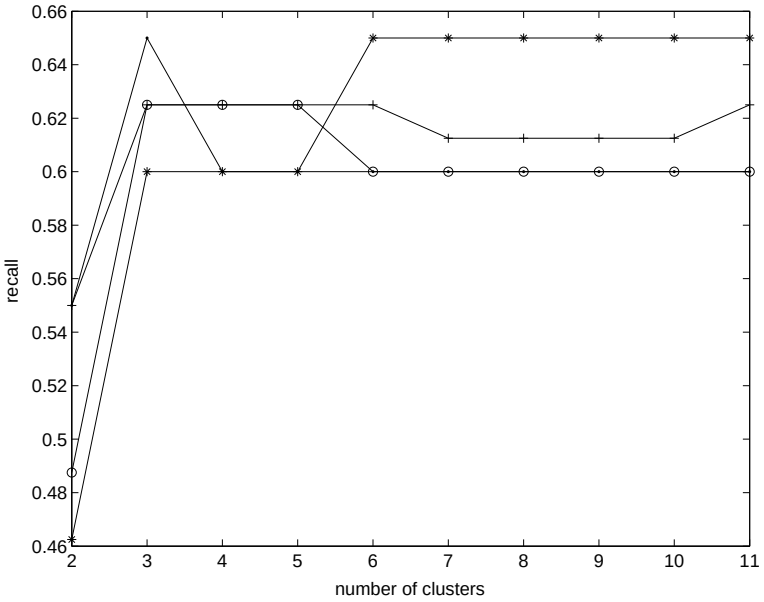
6.4.3 Estimation of the clusters number

When analyzing real data, the number of clusters is not known in advance, in contrast to the previous simulations, where the cluster number was known in the estimation process. In this experiment it is therefore the aim to test with simulated data, if (and how) the algorithm can be applied for the estimation

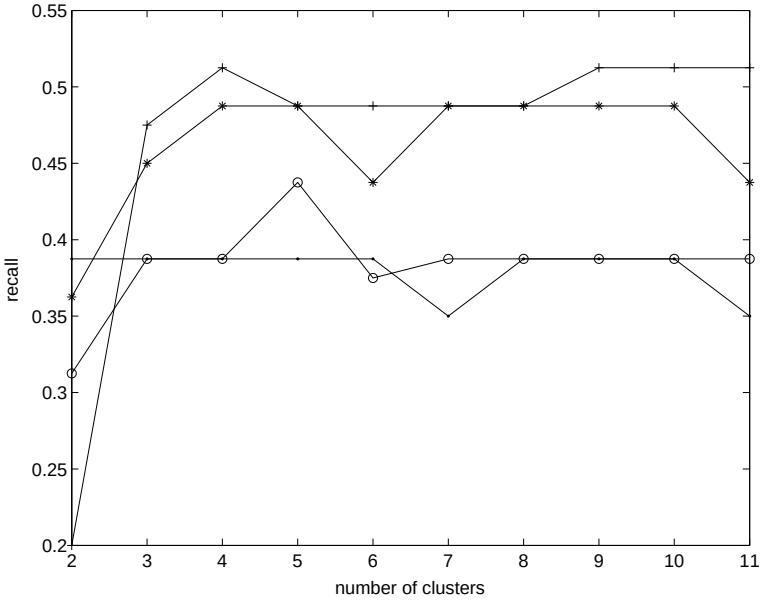
of the number of clusters.
In the experiment the number of elements in D was $|D| = 30$. All graphs contained 8 nodes and 13 edges. The number of edges of the graph H in the decomposition step is $E_H = 4$ (section 6.3). The initial distribution contained a number of center graphs (and cluster) that was 3,4 or 5. This number was not known in the prediction process. Therefore the procedure assumed a certain cluster number and computed the resulting prediction quality in the same way as in the previous experiments. The aim is to find out, if this prediction quality changes for different assumed cluster numbers, which is increased from 2 to 11. The variance of the clusters is a second variable that was modified in the different experiments. In all experiments the same simulation setting was used four times, in order to (visually) eliminate statistical effects.
Figures 6.8, 6.9 and 6.10 show the results of the six experiments (in each figure 4 experiments can be seen, using the same settings for the graph generation). In the first sub-figure in figure 6.8 the original number of clusters is 3, the error variance is $\sigma^2 = 1$; in the second sub-figure the error variance is $\sigma^2 = 3$. In the two sub-figures of figure 6.9, the number of clusters is 4 and the error variances are $\sigma^2 = 1$ and $\sigma^2 = 2$. In figure 6.10 the original number of clusters is 5 and the error variances are $\sigma^2 = 1$ and $\sigma^2 = 2$.
It can first be noticed, that similar to the experiment in figures 6.5 and 6.6, a larger variance results (statistically) in a smaller prediction quality (recall). It can be seen in all experiments that the recall is low when the assumed number of clusters is smaller than the real number of clusters. The recall seems to reach a maximum when the assumed cluster number reaches the real value. For larger values of the assumed cluster number, the recall doesn't change very much and remains at the high value. In some cases, the recall decreases a little at a higher value of the assumed cluster number (figure 6.8b, 6.10b). For smaller error variances the deviation between the respective graphs of the (four) identical simulations is also smaller (e.g. figure 6.9a, 6.9b).
In most cases the original number of clusters is (near) the (x-axis) value of the first local maximum in the respective figures. In figure 6.8a these values are e.g. 3,3,3 and 6 in the identical simulations. In figure 6.8b these values are 4,4,5 and 2. In some of the identical simulations this estimation would be incorrect (6.8b) but in most cases it leads to a value close to the original number of clusters. The estimation of the cluster number seems to become more difficult for a larger variance of the clusters, as can be seen when comparing figure 6.8a and 6.8b.
The results seem to make it easy to present an algorithm for the estimation of the number of clusters. At first the prediction quality has to be computed

based on an observed (or generated) set of profiles and based on an assumed number of clusters. The assumed number of clusters is changed in an adequate range. In the resulting graph (similar to the figures 6.8, 6.9 and 6.10) the first local maximum is found and the respective assumed cluster number is regarded as an estimation of the number of clusters.[6]
The experiments show that this estimation process is in many cases optimal since a value near the real number of clusters is found. The estimation process doesn't seem to depend on the set of profiles very much for small error variances (as can be seen by looking at the different identical simulations, using generated graphs with identical simulation settings). The quality of the estimation depends on the variance of clusters.
In real applications, e.g. the prediction of user profiles, the estimated cluster number based on a set of registered profiles would be used in the prediction algorithm. The different experiments show that thus the prediction quality would be high with respect to the number of clusters while, taking the first local maximum as the estimated real cluster number, the estimated cluster number would be small. This is important since the classification step in section 6.2, which has to be done in real-time for the profile prediction, is time consuming (appendix A).

[6]If this value is not unique, the smallest x-axis value (number of clusters) is taken as the estimated cluster number.

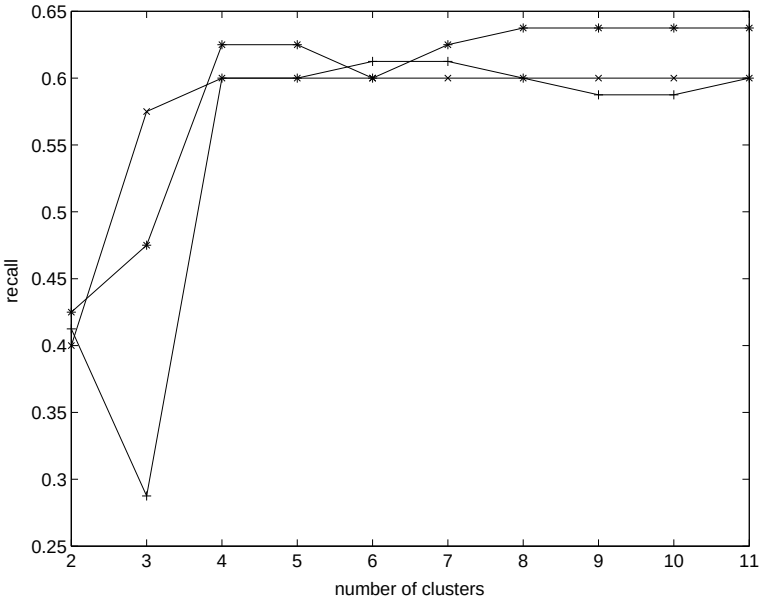


(a) 3 original clusters, error variance $\sigma^2 = 1$

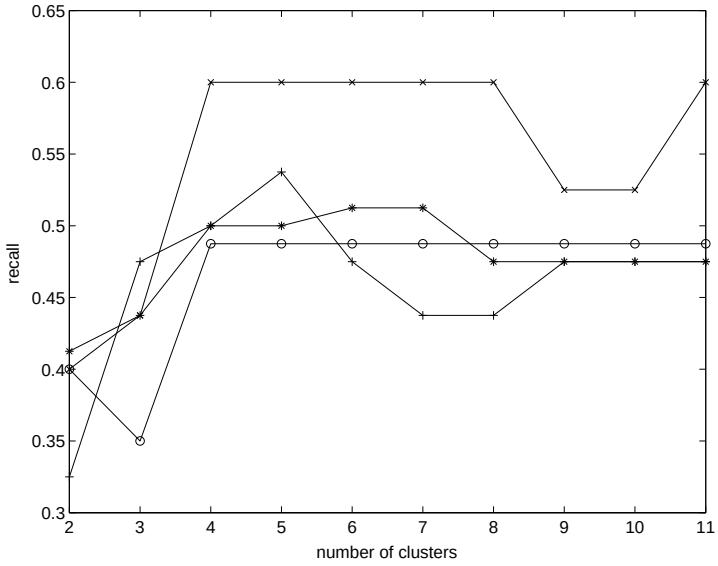


(b) 3 original clusters, error variance $\sigma^2 = 3$

Figure 6.8: Prediction quality (recall) for an original cluster number of 3 and a changing assumed cluster number, there are two different error variance values in both figures (4 identical simulations). The original cluster number is near the x-axis value of the first local maximum (3,3,3 and 6 in figure (a); 4,4,5 and 2 in (b)).

