Diplomarbeit

# Visualization of Web Localities

Boris Diebold

Eberhard Karls
Universität
Tübingen

Diplomarbeit im Fach Informatik

# Visualization of Web Localities

vorgelegt von

Boris Diebold

April 2001

Betreuer: Prof. Dr. Michael Kaufmann

Wilhelm-Schickard-Institut
Arbeitsbereich Paralleles Rechnen
Eberhard-Karls-Universität Tübingen
Sand 13 · 72076 Tübingen

## Erklärung

Hiermit versichere ich an Eides Statt, diese Diplomarbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Ferner habe ich die Arbeit noch keinem anderen Prüfungsamt vorgelegt.

Tübingen, April 2001


## Acknowledgements

I would like to thank the following institutions and individuals for supporting this thesis by supplying Web server access log files:

Native Instruments Software Synthesis GmbH, Berlin
Simon Jarosch & StyleTec GbR, Tübingen
Ouk Magazine, Tübingen
yWorks GmbH, Saarbrücken
Lehrstuhl für Rechnerarchitektur der Universität Tübingen

I would also like to thank Prof. Dr. Michael Kaufmann for supervising this thesis, Markus Eiglsperger and Roland Wiese for providing invaluable advice on all kinds of problems and my labmates Keyan "Blast" Zahedi and Frank "Ebay" Eppinger for bearing all my stupid questions and creating an enjoyable work atmosphere. I am especially grateful to my parents who always encouraged and generously supported my studies and various other endeaveours in the last years. Most importantly, my sister and my brother-in-law helped me keep things in perspective.
And finally to Luzi for her endless patience, love and company.

# Contents


**1 Introduction** **7**

**2 Website Mapping** **13**
2.1 Foundations . . . . . 13
2.1.1 The World-Wide Web . . . . . 13
2.1.2 Graph Theory and Graph Drawing . . . . . 14
2.2 Site Maps revisited . . . . . 16
2.2.1 Automatic Generation of Sitemaps . . . . . 16
2.2.2 Sitemap Design Requirements . . . . . 17
2.3 Sitemap Generation Systems - State of the Art . . . . . 20
2.4 The *YWeb* Sitemap System . . . . . 26
2.4.1 Design Requirements . . . . . 26
2.4.2 Architecture . . . . . 27

**3 Structure Data** **29**
3.1 Structure Data Extraction . . . . . 29
3.1.1 Inter- and Intra-page Structure Extraction . . . . . 30
3.1.2 Global Connectivity Extraction . . . . . 32
3.2 Logic Domain Clustering . . . . . 33
3.2.1 Entry Page Ranking . . . . . 33
3.2.2 Domain Boundary Definition . . . . . 36
3.3 Related Work . . . . . 38

**4 Structure Visualization** **41**
4.1 Force-Directed Topology Layout . . . . . 41
4.2 2D Cone-Tree Layout . . . . . 43
4.3 Tree-Map Layout . . . . . 45
4.3.1 Visual Mapping . . . . . 48
4.3.2 Interactivity . . . . . 49
4.3.3 Web-based Structure Visualization . . . . . 49
4.4 Related Work . . . . . 51

**5 Usage Data** **53**
5.1 Data Sources for Usage Analysis . . . 53
5.2 Inferring User Paths from Web Server Logs . . . 55
5.2.1 Web server access log data . . . 55
5.2.2 Access Log Data Quality . . . 56
5.3 Path Extraction Algorithm . . . 58
5.3.1 Path Model . . . 59
5.3.2 Data Cleaning . . . 60
5.3.3 Session and User Identification . . . 61
5.3.4 Path Reconstruction . . . 62
5.3.5 Popular Path Extraction . . . 65
5.4 Related Work . . . 71

**6 Usage Visualization** **73**
6.1 Popular User Path Layout . . . 73
6.1.1 Width-Restricted Path Layout . . . 74
6.1.2 Visual Mapping . . . 77
6.1.3 Interactivity . . . 78
6.1.4 Node-Path Exploration Layout . . . 79
6.2 Web-based Path Visualization . . . 82
6.3 Related Work . . . 84

**7 Conclusions and Future Work** **87**

**A Web Locality Data Sets** **93**

**B Software Package Description** **97**
B.1 Required Software Packages . . . 97
B.2 Components . . . 98
B.2.1 *YWebCrawler* . . . 98
B.2.2 *YWebLog* . . . 99
B.2.3 *YWeb* . . . 100

**C Document Type Definitions** **105**

# Chapter 1

# Introduction

The Web has revolutionized our conception of communication and interaction. It offers new ways of business-to-business and business-to-customer transactions, new mechanisms for person-to-person communication and new means of discovering and obtaining information, services and products.
As the World-Wide Web is constantly evolving and embracing new technologies such as Flash, DHTML, XML, SMIL and other means of generating content, it has mutated into an extremely "messy" information space as stated by [Abr97]:

> "This medium is *overloaded* with information, *polluted* with redundant, erroneous and low quality information, it progresses toward disorder according to the principle of *entropy* and it has no aggregate *structure* which organizes distinct web localities. Furthermore, users have *no global view* of the entire WWW from which to forage for relevant pages."