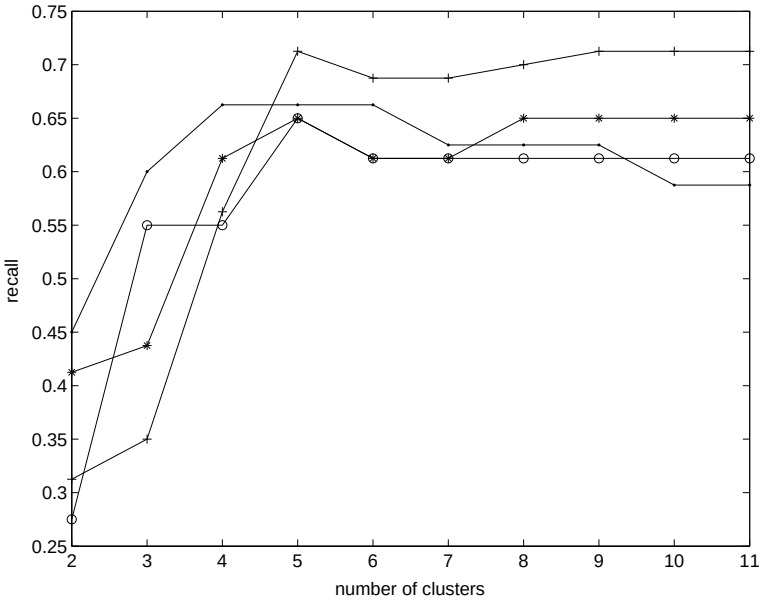


(a) 4 original clusters, error variance $\sigma^2 = 1$

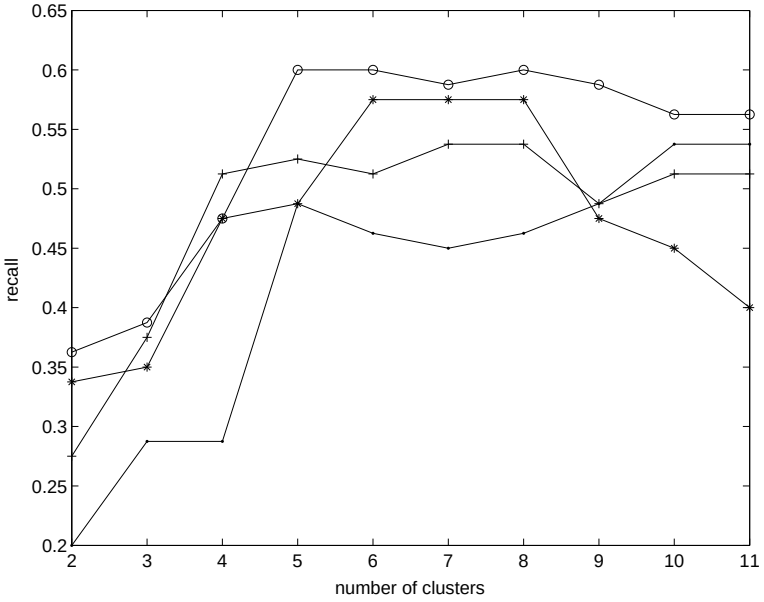


(b) 4 original clusters, error variance $\sigma^2 = 2$

Figure 6.9: Prediction quality (recall) for an original cluster number of 4 and a changing assumed cluster number, different error variance values.



(a) 5 original clusters, error variance $\sigma^2 = 1$



(b) 5 original clusters, error variance $\sigma^2 = 2$

Figure 6.10: Prediction quality (recall) for an original cluster number of 5 and a changing assumed cluster number, different error variance values. The estimated cluster numbers using the described procedure (text) are 4,5,5 and 5 in (a) and 5,5,6,5 in (b).

6.5 Application to registered navigation profiles

The prediction quality measure (recall) in section 6.3 can be applied to real observations (navigation profiles) that have been registered by a software system like the one described in chapter 3. In this section the estimation and prediction process is applied to observations that were registered by clients who visited the Internet server of the Cognitive Systems group at Kiel University in the year 2001 (http:\\www.ks.informatik.uni-kiel.de). The web site consisted at that time of about 3000 data objects[7]. In this experiments the information stored in the access-log files (section 2.4.2) is transformed into the graph data type, which is described in detail in appendix C.[8]
In figure 6.11 the distribution of the size (i.e. the number of nodes) of a subset of the registered profiles can be seen. The maximal size is 45, but most of the profiles have a size smaller than 10. It is important to note that only one server was considered. In the case of a larger, distributed Web environment, this may have an effect on the distribution of the graph size. The result, that most navigation graphs are not very large, is important for the time needed for the graph distance computation (appendix A). Since these algorithms have in the worst case a n.p. complexity, graphs that are too large, would make it hardly possible to use them.
Figure 6.12 shows an example of a histogram of distances of one (arbitrarily selected) profile to a set of other profiles with respect to the metrics $\mathtt{d}_{\mathtt{a}}$ in section 4.4.3. It can be seen that most of the profiles have a maximal distance to the considered profile (the graphs don't have any common nodes). These profiles are obviously located in different cluster(s). In contrast to the simulated data in the example shown in figure 5.6, there is no obvious cluster structure.
In these experiments, it is the aim to apply the procedure for an estimation of the number of clusters presented in section 6.4.3 to subsets of the observed profiles. The number of edges in the graph H in the division step is 2 (section 6.3).
In the first experiment a set of 10 test graphs and 60 training graphs was chosen randomly in a set of 80 profiles (section 6.2). Figure 6.13 shows the observed prediction quality for an increasing (assumed) number of clusters for 3 different testing sets.
Figure 6.14 shows the results of a similar experiment where the size of the

[7] like web pages, images etc. (section 2.1.2)

[8] The direct use of the data provided by the developed system (chapter 3) was not possible because of an insufficient number of observations.

profile set was 200, the set of testing data contained 20 profiles and the training set consisted of 150 data. In figures 6.14(a) and (b) different testing sets were chosen, the training set is identical. In figure 6.15, a similar experiment was performed with a different (disjoint) set of profiles (including different training and testing sets).
In most of the experiments there exists a unique number of clusters, where the prediction quality is optimal (maximal). In two of the experiments in figure 6.13 the optimal number of clusters is about 40 (optimal with respect to the prediction quality). If the number of clusters is further increased, there is hardly any improvement in the prediction quality. In figure 6.14, the optimal number of clusters is about 79 in figure (a) and (b). In figure 6.15 the optimal value is about 25 and 55. In all experiments (except the constant prediction quality in one experiment shown in figure 6.13) the optimal number of clusters can be found easily and nearly unambiguously.
The estimation procedure for the cluster number proposed in the previous section leads to nearly the same values as the cluster number values leading to an optimal prediction quality. The first local maximum is e.g. about 80 or 78 in figure 6.3.
It can be seen in 6.13 that two of the testing sets lead to almost the same result for an estimated number of clusters. One testing shows a constant prediction quality. The estimated number of clusters in figures 6.14(a) and (b) is also nearly the same for both testing sets (about 78 and 80). The deviation in figure 6.15 is larger (25 and 55). The concordance of the estimated number of clusters in both graphs in figure 6.13 and in both figures 6.14 ((a) and (b)) shows that the cluster number estimation is nearly indifferent with respect to the testing set. An exception is the constant graph in figure 6.13 which is likely to result from an unfavorable testing set. Figure 6.15 (based on a different set of profiles) leads to a different estimation of the number of clusters (a value of about 25 or 55) than the similar experiment shown in figure 6.14. Different data files therefore seem to lead to different estimation results. Theoretically there should however exist a number of profiles, where a convergence of the estimation result can be observed as shown in the theoretic simulations in section 6.4.3. The distribution of the profiles in different registered profile sets should be identical and thus the estimated distribution (e.g. the number of clusters). This important property of the estimation couldn't be proved yet for real observations. There are four main reasons why this convergence could not yet have been observed:

- The number of profiles used for the estimation (the size of the training set) is still too small
- The site was visited not frequently enough and there was not enough

information for the estimation process (the 'density' of profiles in graph space has to be high enough).

- The distribution of profiles changed
  - due to a general trend (a changing interest of users)
  - The contents of the site (data objects and included hyperlinks) was changed while the profiles where registered and the distribution of profiles thus changed.

It can be summarized that the described prediction algorithm can be applied to real observations in order to estimate the number of clusters, which is an important property of the described graph distribution characterization (`Charac3` in section 4.5.2). This knowledge about the distribution is necessary for the prediction process as described in section 6.2. The number of clusters may be chosen in a way that the prediction is optimal. The indifference of the estimation result (the number of clusters) with respect to the testing set could be shown in experiments. It was not possible to show an indifference with respect to the training set for real navigation profiles.[9] Possible reasons are an insufficient number of profiles or a different distribution of the profiles.
The optimal testing web site for the prediction procedure is a web site, that is relatively small and static[10] and that is highly frequented. The worst case to test the algorithm is a large web site, that is rarely frequented. It may in future experiments be necessary to test the algorithm at a web site of the first kind in order to prove indifference with respect to the training set.

[9] For the experiments with simulated data in the previous section this indifference could be shown.
[10] i.e. there are not many changes of the data objects and links.

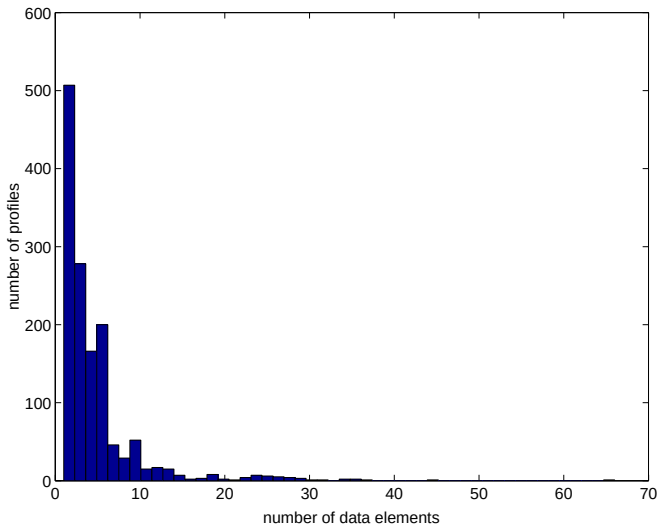


Figure 6.11: Histogram of the length of observed profiles. All profiles have a size smaller than 45. Only a few are larger than 10.

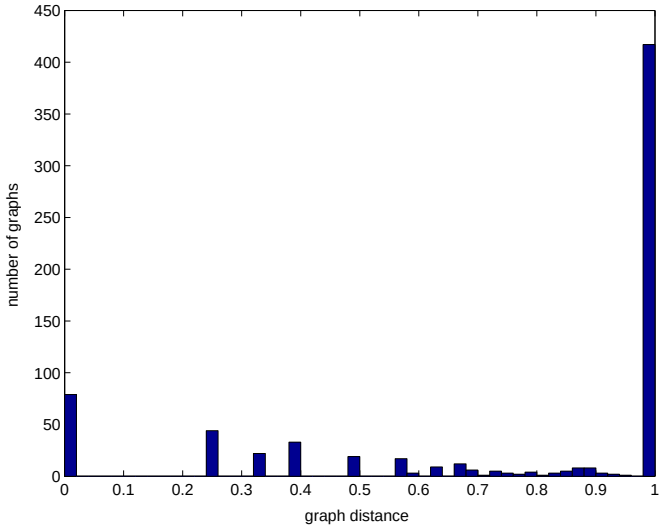


Figure 6.12: Histogram of distances of one (observed) profile to a set of other profiles. Most of the profiles have a maximal distance to the selected profile. There is no obvious cluster structure.

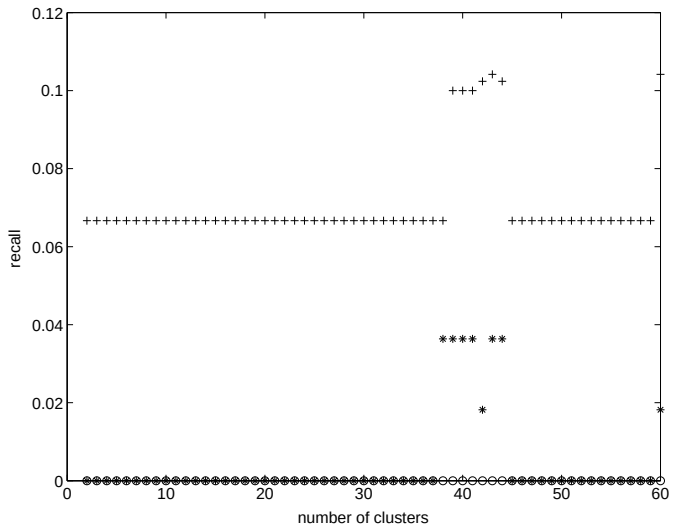
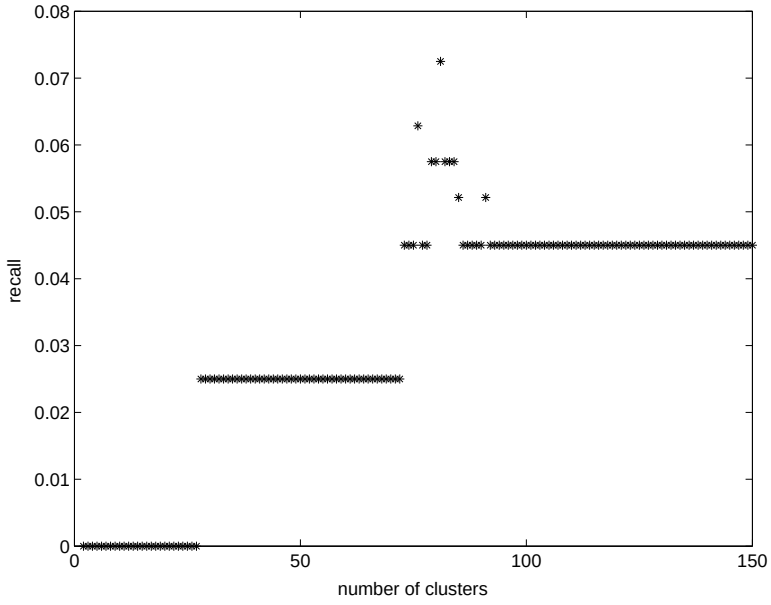
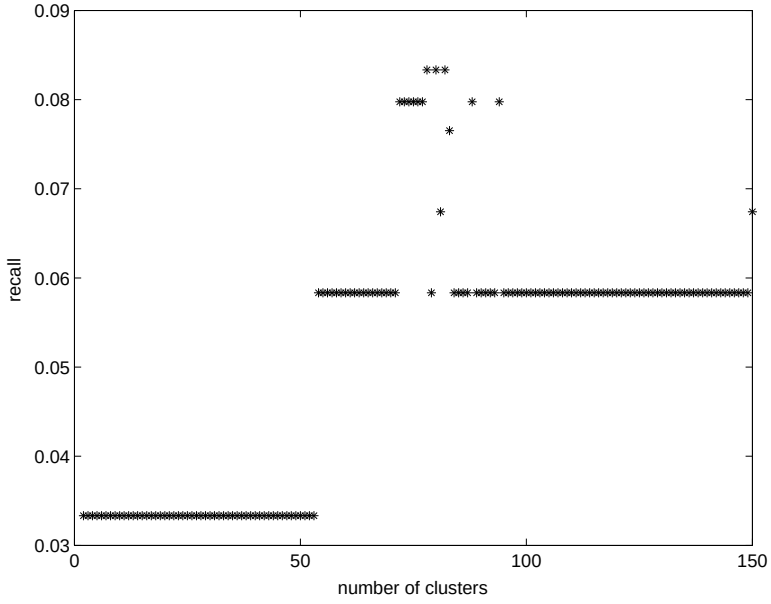


Figure 6.13: Prediction quality (recall) for observed navigation profiles; 10 test, 60 training data (the whole data set contained 80 data). The (assumed) number of clusters is changed in a range from 2 to 60. The different graphs show the prediction quality for different testing sets. Two testing sets lead to a similar estimation of the cluster number (about 38 and 42). One testing set leads to a constant prediction quality.

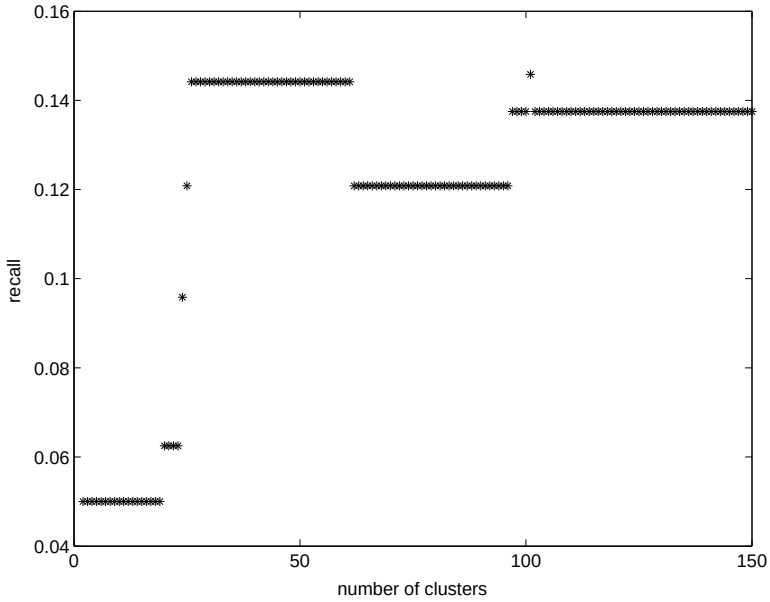


(a)

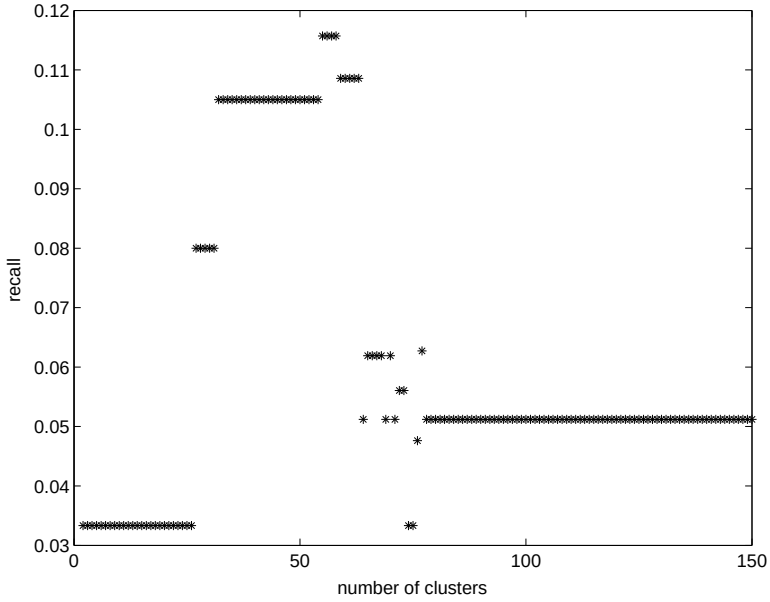


(b)

Figure 6.14: Prediction quality (recall) for observed navigation profiles; 20 test, 150 training data (the whole data set contained 200 data). The (assumed) number of clusters is changed in a range from 2 to 150. Both figures show the prediction quality for the same training set and different testing sets. The estimated number of clusters is about 78 in (a) and 80 in (b). The figure shows the indifference of the cluster estimation with respect to the testing data.



(a)



(b)

Figure 6.15: The same experiment as in figure 6.14 with a different set of profiles. The testing sets in (a) and (b) are different. The estimated number of clusters is about 25 in (a) and 55 in (b). Compared to figure 6.14 it can be seen that the estimated number of clusters is different and thus depending on the profile (training) set.