Although browsers have become a framework for a variety of add-ons, the action of navigating the Web has not changed much since the first hypertext documents became available over a decade ago: scroll through a page, click on a link and jump to another page.

A variety of systems have been proposed to assist the user in navigating the hyperspace, most noticeably by providing alternative views of Web sites as a whole, also known as *sitemaps*.
As Web localities are subject to frequent change and redesign, it is especially important to provide a system for *automatic* generation of such sitemaps from various data sources instead of creating them manually. This also allows the application of up-to-date usage information in order to let the visitor take advantage of previous user navigations to distinguish possibly interesting content.

The subject of this thesis is the design and implementation of such an automated sitemap generation system along with the development of new techniques for clustering and visualization.

In order to provide a further basis and introduction to the field of Web site mapping, we now describe problems and properties inherent in Web navigation.

### Navigating and Searching the WWW

When navigating in a hypertext environment, people easily become lost while looking for information. The *lost in hyperspace syndrome* [Con87] has two main traits: *disorientation* and *cognitive overhead*. While disorientation refers to the tendency to lose one's sense of location and direction in a non-linear environment, the term cognitive overhead refers to the additional effort and concentration which is necessary to maintain several tasks at one time: People easily get caught up in trying to find their way and become totally unable to concentrate on the main task.
This is due to several important short-comings of the Web as we know it:

- *Lack of physical context:* The covers of a printed document provide a pysical context to what it contains. The viewer can see and feel boundaries and sense part-to-whole relationships of each portion of the page being viewed. Since a Web site is viewed one page at a time there is no sense of a start and end nor of a part-to-whole of a single page.

- *Lack of organizational paradigm:* A printed document has a clear order of presentation. Due to the inherent degrees of freedom in Web design, a clear order in a Web page is difficult to convey. In addition, there is a great dissimilarity among the possible organizational paradigms being used.

- *Lack of rhetoric of arrival:* The jumping metaphor used in hypertext navigation leaves the user with no context as where he came from. In addition, followed links can have very different properties (internal, site external, search engine created etc.)

- *Lack of knowledge of site structure:* Since browser can't visualize inter-page relationships, questions such as: "Where can I go from here ?", "Where am I ?" or "Which pages point to that page ?" can not be answered directly.

Because of the vast amount of information available on the Net and the limited access methods available, *information overload* causes people to drift into unintentional "web-surfing".
This is even amplified by the navigation delay experienced due to limited download speed.

A variety of solutions have been proposed to reduce the *lost in hyperspace (LOH) syndrome*. These can be categorized by the underlying *navigational paradigms* that the user follows while foraging for information (although most users use a combination of both):

*Search engines* (such as *Google*, *Altavista* etc.) typically use a global inverted index over document profiles, which enables retrieval of documents based on matching a user-issued query against the index.
If users follow the *search paradigm*, they navigate from the search engine to pages within the result set to locate information, returning when they failed and starting the search anew. Although widely used, this approach suffers from several shortcomings that have to be taken quite serious: As [LG99] reveals, search engines only cover about 16% of the publicly accessible Web as of 1999, with the figure being likely to decrease during the following years. In addition, they are heavily biased toward listing more commercial than educational content as well as overrepresenting US-based domains. As [LG99] put it:

> "The current state of search engines can be compared to a phone book which is updated irregularly, is biased toward listing more popular information and has most of the pages ripped out."

Besides being incomplete and biased, search engines suffer from non-standard interfaces and various usability problems.

The *browse paradigm*, where users navigate the document space by triggering links through visual features on the Web pages, is often characterized as a form of "hit and miss navigation": Locating information solely via this scheme tends to be quite tedious due to the LOH syndrome described above. Therefore, visual navigation aids such as *overview diagrams* that generate meaningful structures from a set of documents and visually represent them have already been suggested in the hypertext era. By providing alternative views of the hypertext topology, they reduce the complexity of the system and allow the user to employ a larger set of navigational strategies. Besides, topology based diagrams enable the user to generate an appropriate *mental map* of the hypertext and also provide visual clues to the underlying structure and the current location within it.

Overview diagrams or *sitemaps* as they are called when employed in context of the Web, have traditionally focused on the visualization of a single Web site. Recently, systems such as [TH98] have been presented, that operate on a higher level by visualizing *clans* of densely connected sites including the corresponding inter-site topology thus providing a limited global navigation facility. However, as [PL98] stated, the architecture of the WWW is so entrenched into todays systems that the problem of real, global navigation in the Web is unlikely to be solved in the near future. Therefore, Web site designers and administrators themselves are required to provide a navigable environment for the users who enter their domain.

The term *sitemap* as introduced above, is not well defined. Often, it is used for text-based structure descriptions of a site or for crude hand crafted diagrams that do not deserve to be called "map". Others employ it for every kind of visualization related to a Web site. The "Atlas of Cyber-Geography" [Dod00] presents a great wealth of such Web-related visualizations.