6.6 Summary: Classification and Prediction of User Profiles

The theoretical introduction in chapter 4 presents possibilities to characterize distributions of (navigation) graphs. Based on the simple characterization `Charac3` in section 4.5.2 (characterization by center elements) a procedure was introduced to estimate graph distributions in chapter 5. In this chapter, an algorithm to predict navigation profiles was presented, based on the estimated graph distribution, a classification step and a prediction step. Additionally, a method was suggested to measure the prediction quality. Based on these considerations, different experiments were performed to show properties of the prediction procedure. A first aspect is the dependence of the classification quality on the classification function. It was shown, how different classification functions, that are based on different knowledge about the distribution may have an influence on the classification quality. A second experiment tried to show the dependence of the prediction quality on the number of graph profiles that are known in the distribution estimation step. It was shown that usually, the more graph profiles are available, the better is the estimation result. However this is not true, if the variances of the clusters become too large. Finally it was the objective to estimate the number of clusters. For this purpose, different cluster numbers are assumed and the respective prediction quality is computed. It is made evident that usually, the real cluster number leads to a maximal prediction quality (or at least a local maximum of the prediction quality). This effect can be used to estimate the cluster number. This procedure was applied to observed graph profiles, registered by a real web server. It would have been desirable to show that the estimated cluster number based on one graph profile set leads to an optimal prediction quality with respect to the number of clusters for other profile sets. This result could not be proved yet. Different reasons for this were discussed. The number of profiles was perhaps not sufficient or the distribution of profiles in the different profile sets may have been different.

Chapter 7

Summary and Conclusions

In this thesis, we studied various aspects concerning the development of a software agent that supports a user who is looking for information in a well-defined (restricted) Web area. First a brief introduction into the problem was given and systems were presented that deal with similar problems. The problem that is laid emphasis on in this thesis is the orientation of a user in the Web environment, which is often refered to as the being *lost in hyper-space* problem.
In the first main part of the thesis (chapter 3) we presented the developed software system. The system was compared to former systems (mostly presented in the field of intelligent Internet agents), and the advantages of our system were described. Main aspects for the comparison are:

1. What kind of support information is provided?
2. How easy is it for a user to use the system, what are the client's technical requirements?
3. How does the system present support information, what are the means that may be applied for presentation purposes?
4. What information can the system acquire about a user?

As far as the first question is concerned, the presented system makes it possible to visualize the *navigation history* of a user, i.e. the set of previously visited web data objects and the applied hyperlinks. Moreover, due to a previous (offline) analysis of the Web area, the system makes it possible to present local maps of the hyperlink structure (adaptive local site-maps).[1] This kind of navigation support is similar to support strategies applied e.g. in [28] and [114]. These systems are different to the presented system with respect to the latter three questions (2,3 and 4).

[1] It is assumed that the hyperlink structure doesn't change in time (section 2.4.1).

One main aspect that is fundamental for questions two to four is the architecture of the respective software system. It was shown, that the architecture has an influence on the user's effort to use the system (question 2). The presented system consists of a component on the server side and a component on the client side. The client's component is a temporarily stored Java applet that is loaded from the server at the beginning of the navigation support and that is automatically installed by the client's browser. In contrast to similar systems that depend on a permanent installation of software on the client side, the presented method is easy to use and can be applied by almost all visitors, including those who rarely visit the Web area or who don't want to permit a (permanent) software installation. A disadvantage in this context is the requirement (the activation) of Java for the client's browser. Java is provided by most of the common browser software, some users however may not want to activate it.[2]

As far as the third question is concerned, is was shown that the presentation possibilities of the presented system are more extensive compared to systems that operate only on the server side like the one presented in [3]. The presentation facilities of the presented system are partially provided by Java. Most presentation options have however to be implemented in Java and thus have to be loaded down through the Internet. Since the time needed to load this program (Java applet) depends on the size of the program, the presentation possibilities are less extensive compared to software agents that have a permanently installed component on a client side.

The main aspect, when comparing the systems is the knowledge that can be acquired about a user (question 4). The presented system provides the possibility to register a client's set of navigation decisions (i.e. hyperlink activations). This is less information than may be obtained by systems that work on the client side, e.g. in [68], but in contrast to those systems, this information can be collected on the server side and can be used for collaborative filtering, which is an important aspect for the second part of this thesis.

In the second part of the thesis (sections 4, 5 and 6) a new collaborative filtering method was presented, that can be used to predict future navigation steps of a client based on the previous behavior of this user and registered navigation behavior of other users.[3] The data type that is registered by the system, i.e. the sets of the clients' navigation decisions, is regarded as a set of graph structures, with the nodes representing the data objects requested by a single client and the edges representing the (hyperlink) transitions. After

[2] e.g. due to still existing security problems

[3] These predictions can then be presented to a user with the method presented in the first part of the thesis.

a short introduction into graph theoretic basics, a method was presented to estimate the distribution of graphs in a graph space. This is one main step in the prediction process. In contrast to former methods, developed e.g. in the field of data mining [24], this estimation takes into account topological information that is provided by a metrics in a graph space. The distribution of the graphs is characterized by a set of center graphs, that are located in the centers of a number of graph clusters.[4] In this thesis, the evaluation of the presented graph distribution estimation was mainly based on simulated data. Graph profiles were generated according to a previously fixed distribution and the estimation results were compared to this original distribution. Some properties of the estimation procedure were shown in this way, like the convergence of the estimation quality for an increasing numbers of graphs, the dependence of the estimation quality on the variances of clusters and the influence of topological (edge) information of graphs in the estimation process.

These results were a basis for the main aspect that was considered in this second part of the thesis: the prediction of user profiles. A new prediction algorithm was presented, consisting of two main steps. The first step is a distribution estimation using previously registered (or simulated) data. The second step is a classification according to this estimated distribution, using (again) a distance measure between graphs. A measure for the quality of this prediction process was presented, that is based on a decomposition of (already registered) profiles into two parts. One part of a decomposed profile is used for the prediction, the other is used for the prediction quality test.

Some statistical properties of the prediction process were shown in experiments, like the convergence of the prediction quality for different classification functions, the convergence of the prediction quality for an increasing number of graphs and the effect of the size of the decomposed graphs in the prediction quality estimation step. One main experiment was the estimation of the number of clusters, which is a main property of the described graph distribution. It was shown for generated data, that the cluster number can be estimated by 'assuming' different cluster numbers and thus computing the prediction quality. The estimated cluster number is the (smallest) assumed cluster number, where the prediction quality has a local maximum.

The estimation results did hardly depend on the training set, which was used for the distribution estimation, and they did hardly depend on the testing set, used for the prediction quality estimation (which is the desired case). The same experiment was performed for observed navigation profiles, regis-

[4] This characterization can be extended with respect to the 'shape' of clusters (e.g. the distribution within a cluster), which hasn't been considered yet.

tered by an Internet server. It was shown, that the results hardly depended on the testing set similar to the previous experiments, but the estimation results depended on the training sets. Different arguments why the latter convergence was not observed were proposed, like an insufficient number of profiles and a possible change in the distribution.

To summarize, the two main results of this thesis are on the one hand, the realization of a software system that makes it possible to register navigation behavior and to visualize navigation history and local site maps for almost every Internet user. On the other hand a new learning algorithm for the available information about user behavior was presented. Basic statistical properties of the algorithm were verified and the algorithm was shown to give reliable prediction results near the optimum for simulated data with regard to an estimated number of clusters.

The motivation to consider the two support strategies, i.e. the presentation of maps of the hyperlink structure and the hyperlink prediction in this thesis, was to lay a basis for more advanced search support systems in the Internet. These systems shall help a user to orientate himself by presenting maps of the hyperlink structure, which are adapted to his individual needs. Only those data objects and routes in the Web should be presented, that are relevant for a specific user.
The thesis shows that there are many research problems that still have to be considered. As far as the map presentation is concerned, the layout problem of the presented graphs should be examined more closely. Consecutive graph presentations should have a similar layout in order to make it possible for a user to 'understand' the change. Further problems are the visualization of hyperlink graph structures that change in time (section 2.4.1) and the visualization of hyperlink structures that contain web pages which are generated for each individual user (dynamic web pages). Another research problem is the generation of hyperlink structure views with regard to semantic aspects. As far as the prediction technique is concerned, different distribution characterizations, graph distance functions, distribution estimation (including clustering), classification and prediction techniques should be compared to each other with respect to the resulting prediction quality. An interesting additional problem is the velocity of the convergence of the estimation results when the profile number is increased, depending on the different estimation settings. In this field, there is still a lot of work to do. The presented prediction strategy should be compared to other prediction methods, like Markov prediction or other clustering techniques. The thesis provides methods to test and to evaluate the different prediction strategies.

A main open question still to examine is, if the estimation results are valuable for a user. Up to now, only the prediction quality was considered, which can easily be tested, using registered profiles. The assumption that predictions are related to the relevance of the predicted objects for a user still has to be examined in detail. One possibility would be to present the prediction results to users (applying the same presentation technique as described in the thesis) and to let the users evaluate the relevance of the presented data objects (and sequences).
One problem that may occur when the estimation results are presented to users may be denoted as 'snowball effect'. If some users follow a wrong path, the system may learn the wrong path and present it to other users who (therefore) make the same wrong navigation decisions. Thus, again, the system learns the wrong navigation path and makes bad predictions in the future. The reason, why this effect is unlikely to be a problem for the presented system is the fact that the system may register (and thus learn from) navigation profiles, even if the users don't activate the support system (the redirection technique may be activated for all users or action inferring may be applied). Thus the percentage of users that actually apply the system has to be sufficiently high in order to cause a snowball effect.
The prediction method presented in this thesis may however be valuable for a user even if the predicted data elements are not relevant for this user. If the predictions are sufficiently good (and not destroyed by a snowball effect), the user would have visited these wrong data objects anyway. By using the system the user may find these irrelevant data objects easier and in a shorter time. Therefore the user saves time and can concentrate on finding the 'right' data objects.

Appendix A

Graph Matching Algorithms

In this appendix graph matching algorithms are presented that make it possible to compute some of the graph distances defined in section 4.4. These algorithms have been previously described by several authors. Graph matching is a problem that is e.g. applied in the field of Computer Vision in order to compare (parts of) images.[1] In many cases the problem is to compare a new input graph to a set of model graphs, where the model graphs are tested to be exact or error-correcting subgraphs of the input graph (definition 20). Well known algorithms are e.g. the Ullman algorithm for exact subgraph-isomorphism detection [109] or the A^* algorithm for inexact (error-correcting)-subgraph isomorphism detection presented by [102], [101] and [108]. These (exact and error-correcting) techniques have recently been improved in [74], [75] by considering common substructures in the model graphs. The idea is first to decompose the model graphs and to store them in a network in an off-line step. The (online) matching step starts by comparing the small structures in the network to the input graph and continues by combining the results in order to match larger substructures.
The algorithms for exact subgraph-isomorphism matching (and also the exact decomposition-based algorithms) can't be applied in our context, since it is the problem to look for common substructures, i.e. the problem is to find the largest common subgraph in the exact case. The error-correcting algorithms may however be applied in this context.
In the following text at first an algorithm is described to find maximal common subgraph isomorphisms by a 'clique' detection. Similar algorithms have been presented in [66], [19], [12] and [77]. Next, an algorithm for the error-correcting graph and subgraph isomorphism problem is presented, based on the $A^\star$ tree search algorithm. Most of the definitions necessary for the de-

[1] In practical applications it is also one main problem to find a graph representation for an image [64], [60].

scription of the algorithms have been presented in section 4.3.

A.1 A maximal-common-subgraph algorithm for graphs by clique detection

The (well-known) algorithm applied in the experiments makes it possible to find all maximal common subgraphs of a graph $G_1 = (V_1, E_1, \mu_1, \nu_1)$ and a graph $G_2 = (V_2, E_2, \mu_2, \nu_2)$. The presented algorithm is based on finding all the *cliques* of the so-called *association graph* (a clique is a subgraph of a graph, where each vertex is connected to each other vertex). This assoc. graph is constructed by mapping each vertex $u_1 \in V_1$ onto each vertex $u_2 \in V_2$ with the same label. Each mapping (u_1, u_2) is a vertex of the association graph $G_A = (V_A, E_A, \mu_A, \nu_A)$. Two vertices (u_1, u_2) and (u_1', u_2') of this graph are compatible, denoted by an edge, if there exist the edge $(u_1, u_1') \in E_1$ and $(u_2, u_2') \in E_2$. A maximal clique in the association graph represents a largest common subgraph of G_1 and G_2.

Definition 24 *Given the two graphs* G_1 *and* G_2*, an association graph for* G_1 *and* G_2 *is a graph* $G_A = (V_A, E_A, \mu_A, \nu_A)$ *with* $V_A \subseteq V_1 \times V_2$ *and* $E_A \subseteq V_A \times V_A$ *with*

- $V_A = \{(u_1, u_2) \in V_1 \times V_2 \mid \ \mu_1(u_1) = \mu_2(u_2)\}$
- $e_A := (u_A, u_A') = ((u_1, u_2), (u_1', u_2')) \in E_A$ *if:*
  a) $u_1 \neq u_1'$ *and* $u_2 \neq u_2'$
  b) exists $(u_1, u_1') \in E_1 \Longrightarrow$ *exists* $(u_2, u_2') \in E_2$ *and* $\nu_1(e_1) = \nu_2(e_2)$
  c) no edge $(u_1, u_1') \in E_1$ *exists* $\Longrightarrow$ *no edge* $(u_2, u_2') \in E_2$ *exists*

The algorithm in figure A.1 shows the main idea how the search for maximal cliques can be applied for the maximal common subgraph problem. At first the association graph is computed, then an algorithm for maximal clique detection is applied. If $C = \{(u_{a_1}, u_{b_1}), ...(u_{a_n}, u_{b_n})\}$ is a maximal cliques found for the association graph, there exists a graph isomorphism from the subgraph $S_1 = (V_{S_1}, E_{S_1})$ with $V_{S_1} = \{u_1 | (u_1, u_2) \in C\}$ of G_1 to the subgraph $S_2 = (V_{S_2}, E_{S_2})$ with $V_{S_2} = \{u_2 | (u_1, u_2) \in C\}$ of G_2.

The main idea for the clique detection is to choose a vertex u_{A_1} from the association graph and add it to an empty list C. The list C is then extended by adding a second vertex u_{A_2} such that there exists an edge between both nodes in the association graph. In general, every vertex added to C has to be connected to every vertex in C, which is thus a clique. The process stops,

Maximal common subgraph by maximal clique-detection

1. Create an association graph $G_A = (V_A, E_A, \mu_A, \nu_A)$ from $G_1 = (V_1, E_1, \mu_1, \nu_1)$ and $G_2 = (V_2, E_2, \mu_2, \nu_2)$ according to definition 24.
2. Find set of maximal cliques '$Cliq$' according to the algorithm in fig. A.2 Clique-Detection(G_A,$min\{V_1, V_2\}$).
3. for each $C \in Cliq$
   let $S_1 = (V_{S_1}, E_{S_1})$ be the subgraph of G_1 with $V_{S_1} = \{u_1 | (u_1, u_2) \in C\}$ and $S_2 = (V_{S_2}, E_{S_2})$ be the subgraph of G_2 with $S_2 = \{u_2 | (u_1, u_2) \in C\}$. Both subgraphs are isomorphic. Output the implied graph isomorphism function $f : S_1 \longrightarrow S_2$.

Figure A.1: Graph-matching based on clique detection.

Clique-detection($G_A = (V_A, E_A, \mu_A, \nu_A)$, S)

1. Let i=1, $C, C_{max}, Cliq = \emptyset$, $L(k) = \emptyset \ \forall k = 1, ...S$.
2. if $\exists u_A \in V_A$ with $u_A \notin L(k) \ \forall k \leq i$ then:
   a) $L(i) := L(i) \cup \{u_A\}$
   b) if exists an edge $e_A = (u_A, w_A) \in E_A$ for all $w_A \in C$, then $C := C \cup \{u_A\}$ and $i := i + 1$ and
      if $|C| = |C_{max}|$ then set $Cliq = Cliq \cup \{C\}$
      if $|C| > |C_{max}|$ then set $Cliq = \{C\}$ and $C_{max} = C$.

   else

   c) remove the i-1-th vertex from C
   d) set $L(i) := \emptyset$ and $i := i - 1$.
3. if $i > 0$ goto 2.
4. output Cliq

Figure A.2: Algorithm for clique detection.

if no more vertices can be added. The process then removes the previously added node and adds a node that hasn't been tried before. A set of lists L_i, $i = 1, ..|V_A|$ is introduced to keep track of the nodes that have already been tried.
The algorithm in figure A.2 shows the different steps for the clique search. At first C, C_{max}, $Cliq$ and L_i, $i = 1, ..S$ are initialized as empty, i=1. In the second step, a vertex u_A is chosen that has not been used before, i.e. $u_A \notin L(k)$ for all $k \leq i$. u_A is added to $L(i)$. It is checked if $C \cup \{u_A\}$ is a clique in step 2b. If $C \cup \{u_A\}$ is a larger clique than the current maximal clique, the set of maximal cliques (C_{max}) is emptied and initialized with C. If C has the size of a maximal clique, it is added to the set of cliques ($Cliq$). If a new u_A value couldn't be found, the steps 2c and 2d perform a backtracking. The last vertex (i-1) is removed from C, $L(i)$ is emptied, and i is decremented by one. As long as it is possible to find larger cliques, the algorithm continues in step 2. Finally if $i = 0$ all cliques have been detected and are output.
The main drawback of the described algorithm is its complexity. Let $|V_1| = n$ and $|V_2| = m \in \mathbb{N}$ with $n < m$. In the best case both graphs are uniquely labeled and each vertex of G_1 matches to exactly one vertex of G_2. The corresponding association graph has $O(n)$ vertices and can be created in $O(nm)$ steps. The maximal clique search requires the extension of the initial clique $O(n)$ times with $O(n)$ connectivity tests. The total complexity is thus bounded by $O(nm + n^2)$.
In the worst case, the two graphs are identically labeled and each graph is completely connected. The association graph consists of $O(nm)$ vertices and each vertex is connected to $O(nm - n)$ other vertices. At each level of the backtracking process a clique has to be extended with $O(nm)$ vertices. There are $O(n)$ levels and thus the complexity is $O((nm)^n)$ in the worst case.