Taking into consideration the *timeframe* (longterm vs. shortterm), *purpose* (navigation vs. analysis), *preparation* (manual vs. automatic), *structure* (static vs. dynamic) and *audience* (site administrator vs. visitor), we distinguish between three different sitemap types:

### Analysis Sitemaps

The most obvious need for site mapping probably originates from the domain of site analysis. By comparing traffic patterns and existing link structure, the administrator may gain valuable insight into user behaviour and preferences that enables him to optimize the site structure and establish user profiles. Nearly all customer relationship management (CRM) solutions for e-business (such as *Accrue's Insight*[1]) include a simple form of analysis sitemap although they are more concerned with using the collected data together with the order database for data mining.

### Surfmaps

Surfmaps such as *Y-Nansen* [Gal00] act as visual browsing histories that are often augmented by extended bookmarking capabilities. Unfortunately most surfmaps require the user to install special software on his computer to

[1] `http://www.accrue.com`

record his path through the Web. This might explain the fact that their use is currently not very wide-spread.

### Navigation Sitemaps

Navigation sitemaps such as overview diagrams described above are also implemented for the visitor of a particular site in order to facilitate access to possibly interesting content. By supplying a (perhaps interactive) navigation map, the user will be enabled to locate the desired information a lot faster than by pure browsing and is therefore less inclined to switch to another site. The requirements for the design of such maps are presented in 2.2.2. Since this thesis is focused on the automatic generation of navigation sitemaps, we will use the term sitemap as a synonym for navigation sitemap from now on.

[PL98] states that navigation sitemaps may have the positive effect of reducing disorientation by improving spacial context and providing a sense of extent of a particular Web site. They also may act as a visual surrogate of the user's short term memory.

On the other hand, maps often don't provide inherent clues to their navigational nature and are easily used as "bubble-gum" to fix poor site design. In practice, speed, complexity and maintenance issues as well as notorious time constraints hinder many administrator from implementing such sitemaps.

### Motivation

The purpose of this thesis is to demonstrate the feasability of automated generation of navigation sitemaps via a prototype. The system which also serves as a basis for further research introduces two novel navigation map approaches. A special focus is the incorporation of usage data in the form of access log files in order to provide an alternate view of the locality.
This work touches many diverse areas like graph drawing, human-computer interaction, information visualization and computer networks.

This thesis is organized as follows:

In Chapter 2 we first review important terminology of Web-related topics and graphs in general. In addition, the process of automatic generation of Web site maps is studied followed by important design requirements for such maps. 2.3 presents some state of the art systems for automatic site map generation which is followed by an architectural overview and the design requirements of the *YWeb* system proposed by us (in 2.4).

Chapter 3 presents the components for extracting structural data of a Web locality followed by a clustering technique for structure data that identifies logical sub-domains in a site.

In Chapter 4, we present several visualization techniques for the structural Web site data and a Web-based implementation of a visualization for the logical domains.

Chapter 5 focuses on the extraction and reconstruction of meaningful user traversal path through the respective site.

In Chapter 6, we propose an interactive visualization of the locality based on the extracted user paths and present a Web-based implementation for this approach.

Finally, Chapter 7 evaluates the application of the system to several Web localities and presents problems and further enhancements possible in this context. In addition, we also give an outlook on possibly interesting research activities in the Web mapping field.

# Chapter 2

# Website Mapping

In this chapter we first review important terminology of Web-related topics and graphs in general. Later on, the process of automatic generation of Web site maps is studied followed by important design requirements for navigation maps. A state of the art review of sitemap generation systems provides the basis to deduct important design requirements for our own system that is then described on an architectural level.

## 2.1 Foundations

In this section,we first review important terminology of Web-related topics and graphs in general that is used throughout the thesis.

### 2.1.1 The World-Wide Web

The World-Wide Web as we know it today, is the outcome of *hypertext* introduced in the 1980s that relies on the basic idea of free, untyped connections of associated material. The Web which can be seen as a huge, distributed hypertext, enhanced the concept by allowing links across documents and sites that do not have to be limited on a single topic. In addition to being constantly evolving, the Web is known as being highly redundant and incomplete. As stated before, the concept of the Web does not inherently provide any kind of structural information (except for the file-system folder structure represented in the URLs - therefore often used by users for orientation and navigation).

In order to clarify some often used, Web-related terms, we introduce the following formal definitions:

**Definition 1 (Web Page)** *A Web page p is a single HTML text document that may contain several embedded links and media types (pictures, sound, applets etc.). It is uniquely defined by its Uniform Resource Locator (URL)* [1]*, which consists of the access protocol, the domain name, an URL path and the page filename.*

The HTML source of a Web page contains various tags (such as `<A>`, `<MAP>` or `<FRAME>`) that link to other content. These uni-directional *hyperlinks* either point to anchors within the current Web page, to pages on the same or different site and to server programs or embedded media types. Besides, links can also be coded in non-HTML format like Javascript, PDF, Shockwave, applets etc.