A.2 Error-correcting subgraph isomorphism detection

In this section an algorithm for the error-correcting subgraph isomorphism problem is presented, that is based on the A^* algorithm [79]. Some basic definitions have been given in section 4.3. This algorithm was not used in the experiments, it is only presented here to demonstrate, how the distance measure between graphs in section 4.4.4 may be computed. Given two graphs $G_1 = (V_1, E_1, \mu_1, \nu_1)$ and $G_2 = (V_2, E_2, \mu_2, \nu_2)$ the objective is to find an optimal error correcting subgraph isomorphism (Δ, f), i.e. there

exists a subgraph isomorphism from $\Delta(G_1)$ to G_2 with minimal costs. Let $p = \{(u_1, w_1), ...(u_k, w_k)\}$, $u_1, ..u_k \in V_{G_1}$ and $w_1, ..w_k \in V_{G_2} \cup \{\text{⫛}\}$ be a mapping from the vertices of G_1 to the vertices of G_2 such that for all tuples $(u_i, w_j), (u_j, w_j) \in p : u_i \neq u_j$ and $w_i \neq w_j$ unless $w_i =$ ⫛ (the symbol ⫛ denotes the deletion of a node). Define $V_1^p := \{u | (u, w) \in p\}$ and $V_2^p := \{w | (u, w) \in p\}$. A mapping p implies a set of edit operations Δ^p. Let $(u, w) \in p$. If $w \neq$ ⫛ and $\mu_1(u) \neq \mu_2(w)$ then a substitution of the respective labels is implied by p. If $w =$ ⫛, the mapping p implies the deletion of v.
If (u', w') is another element of p, the implied edge operations can be demonstrated. If $e_1 = (u, u') \in E_1$ and $e_2 = (w, w') \in E_2$ and $\nu_1(e_1) \neq \nu_2(e_2)$, the substitution of the respective edge labels is implied. If there exists $e_1 = (u, u') \in E$ but there is no edge $e_2 = (w, w') \in E_2$, the deletion of the edge e_1 is implied. In the opposite case, p implies the insertion of an edge $e_2 = (w, w')$.
The subgraph isomorphism implied by p is the mapping f^p from $\Delta^p(G_1)$ to G_2:

$$f^p(v) = \begin{cases} w & \text{if } (v, w) \in p \text{ and } w \neq \text{⫛} \\ \text{undefined} & \text{otherwise} \end{cases} \tag{A.1}$$

The costs of p are the costs of the implied edit operations, $C(p) := C(\Delta^p)$.
The problem when looking for optimal error correcting subgraph isomorphisms can be restated as the search for a mapping p with $V_1^p = V_1$ with minimal costs. The algorithm in figure A.3 gives a formal description of the error-correcting subgraph isomorphism search. The algorithm starts with an initialization of an Open list which has got the function to organize the mappings during the search process. At first, a node is chosen from V_1 and all mappings to $V_2 \cup \{\text{⫛}\}$ are stored in Open. If Open is empty in step 2 the algorithm terminates. In step 3 the mapping in Open with the least cost $C(p)$ is chosen and removed from Open. The cost is compared to a threshold value. At first the threshold is an upper bound for the costs of a complete subgraph isomorphism. If a subgraph isomorphism is found with a lower cost, the threshold is set to this value in order to find 'optimal' e.c.s.g.-isomorphisms in step 5. The algorithm terminates in step 4 if all remaining subgraph isomorphisms have higher costs. In step 5 it is checked, if p is a complete subgraph isomorphism and output in this case, the threshold value is adjusted and the algorithm continues in step 2. If p is not yet complete, in the steps 7a,b a new vertex u_{k+1} is chosen and all mappings from u_{k+1} to the remaining elements of V_2 are used to generate new mappings $p_{new} = p \cup \{(u_{k+1}, w)\}$ with $w \in V_2 \backslash V_2^p \cup \{\text{⫛}\}$ that are added to Open. The costs of these mappings are computed and the algorithm continues in step 2. In the best case for the algorithm, both graphs G_1 and G_2 are uniquely la-

E.C.S.I by $A^*(G_1 = (V_1, E_1, \mu_1, \nu_1), G_2 = (V_2, E_2, \mu_2, \nu_2))$

1. initialize Open: for each node $w \in V_2\{\Uparrow\}$ create a mapping $p = (u_1, w)$ and add it to Open, Open := Open $\cup \{p\}$. Let the cost $C(p)$ be the cost of the implied label substitution.
2. If Open is empty then exit.
3. select $p \in$ Open such that $C(p)$ is minimal, remove p from Open.
4. if $C(p) >$ treshold then exit.
5. if p is a complete mapping from G_1 to G_2 then output p, set threshold := $C(p)$.
6. let $p = \{(u_1, w_i), ...(u_k, w_j)\}$ and $V_2^p = \{w | (u, w) \in p$ and $w \in V_2\}$
7. for all $w \in (V_2 \backslash V_2^p) \cup \{\Uparrow\}$
   a) set $p' = \{(u_1, w_i), ...(u_k, w_j), (u_{k+1}, w)\}$ and compute $C(p')$ (text).
   b) add p' to Open
8. goto 2.

Figure A.3: Error correcting subgraph isomorphism by A^*.

beled and G_1 contains an isomorphic copy of G_2. If $|V_1| = n$ and $|V_2| = m$, the algorithm first maps the first vertex of G_1 onto every node of G_2 and thereby generates $O(m)$ mappings. Only the mapping with zero cost is extended, resulting in $O(m-1)$ new mappings. For n vertices in G_1 this results in $O(nm)$ mappings. The algorithm performs $O(n)$ edge tests for each mapping and thus the complexity is bounded by $O(n^2 m)$.
In the worst case, the error in the graph G_1 is very large and the edit costs become very high. Thus each mapping has to be extended. The first vertex can be mapped onto $O(m)$ vertices in the graph G_2. Each of the mappings is extended, resulting in $O(m(m-1))$ mappings. For the k-th vertex of G_2 there will be $O(m^k)$ mappings. The total number of mappings is thus bounded by $O(nm^n)$. Since each mapping requires $O(n)$ edge tests, the complexity is $O(n^2 m^m)$.
In [116] a method was presented to improve the performance of the algorithm. Given a mapping p the minimal costs of any mapping p' implying p is estimated. The idea is to estimate for each vertex in $V_1 \backslash V_1^p$ the minimal cost for mapping it onto a vertex in $V_2 \backslash V_2^p$. Each $v \in V_1 \backslash V_1^p$ is mapped onto each vertex $w \in V_2 \backslash V_2^p$ and the implied edit costs of the mappings are compared to each other. First the label $\mu_1(u)$ has to be substituted by $\mu_2(w)$ and then for each mapping $(u', w') \in p$ the edge (u, u') has to mapped onto (w, w') by an edge operation. The sum of the minimal cost for each vertex

$v \in V \backslash V^p$ is a lower bound of the real future costs of p (that is used in step 3).

The complexity of the future cost estimation is $O(nm)$ for each mapping, since each of the remaining vertices in $V_1 \backslash V_1^p$ is mapped onto each of the remaining vertices in $V_2 \backslash V_2^p$. The complexity in the best case thus becomes $O(n^2m^2)$ and in the worst case $O(n^2m^{n+1})$. The reduction of the number of mappings does not show up in the complexity analysis.

Appendix B

Clustering Algorithms

In section 4.6 one step in the described distribution estimation procedure is the clustering of graph data. The applied algorithm is a *hierarchical* clustering technique. These clustering techniques can be subdivided into agglomerative methods, which imply a series of fusions of the individuals into groups, and divisive methods, which separate the elements into finer groupings [33]. Both techniques are based on a distance measure between the respective elements.[1] Let $\mathtt{dist}(.,.)$ denote this distance measure. The applied technique, single linkage or nearest neighbor clustering is an agglomerative technique. Different agglomerative techniques are based on different functions applied to define a distance between groups ('group distance'). Let C and D be two clusters (groups), consisting of $|C| = n \in \mathbb{N}$ and $|D| = m \in \mathbb{N}$ elements. Let $\{x_1, ...x_n\}$ be the elements in C and $\{y_1, ...y_m\}$ be the elements in D.
One first possibility to define a (group-) distance, applied by the single linkage algorithm, uses the smallest distance between two objects in the two groups:

$$d(C, D) = \min_{i,j}(\mathtt{dist}(x_i, y_j)), \qquad i \in \{1, ..n\},\ j \in \{1, ...m\} \tag{B.1}$$

In contrast to this, the complete linkage algorithm uses the largest distance between objects in two groups. The average linkage uses the average distance between all pairs of objects in cluster C and D.

$$d(C, D) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} \mathtt{dist}(x_i, y_j) \tag{B.2}$$

[1] In the 'usual' case, this distance measure operates in an Euclidean Vector space, like the Euclidean distance, the Mahalanobis distance, the City Block metrics, the Minkowski metrics etc. In the presented case, a graph metrics is used as defined in section 4.4.

Centroid linkage uses the distance between the centroids of the two groups

$$d(C, D) = d(\frac{1}{n}\sum_{i=1}^{n} x_i, \frac{1}{m}\sum_{j=1}^{m} y_j) \tag{B.3}$$

Other techniques exists like the Ward linkage, that is based on the the centroid method [112].
The steps of the different algorithms are basically the same. Let A be a set of basic elements or groups consisting of these elements. At the beginning A consists of the basic elements. In the first step the two elements are chosen, that have, according to the applied metrics, the smallest distance. These elements are joined to form a two member cluster (group). The two elements are deleted in A and the new group is inserted. A new (group-) distance matrix is computed, that contains the distances between the remaining elements in A. The elements (original elements or formed clusters) with the smallest (group-) distance are merged. The algorithm continues in this way until one group in A, containing all the elements, is obtained. This is the fundamental algorithm; more efficient methods have been presented e.g. by [91] and [103].
Problems of these algorithms are e.g. the *chaining* problem in the case of single linkage or a bias towards finding 'spherical' clusters in the case of e.g. the centroid clustering [33]. These problems were examined in various articles and often depend on the specific problem domain.
It can be seen that not all described cluster methods can be applied for the problem in this thesis. Since many algebraic operations that are available in a vector space are not available in a graph space (the definitions in section 4.3 have another meaning), it is e.g. not possible to apply centroid linkage. Only those cluster algorithms can be applied, that are based only on the distance operation. Further algebraic operations like summation or multiplication make it difficult to use the method.
In the experiments, the implementation of the single linkage algorithm was used, provided by the Matlab (Statistics-) tool-box [88].

Appendix C

Preparation of Registered Access-log Data

In order to test the algorithms presented in chapter 6 for real profiles, data registered in access-log files by the web server of the Cognitive Systems group at Kiel University were exploited. These data were preprocessed in order to obtain the data structure necessary for the algorithms. A simplified example for the data that are available in the access-log files can be seen in figure C.1. It is assumed that a client can be uniquely identified by his IP-address

```
62.226.76.63 - - [01/Aug/2001:00:03:11 +0200] "GET / HTTP/1.0"
138.15.10.5 - - [01/Aug/2001:00:03:46 +0200] "GET / tbl/tobue.html HTTP/1.0"
62.226.76.63 - - [01/Aug/2001:00:18:17 +0200] "GET /staff.html HTTP/1.0"
138.15.10.5 - - [01/Aug/2001:00:20:00 +0200] "GET / tbl/publications.html
HTTP/1.0"
138.15.10.5 - - [01/Aug/2001:00:25:39 +0200] "GET / tbl/projects.html HTTP/1.0"
62.226.76.63 - - [01/Aug/2001:00:42:06 +0200] "GET / chp/chris.html HTTP/1.0"
```

Figure C.1: Partial access-log file, consisting of a client's IP-address, a time stamp and the requested data object.