**Definition 2 (Web Site or Web Locality)** *A Web locality or Web site is a set of Web pages* $W = (p_1, \ldots, p_n)$ *defined by the same host domain name or IP address in the respective URLs.*

This definition assigns all pages available on `http://www.mp3.com` to the same Web locality, while the pages available via `http://artists.mp3.com` are considered to belong to another Web site. The Web pages and embedded media are delivered to the user's browser from the Web server via the *hypertext transfer protocol* (HTTP)[2] .
Web sites or subsections of sites can be categorized by different *genres* that include shops, portals, discussion boards, communities, faqs etc. We also distinguish between small ($|W| < 1000$), medium ($1000 < |W| < 10000$) and large ($|W| > 10000$) Web localities.

According to [LG99], the publicly indexable Web contained an estimated 800 million pages as of february 1999, encompassing about 15 terabytes of information, not including page-embedded media ! Due to the low metadata use - only 34% of the pages use the simple "keywords" or "description" metatags and 0.3% comply to the *Dublin Core metadata standard* - the vision of the *semantic Web*[3] is far from becoming reality.

### 2.1.2 Graph Theory and Graph Drawing

In the last few years, graph theoretic methods already employed in diverse areas such as sociology, physics, communication science etc. have been applied to the Web. This is due to the fact that *graph theory* [Har69] serves as a mathematical model for any system involving a binary relation.

[1] See RFC1738 at `http://www.w3c.org/Addressing/1738.txt`
[2] See `http://www.ics.uci.edu/pub/ietf/http/rfc1945.txt`
[3] `http://www.w3.org/2001/sw/`

The following section introduces definitions for some important concepts related to this field that will be used throughout this thesis:

**Definition 3 (Directed Graph or Digraph)** *A digraph $G = (V, E)$ consists of a finite set of vertices $V$ and a finite set of directed edges $E$ with $e = (u, v) \neq (v, u) \in E$ and $u, v \in V$.*

**Definition 4 (Path)** *A path $p$ of length $k$ from $u$ to $u'$ is the sequence $< v_o, \ldots, v_k >$ of vertices such that $u = v_0$, $u' = v_k$ and $(v_{i-1}, v_i) \in E$. The length $k$ is the number of edges in the path. If $u'$ is reachable from $u$ via a path $p$, this is denoted by $u \overset{p}{\rightsquigarrow} u'$.*

In a directed graph, a path $< v_o, \ldots, v_k >$ forms a *cycle* if $v_0 = v_k$ and $k \geq 1$. A cycle of length $k = 1$ is called a *self-loop* whereas a graph with no cycles is said to be *acyclic*. In a *connected* graph, each pair of vertices is connected by a path.

**Definition 5 (Tree)** *A tree is a connected, acyclic, undirected graph with the root being a distinguished vertex $v \in V$.*

*Graph drawing* [BETT99] addresses the problem of constructing geometric representations or diagrams of graphs. The automatic generation of drawings of graphs has numerous applications in diverse areas like software engineering (UML and data flow diagrams), real-time systems (petri nets), chemistry (molecule drawings) and cartography (map schematics).
Usually, vertices are represented by points or boxes and edges as simple curves. As a graph has infinitely many different diagrams, the usefulness of the drawing depends on its "readibility" which can be expressed by different aesthetic criterias such as the display of symmetries, the minimization of edge crossings or planarity. Thus, many graph drawing problems can be formulated as multi-objective optimization problems (which tend to be NP-hard) that make several trade-offs necessary.

Naturally, the Web itself can be represented as a directed graph in many different ways. The most straightforward representation assumes that the graph contains a node for each page $u$ and a directed edge $(u, v)$ if the page $u$ contains a hyperlink to page $v$. As [Hen00] points out, graph-theoretic methods have been applied to various Web-related areas like information retrieval, Web mining, navigation and visualization. A special property of "web-graphs" is that they are mostly *sparse*: The number of edges $e$ in relation to the nodes $n$ in the graph is given by $|E| = O(n \log_n)$.

## 2.2 Site Maps revisited

The following section explores the automatic sitemap generation process as a whole while identifying key aspects and inferring design requirements for navigation sitemaps.

### 2.2.1 Automatic Generation of Sitemaps

Sitemaps that serve as visual navigation aids by acting as overview diagrams are usually generated using the following scheme: First of all, Web-related data is collected which is then processed to extract meaningful structures that are used in the last step for generating the sitemap diagram. We now explain the individual steps in more detail.
Navigation sitemaps can be generated from a variety of Web-related data sources. The widely accepted *CUT model* distinguishes between:

1. *Content:* The content refers to the *real* data on the respective Web pages which usually consists of text and graphics but can contain any other media as well. This type of data can be used to compute page similarities or generate clusters of pages of the same topic by using text or image classification.

2. *Usage:* Usage data captures the patterns of usage of Web sites via IP addresses, page references, time and date of access and so on. It can be used to calculate popularity metrics for Web pages, clusters of pages or reconstruct user paths through the locality.

3. *Topology:* Topology data characterizes the organization of the locality via inter-page relationships such as hyperlinks and intra-page structure such as the HTML tag occurrence. This data type is often used for clustering and hierarchy induction.

One of the main challenges in visualizing Web localities is not only the useful combination of the data sources described above, but the handling of the vast amount of available data (access data from Web server log files is usually in the GB/month range!).
In his survey on visualization of hypermedia, [Muk99] identified two main data simplification methods that are widely used:

a) *Filtering* refers to the omission of possibly irrelevant data. Dynamic filtering methods are often used to inspect a data set at different levels of granularity.

b) *Abstraction* in constrast, refers to the creation of new meta-level structures from the original data for visualization and interaction. In general, *clustering* is used to group objects that are similar according to a certain metric, while *hierarchization* exploits the relationships between objects to induce a kind of parent-child relationship between them.

After the data simplification step, the main task is the automatic generation of a diagram from the resulting data that is both visually appealing and easy to navigate. These requirements are classical topics in the area of *information visualization* defined as "the use of computer-supported, interactive, visual representations of abstract data to amplify cognition" [CMS99].
In their survey, [HGM00] identified important differences of diagrams in information visualization as opposed to traditional graph drawing: First of all, planarity is not a central issue. More important is in fact that the layout should be interactive, which rules out most algorithms used in graph drawing because they are computationally too expensive. Because of the repeated diagram generation required for interactivity, the results of the employed algorithms also have to be predictable (which is not the case with many randomized methods that produce a slightly different layout every time) in order to preserve the user's *mental map* of the data.

The choice of the *visual metaphor* used for the diagram is particular important, since it supports the user's perceptual reasoning. Traditionally, the *node and link metaphor* has been used to visualize graph-like structures. Other metaphors include the *landscape*, *bullseye* or *auditorium* (see 4.4). Section 2.2.2 summarizes and enhances the design requirements for good navigation sitemaps.

[HGM00] especially discourages the use of 3D visualization techniques in information visualization since these have significant problems which can be attributed to the inherent cognitive difficulties with navigating in a three dimensional space, such as the discrepancy of using a 2D screen and 2D input to interact with a 3D world and the missing motion and stereo cues.

Last but not least, the technical aspects of the sitemap's implementation should also be considered since it has to be able to scale for a large number of simultaneous users and has to be sufficiently fast in order to create a real added value for the user (otherwise, he may just leave the site and search somewhere else).

### 2.2.2 Sitemap Design Requirements

Taking into consideration the process described in section 2.2.1, we can now formulate important design requirements and properties for naviga-

tion sitemaps (some already stated by [PL98]) that should be kept in mind while choosing an appropriate visual metaphor and implementing layout algorithms:

*Landmarks.* Research on human wayfinding suggested that navigation starts with an orientation stage where the user attempts to recognize his location within the Web locality. Therefore, landmarks are defined as nodes that provide salient qualities. Although [MM97] empirically determined that structural landmarks are in fact poor predictors of behavioural landmarks, in our opinion the right set of landmarks might improve navigation performance by providing "navigational hubs" for the users.

*Hierarchical Abstraction.* Hierarchical representations are useful in organizing information and reducing the number of alternatives that have to be considered during navigation. It has been shown that even in the most random topology, users tend to impose a hierarchical structure in order to make sense of the presented data. In their survey, [MM97] also revealed that strongly hierarchical sites are more usable since they are perceived as smaller then weakly hierarchical ones (given the same total size) and nodes up in the hierarchy tend to be more memorable. By providing a hierarchical abstraction of the site, the map assists the user in constructing an appropriate mental model and hence improves his ability to interact with the system.

*Dynamic Filtering.* Filtering tools applied to site maps may allow the user to control the level of detail, thus providing a scalable representation of the site that can be adjusted to the current information need.

*Multi-Attribute Representation.* A Web site can be seen as a large and complex data set with the basic unit of storage being a single Web page. Since each page as well as the related structures have a variety of different properties and attributes, an appropriate multi-attribute presentation provides the user with additional insights into the examined data. Particular interesting is the use of multiple attributes in a single diagram that allows users not only to locate extreme values, but also to discover patterns and relationships in the data.

*Context & Local Views.* In order to maintain the user's sense of orientation, both the local navigation task of moving between pairs of specific nodes (as provided by Web browsers) as well as the global navigation task which involves movements between several nodes should be supported. By providing hierarchical expand and contract features as an example, the user can obtain

a sense of the extent of the system without getting too much detail, while still having the possibility to "drill-down" for more information.

*Answer User's Questions.* A good diagram should answer popular questions of the average user: "Where am I ?", "Where have I been ?", "How did I got here ?", "How many other users have been here ?" and "Where can I go from here and what options are interesting for me in particular ?"

*Ease of Use.* Particular important for the acceptance of sitemaps as navigation aids is their inherent ease of use. The diagram and visual metaphor should be easy to grasp by the occasional visitor and the navigational features and interface should be designed as not to overwhelm the user. Therefore, we suggest a minimalistic approach with regard to these topics since users foraging for information on the Web are known to quickly leave a site and move on to the next if they do not catch the "information scent" (as [Pit98] discovered, the average number of page requests per Web site by a single user is merely 3 with a mean reading time of 30 seconds per page !).

*Compact View.* Navigation sitemaps, will most probably be displayed either in or alongside the Web browser. This dramatically restricts the screen estate available for the diagram and requires the use of space-constrained layout algorithms if we don't want to use zoom and pan navigation techniques. In practice, we suggest using a view of about $400 \times 400$ pixels for the map display.