(which may not be correct in all cases, as discussed in section 3.4.3). Figures C.1, C.2 and C.3 show the steps for the access-log processing. In a first step the set of data objects requested by a user are extracted. For this purpose, a maximal 'dwell time' is fixed, as discussed in section 4.1, which is set to 50 minutes. The entries in the access-log files are first sorted by the IP-address and then by their time stamp (which is the original order). Then the requests belonging to one client that occur in the dwell time interval beginning with the first request are collected and stored. Now each profile contains the information about the IP-address of a client and the set of his

requested data objects, ordered by the time stamp (fig. C.2). In a last step, the edges between the requested data objects, i.e. the transitions between the data objects are added (figure C.3). In the experiments in chapter 6 consecutive data objects are linked by an edge in the graph structure. A further method is to apply action inferring techniques with regard to the hyperlink structure (section 3.4.2).

```
62.226.76.63        /
                    /staff.html
                    / chp/chris.html

138.15.10.5         / tbl/tobue.html
                    / tbl/publications.html
                    / tbl/projects.html
```

Figure C.2: Two sets of user requests, extracted from the access-log files.

```
62.226.76.63
nodes       1       /
            2       /staff.html
            3       / chp/chris.html
edges       1       2
            2       3
138.15.10.5
nodes       1       / tbl/tobue.html
            2       / tbl/publications.html
            3       / tbl/projects.html
edges       1       2
            2       3
```

Figure C.3: Two graph profiles, extracted from the sets of requests. A graph consists of a set of nodes (requested URL addresses) and a set of edges. An edge '1 2' denotes a transition from node 1 to node 2 in the node list.

Bibliography

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia, editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[2] D. W. Albrecht, I. Zukerman, and A. E. Nicholson. Pre-sending documents on the WWW: A comparative study. In *IJCAI*, pages 1274–1279, 1999.

[3] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. WebWatcher: A learning apprentice for the World Wide Web. In *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, pages 6–12, 1995.

[4] U. Babiak. *Effektive Suche im Internet: Suchstrategien, Methoden, Quellen.* O'Reilly, 1999.

[5] S. C. Basak, V. R. Magnuson, G. J. Niemi, R. R. Regal, and G. D. Veith. Topological indices: Their nature, mutual relatedness, and applications. *Mathematical Modelling*, 8:300–305, 1987.

[6] B. Bekavac and M. Rittberger. Ein Navigationsassistent für das WWW. In H. Zimmermann and V. Schramm, editors, *Knowledge Management und Kommunikationssyteme. Proceedings des 6. Internationalen Symposiums für Informationswissenschaft (ISI'98)*, pages 438–452. Universitätsverlag Konstanz, Nov 1998.

[7] J. P. Bigus and J. Bigus. *Constructing Intelligent Agents using Java.* Wiley and Sons, 2001.

[8] B. Bollobas. *Modern Graph Theory.* Springer, 1998.

[9] W. Brenner, R.Zarnekow, and H.Wittig. *Intelligente Softwareagenten.* Springer, 1998.

[10] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[11] A. Z. Broder, S. R. Kumar, F. Maghoul, P.Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener. Graph structure in the Web. *WWW9/Computer Networks*, 33(1-6):309–320, 2000.

[12] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[13] M. R. Brown. *Netscape Communicator 4*. Markt & Technik Buch- und Software-Verlag GmbH, 1997.

[14] P. Brusilovsky. Efficient techniques for adaptive hypermedia. In C. Nicholas and J. Mayfield, editors, *Intelligent Hypertext*, LNCS, pages 12–30. Springer, 1997.

[15] P. Bühlmann and A. J. Wyner. Variable length markov chains. *Annals of Statistics*, 27(1):480–513, 1999.

[16] H. Bunke. On a relation between graph edit distance and maximum common subgraph. In *Pattern Recognition Letters*, 18, pages 689–694. Elsevier, 1997.

[17] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19:255–259, 1998.

[18] E. Carmel, S. Crawford, and H. Chen. Browsing in hypertext: A cognitive study. *Transactions on System, Man and Cybernetics*, 22:865–883, 1992.

[19] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[20] S. Chakrabarti, B. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. Mining the Web's link structure. *IEEE Computer*, 32(8):60–67, 1999.

[21] S. Chakrabarti, B. Dom, S. R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, D. Gibson, and J. Kleinberg. Neue Pfade durch den Internet-Dschungel. *Spektrum*, 8:44–49, 1999.

[22] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific Web resource discovery. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(11–16):1623–1640, 1999.

[23] L. Chen and K. Sycara. WebMate: A personal agent for browsing and searching. In K. P. Sycara and M. Wooldridge, editors, *Proceedings of the 2nd International Conference on Autonomous Agents (Agents'98)*, pages 132–139. ACM Press, 1998.

[24] M. S. Chen, J. S. Park, and P. S. Yu. Data mining for path traversal patterns in a web environment. In *Sixteenth International Conference on Distributed Computing Systems (ICDCS)*, pages 385–392, 1996.

[25] W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI*, 17(8):749–764, 1995.

[26] G. Cornell and C. Horstmann. *Java bis ins Detail*. Heise, 1996.

[27] J. Cove and B. C. Walsh. Online text retrieval via browsing. *Information Processing and Management*, 24(1):31–37, 1988.

[28] W. Dalitz and G. Heyer. *Hyper-G, Das Internet-Informationssystem der 2.Generation*. dpunkt-Verlag, 1995.

[29] D. D'Aloisi and V. Giannini. The Info Agent: An interface for supporting users in intelligent retrieval. In *Proceedings of the ERCIM Workshop on Towards User Interfaces for All: Current Efforts and Future Trends*, pages 145–158, 1995.

[30] P. de Bra, P. Brusilovsky, and R. Conejo, editors. *Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, AH 2002, Malaga, Spain, LNCS 2347*. Springer, May 2002.

[31] D. M. Edwards and L. Hardman. 'Lost in hyperspace': Cognitive mapping and navigation in a hypertext environment. In R. McAleese, editor, *Hypertext: Theory into practice*, pages 105–125. Oxford: Intellect Limited, 1989.

[32] K. Ericsson and H. A. Simon. *Protocol analysis: verbal reports as data*. Cambridge, MA: MIT Press, 1993.

[33] B. Everitt. *Cluster Analysis*. Edward Arnold, 3 edition, 1993.

[34] M. Fernandez and G. Valiente. A graph distance metric combining maximum common subgraph and minimum common supergraph. *Pattern Recognition Letters*, 22(6-7):753–758, 2001.

[35] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP, RFC 2616, 1999.

[36] R. T. Fielding. Maintaining distributed hypertext infostructures: welcome to MOMspider's Web. *Computer Networks and ISDN Systems*, 27(2):193–204, 1994.

[37] D. Flanagan. *Java Examples in a Nutshell*. O'Reilly, 1998.

[38] D. Flanagan. *Java Foundation Classes*. O'Reilly, 1999.

[39] D. Flanagan. *Java in a Nutshell*. O'Reilly, 1999.

[40] D. Flanagan. *JavaScript*. O'Reilly UK, Dec 2001.

[41] K. Florek, J.Lukaszewicz, J.Perkal, H.Steinhaus, and S.Zubrzycki. Sur la liason et la division des points d'un ensemble fini. *Collequium Mathematicum*, 2:282–285, 1951.

[42] J. Furner, D. Ellis, and P. Willet. The representation and comparison of hypertext structures using graphs. In M. Agosti and A. F. Smeaton, editors, *Information Retrieval and Hypertext*, pages 75–96. Kluwer Academic Publishers, 1996.

[43] N. D. Gershon, J. LeVesseur, J. Winstead, J. Croall, A. Pernick, and W. Ruh. Visualizing Internet resources. In *Proceedings of Information Visualization*, pages 122–128. Los Alamitos, CA:IEEE, 1995.

[44] D. Gilbert. Intelligent agents: the right information at the right time. Technical report, IBM Corporation, Research Triangle Park, NC, USA, May 1997.

[45] C. F. Goldfarb and P. Prescod. *XML Handbuch*. Verlag Prentice Hall, 1999.

[46] I. Graham. *HTML Sourcebook*. John Wiley & Sons, Inc., 1995.

[47] V. N. Gudivada, V. V. Raghavan, W. I. Grosky, and R. Kasanagottu. Information retrieval on the World Wide Web. *IEEE Internet Computing*, 1(5):58–68, Sep 1997.

[48] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly UK, 2001.

[49] I. Herman, G. Melancon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transaction on Visualization and Computer Graphics*, 6(1):24–43, 2000.

[50] A. Heuer. *Objektorientierte Datenbanken : Konzepte, Modelle, Standards und Systeme*. Addison-Wesley-Longman, 2 edition, 1997.

[51] C. Hoelscher and G. Strube. Web search behavior of Internet experts and newbies. In *Proceedings of the 9th Int. World Wide Web Conference*, pages 337–346, 2000.

[52] S. Holzner. *HTML Black Book: The Programmer's Complete HTML Reference Book*. Coriolis Group Books, May 2000.

[53] J. I. Hong and J. A. Landay. WebQuilt: A framework for capturing and visualizing the web experience. In *WWW 2001*, ACM, pages 717–724, May 2001.

[54] S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32:241–254, 1967.

[55] P. Kahn and K. Lenk. *Mapping Web Sites*. Robot Vision SA, 2001.

[56] J. Kittler, W. J. Christmas, and M. Petrou. Probabilistic relaxation for matching of symbolic structures. In H. Bunke, editor, *Advances in Structural and Syntactic Pattern Recognition*, pages 471–480. World Scientific, 1992.

[57] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632, 1999.