Further requirements are not related to the diagram itself but comprise the technical aspects of the sitemap implementation such as:

*No client-side Installation.* Since users generally visit a large number of different Web-sites with distinctive sitemap implementations, it is essential that the sitemap itself is completely Web-based and requires no special client-side installation on the user's machine. In addition, special care has to be taken about start-up times and network bandwith required by the sitemap implementation.

*Scalability.* As stated before, the sitemap implementation should be highly scalable in terms of server load and network bandwith in order to support a large number of simultaneous accesses by visitors of the Web locality.

*Ease of Deployment.* The generated sitemaps should be especially easy to integrate in the existing Web locality without requiring major changes to the server configuration or the existing HTML code.

## 2.3 Sitemap Generation Systems - State of the Art

Although, there exists a large variety of navigation sitemaps (see [Dod00]), most of them can be classified as research prototypes that have mostly been tested on a single site, do not provide all functionality required for a real-world automatic sitemap generation system or are not available to the public (such as *Footprints* [WM97] or *SiteTree* [PL98] for example).

In the following system review, we therefore focus on more or less refined, applicable navigation sitemap generation systems and turn our attention to the system architecture as a whole as well as to the specific properties of the resulting sitemaps.

### PowerMapper

*PowerMapper*[4] from *Electrum Ltd.* is probably the most simple and straightforward system available. The application which is available for Windows platforms in a price range from US $39-$99 includes a crawler module that extracts the Web site topology as well as five sitemap generation modules: 3D buttons, page clouds, text-based table of contents, isometric map and tree view. Depending on the visualization, pages are represented by thumbnails of the actual Web page, which are expandable for exploration down the hierarchy. The map itself (which can be edited before generation, see Figure 2.1) is realized via several HTML and GIF files which form a client-side image map that can easily be integrated into the existing site by adding a single hyperlink.

### SiteBrain

*SiteBrain*[5] is an interesting approach to the site navigation problem. By applying their patented user interface (which consists of "a graphical representation of thoughts in a hyperlinked matrix") based on "thoughts" to the Web, *The Brain LLC* provides a simple, yet elegant navigation solution that nicely animates transition between pages via zoom, rotation etc. of the nodes and edges. "Thoughts" can be generated automatically from a site via

[4] Available from `http://www.electrum.co.uk/mapper/`

[5] Available from `http://www.thebrain.com`

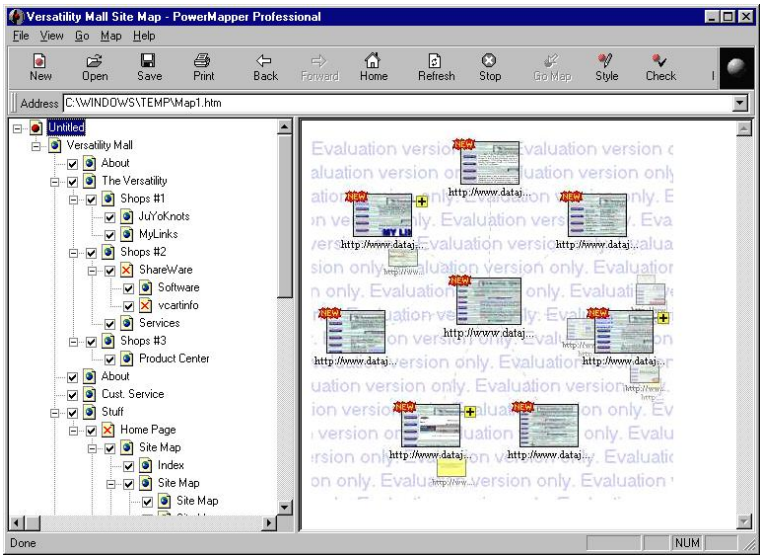


Figure 2.1: PowerMapper

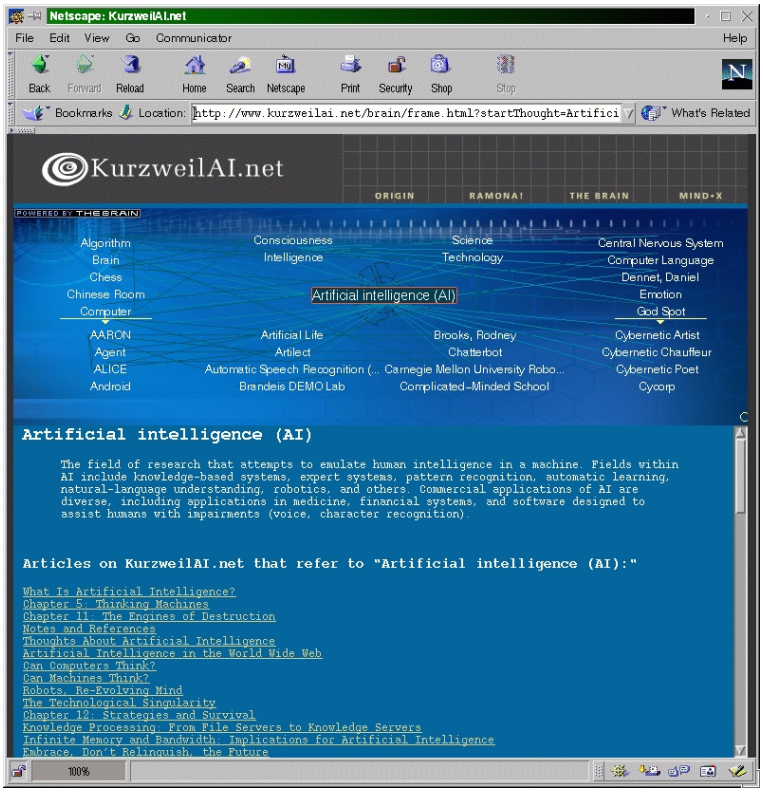


Figure 2.2: *SiteBrain* sitemap integrated in a Web page

a crawler or generated by hand or from existing "personal brains". The map itself which also enables the visualization of complex webs and relationships is realized as a highly configurable Java 1.1 applet that fits nicely into an already existing Web page (see Figure 2.2). The price for the Windows-based product ranges from US $295 (100 thoughts) to US $1995 and above (with the price based on the number of thoughts).

## Tree Studio

*Tree Studio*[6] which was formerly known as *Site Lens Studio* is a high-end site map solution provided by *Inxight*, a Xerox PARC spin-off. The simplest version of the Windows-based system is available for US $4000, is highly configurable and has automatic publishing features. Their patented visualization which is implemented as a Java 1.1 applet is based on the *hyperbolic tree* [LR96] that uses non-euclidian geometry to layout very large trees uniformly by first mapping the hierarchy onto a hyperbolic plane and then projecting it onto a circular display region (see Figure 2.3). Since the circumference of a circle on the hyperbolic plane grows exponentially with its radius and hierarchies tend to do so also, they can be laid out in a uniform way. The model provides a kind of fisheye distortion that supports smooth blending between focus and context nodes as well as animation for transitions between nodes.

## WebToc

Although still in prototype state, *WebToc*[7] offers a simple, yet interesting approach for the visualization of large hierarchies and Web sites. The sitemap which is displayed via frames in the same browser window as the explored site, is a Java applet generated by a crawl from the specific Web locality which is also done via a Java based software. The visualization employs an expand/contract table (similar to a table of contents) that provides graphical information indicating the number of elements in branches of the hierarchy as well as individual and cumulative sizes (see Figure 2.4). The intuitive interface extensively uses color cues and value bars in order to represent attributes such as file types and provide a quick overview capability.

[6] Available from `http://www.inxight.com`
[7] Available from `http://www.cs.umd.edu/hcil/webtoc/`