[58] M. Klusch, editor. *Intelligent Information Agents*. Springer, 1999.

[59] T. Kopetzky and M. Mühlhäuser. Visual preview for link traversal on the WWW. *Proceedings of the 8th Intl. WWW Conference*, pages 447–454, May 1999.

[60] N. Krüger, M. Potzsch, and C. von der Malsburg. Determination of face position and pose with a learned representation based on labeled graphs. Technical report, Institut für Neuroinformatik, Ruhr-Universität Bochum, Jan 1996.

[61] D. Kukulenz. Prediction of navigation profiles in a distributed internet environment through learning of graph distributions. In P. de Bra, P.Brusilovsky, and R.Conejo, editors, *Adaptive Hypermedia and Adaptive Web-Based Systems, Second International Conference, AH 2002, Malaga, Spain, LNCS 2347*, pages 233–241. Springer, May 2002.

[62] D. Kukulenz and J. Pauli. Navigation-dependent visualization of distributed Internet structures. In *IEEE Conference on Information Visualization*, pages 518–523, 2000.

[63] D. Kukulenz and J. Pauli. A software agent for adaptive visualization of hyperspace based on clustering of navigation graphs. In *Applied Informatics, Int. Sympos. on Software Engineering, Databases and Applications, Proc. of the IASTED International Conference*, pages 248–253, 2002.

[64] P. Kuner and B. Ueberreiter. Pattern recognition by graph matching–combinatorial versus continuous optimization. *Intl. J. Pattern Recognition and Artificial Intelligence (IJPRAI)*, 2:527–542, 1988.

[65] B. Laurie and P. Laurie. *Apache. The Definitive Guide.* O'Reilly & Associates, Inc., 1997.

[66] G. Levi. A note on the derivation of maximal common subgraphs of two directed or undirected graphs. *Calcolo*, 9:341–354, 1972.

[67] M. S. Lew, editor. *Principles of Visual Information Retrieval.* Springer, 2001.

[68] H. Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 924–929. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1995.

[69] H. Lieberman. Autonomous Interface Agents. In *Proceedings of the ACM Conference on Computers and Human Interface, CHI-97*, pages 67–74, Atlanta, Georgia, 1997.

[70] H. Lieberman. Beyond information retrieval: Information agents at the MIT Media Lab. *Künstliche Intelligenz*, 12(3):17–23, 1998.

[71] H. Lieberman, N. W. V. Dyke, and A. S. Vivacqua. Let's browse: a collaborative web browsing agent. In *Proceedings of the 1999 International Conference on Intelligent User Interfaces (IUI'99)*, pages 65–68, Los Angeles, CA, USA, 1999. ACM Press.

[72] C. Liu, J. Peek, R. Jones, B. Buus, and A. Nye. *Internet-Server, Einrichten und Verwalten*. O'Reilly/International Thomson Verlag GmbH &Co KG, 1995.

[73] H. Maurer, N. Scherbakov, Z. Halim, and Z. Razak. *From Databases to Hypermedia*. Springer, 1998.

[74] B. Messmer and H. Bunke. *Efficient graph matching algorithms for preprocessed model graphs*. PhD thesis, Bern University, 1996.

[75] B. Messmer and H.Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):493–504, 1998.

[76] Microsoft. *Internet Explorer 6.0 Resource Kit*. Microsoft Press, Okt 2001.

[77] S. H. Myaeng and A. Lopez-Lopez. Conceptual graph matching: a flexible algorithm and experiments. *Journal of Experimental and Theoretical Artificial Intelligence*, 4:107–126, 1992.

[78] C. Nicholas and J. Mayfield, editors. *Intelligent Hypertext - Advanced Techniques for the World Wide Web*. Springer, 1997.

[79] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers, 1986.

[80] M. J. Pazzani, J. Muramatsu, and D. Billsus. Syskill Webert: Identifying interesting web sites. In *AAAI/IAAI, Vol. 1*, pages 54–61, 1996.

[81] M. Perkowitz and O. Etzioni. Adaptive web sites: an AI challenge. In *Proceedings of the 15th International Joint Conference on AI (IJCAI-97) (1)*, pages 16–23, 1997.

[82] M. Perkowitz and O. Etzioni. Adaptive web sites: Automatically synthesizing web pages. In *AAAI/IAAI*, pages 727–732, 1998.

[83] M. Perkowitz and O. Etzioni. Towards adaptive web sites: Conceptual framework and case study. *Artificial Intelligence*, 118(1–2):245–275, 2000.

[84] N. Pohlmann. *Firewall-Systeme : Sicherheit für Internet und Intranet*. MITP-Verlag, 2001.

[85] J. Postel. *Internet Protocol- DARPA Internet Program Protocol Specification*. STD 5, RFC 791, DARPA, Sep 1981.

[86] J. Postel. *Transmission Control Protocol - DARPA Internet Program Protocol Specification*. STD 5, RFC 793, DARPA, Sep 1981.

[87] J. Postel and J. Reynolds. *File Transfer Protocol (FTP)*. W3C, Network Working Group, RFC 959, Oct 1985.

[88] D. Redfern and C. Campbell, editors. *The MATLAB 5 Handbook*. Springer, 1998.

[89] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186. ACM, 1994.

[90] E. S. Ristad and P. N. Yianilos. Learning string edit distance. In *Proc. 14th International Conference on Machine Learning*, pages 287–295. Morgan Kaufmann, 1997.

[91] F. J. Rohlf. A probabilistic minimum spanning tree algorithm. *Information Processing Letters*, 7:44–49, 1978.

[92] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable length. *Machine Learning*, 25:117–149, 1996.

[93] G. Rossi, D. Schwabe, and F. Lyardet. Improving Web information systems with navigational patterns. In *8th World Wide Web Conference*, pages 589–600, Toronto, Canada, May 1999.

[94] G. Salomon. Designing casual-use hypertext: The CHI'89 InfoBooth. In *Proceedings of CHI'90: Computer-Human Interface*, pages 451–458, 1990.

[95] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–979, 1991.

[96] G. Salton and M. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[97] M. Sanfaller. *TCP/IP und NFS in Theorie und Praxis, Unix in lokalen Netzwerken*. Addison-Wesley, 1990.

[98] A. Sanfeliu and K. S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 13(3):353–362, 1983.

[99] R. Sarukkai. Link prediction and path analysis using markov chains. *Computer Networks*, 33(1–6):377–386, 2000.

[100] H. Sauer. *Relationale Datenbanken : Theorie und Praxis*. Addison-Wesley-Longmann, 4 edition, 1998.

[101] L. G. Shapiro. Relational matching. In *Handbook of Pattern Recognition and Image Processing: Computer Vision*, pages 475–496, 1993.

[102] L. G. Shapiro and R. M. Haralick. Structural descriptions and inexact matching. In *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-3*, pages 504–519, 1981.

[103] R. Sibson. SLINK: An optimally efficient algorithm for the single link method. *Computer Journal*, 16:30–34, 1973.

[104] P. Sneath. The application of computers to taxonomy. *Journal of General Microbiology*, 17:201–226, 1957.

[105] T. Sokolofsky and C. Kale. *A TCP/IP Tutorial*. RFC 1180, 1991.

[106] W. Stanek. *Netscape Mozilla Source Code Guide*. Ventana Communications Group, 1999.

[107] D. Stanyer and R. Procter. Improving Web usability with the link lens. *Journal of Computer Networks and ISDN Systems*, 31:455–466, 1999.

[108] W. Tsai and K. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(12):757–768, 1979.

[109] J. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23(1):31–42, 1976.

[110] B. Underdahl. *Macromedia Flash MX: The Complete Reference*. Osborne McGraw-Hill, March 2002.

[111] W. Wallis, P. Shoubridge, M. Kraetz, and D. Ray. Graph distances using graph union. *Pattern Recognition Letters*, 22(6/7):701–704, 2001.

[112] J. H. Ward. Hierachical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58:236–244, 1963.

[113] H. Weinreich and W. Lamersdorf. Concepts for improved visualization of Web link attributes. *Computer Networks (Amsterdam, Netherlands: 1999)*, 33(1–6):403–416, 2000.

[114] A. Wexelblat and P. Maes. Footprints: History-rich tools for information foraging. In *Proceedings of the Conference on Human Factors in Computing Systems (CHI '99) (Pittsburgh, PA)*, pages 270–277. ACM Press, 1999.

[115] R. Widhalm and T. Mück. *Topic Maps*. Springer Verlag, 2002.

[116] E. Wong. Three-dimensional object recognition by attributed graphs. In H. Bunke and A.Sanfeliu, editors, *Systanctic and Structural Pattern recognition, Theory and Applications*, pages 381–414. World Scientific, 1990.

[117] O. R. Zaiane, M. Xin, and J. Han. Discovering web access patterns and trends by applying OLAP and data mining technology on web logs. In *Advances in Digital Libraries*, pages 19–29, 1998.

[118] O. Zamir and O. Etzioni. Grouper: A dynamic clustering interface to Web search results. *WWW8 / Computer Networks*, 31(11-16):1361–1374, 1999.

[119] I. Zukerman, D. Albrecht, A. Nicholson, and K. Doktor. Trading off granularity against complexity in predictive models for complex domains. In *Proceedings 6th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2000), Melbourne, Australia*, pages 241–251. Springer-Verlag, Sep 2000.

[120] I. Zukerman and D. W. Albrecht. Predictive statistical models for user modeling. *User Modeling and User-Adapted Interaction*, 11(1-2):5–18, 2001.

Acknowledgments

I am especially thankful to the following people for their help and support to write this thesis:

1. Mr. Prof. G. Sommer who supervised my research concerning this thesis,

2. Dr. J. Pauli and Prof. G. Weber for many helpful discussions and useful suggestions

3. Mrs. K.-B. Spindler, head of the 'Forschungsdezernat der Kieler Universität' for financial support

4. the members and former members of the Cognitive Systems group at Kiel university for friendly and solid cooperation, especially to S. Buchholz, V. Banarer and C. Perwass

5. H. Schmidt and G. Diesner who helped me with technical questions.