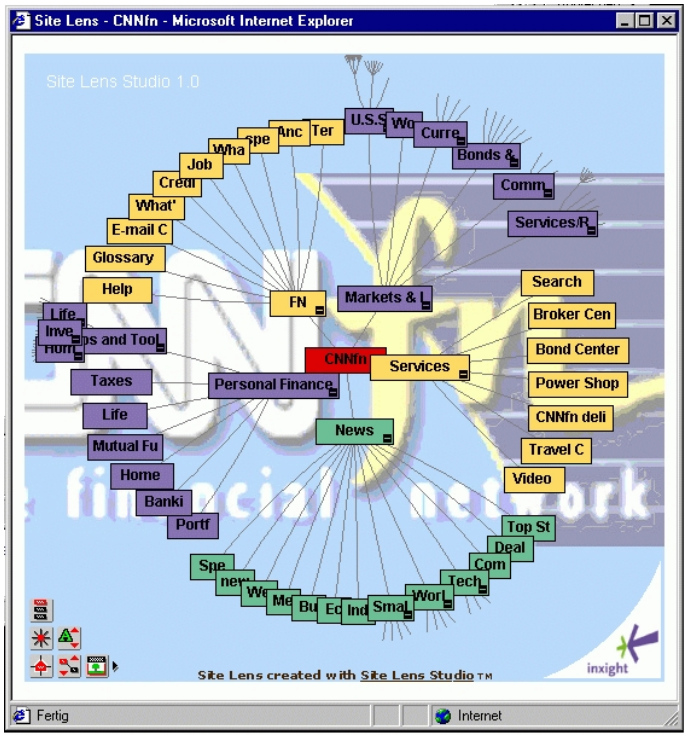


Figure 2.3: Sitemap generated by *Inxight Tree Studio*

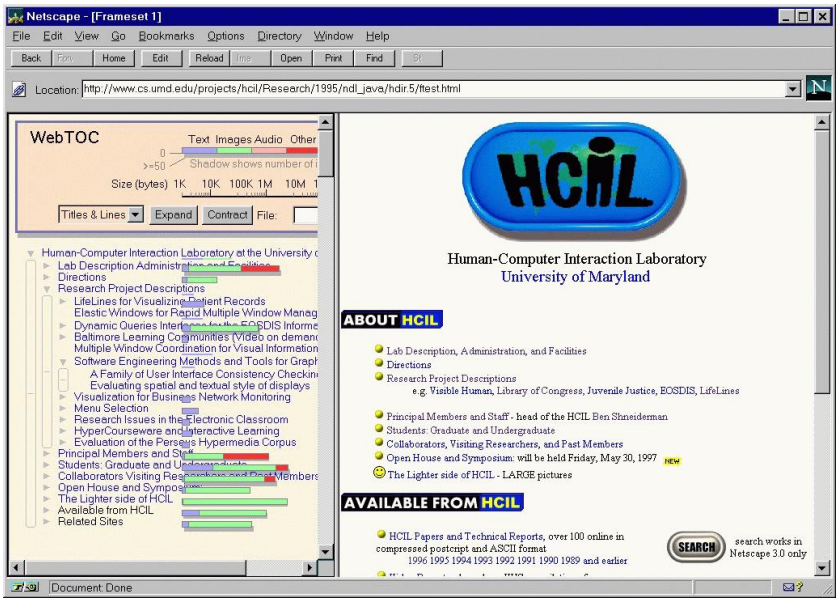


Figure 2.4: Sitemap generated by *WebToc*

## Mapa

Although not a refined sitemap generation system, *Mapa* from *Dynamic Diagrams*[8] is quite interesting due to the *landscape visual metaphor* being employed. A crawler first generates a hierarchy from the Web locality by applying several learned and user-defined inference rules and heuristics. The

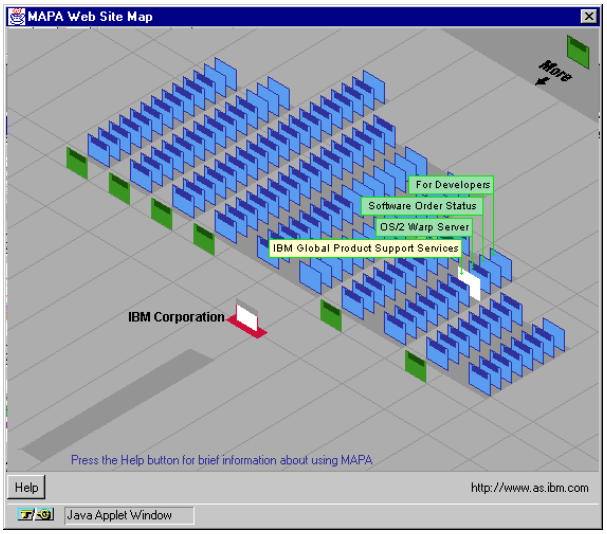


Figure 2.5: Sitemap generated by *Mapa*

applet-based visualization supplies an interactive *z-factor isometric map* that provides a 2.5D perspective view of the hierarchies with single Web pages depicted as flat 2D rectangles (see Figure 2.5). It uses the placement in virtual space to indicate logical connections between pages with the navigation being very appealing due to extensive animation capabilities.
Unfortunately, *Dynamic Diagrams* has recently been acquired by a larger company and *Mapa* is not available to the public anymore.

[8] `http://www.dynamicdiagrams.com`

## Evaluation

While evaluating a large number of sitemap systems (including the ones previously described), it became obvious that most systems actually focus on a single form of visualization and data simplification procedure. In addition, the application of usage and content data as a data source for representing the Web locality seems to be lacking, except for highly experimental systems since nearly all approches focus on exploiting the site's inter-page structure and topology. Besides, all applications tend to be proprietary and are not available cross-platform.

In his survey on information visualization in hypermedia systems, [Muk99] indicates that the discovery of new metaphors for navigation and visualization is vital for advances in the field of navigation sitemaps since none of the available systems today proved to be really useful after extensive usability studies. He also mentioned, that it is important to make the systems usable by common computer users in order to gain more acceptance.

[Muk99] also identifies the creation of information visualization systems that provide a library of effective data simplification and visualization techniques which can be combined to generate a representation of the underlying Web locality as a key challenge in this area. This also includes the application of standard data exchange formats.

In consideration of the facts presented above, we propose *YWeb* (see 2.4) as an open, extensible, easy-to-use sitemap generation system for small to middle-sized Web localities that provides different visualization and data simplification methods and especially focuses on exploiting the available usage data.

## 2.4 The *YWeb* Sitemap System

As stated before, the purpose of this thesis is to present a prototype system for the automatic generation of navigation sitemaps that produces appealing, interactive, Web-based sitemaps that can be used by visitors of a specific Web site. In addition, new forms of visualization of structure and usage related Web site data should be investigated.
The target group of such a system are administrators of small to middle-sized sites that obviously (see [Ver00]) have not enough time to spend on hand-crafted, elaborate navigation maps and want to give their Web localities an added value by providing sitemaps to their users.

In the following section, further design requirements (in addition to the ones established for the sitemaps itself in 2.2.2) for such a system are formulated, before describing the architecture in depth.

### 2.4.1 Design Requirements

From the evaluation of the existing sitemap systems in section 2.3, we can establish some important design requirements for an automated sitemap generation system.

1. *Cross-platform*: Naturally, the system should be available cross-plattform in order to reach a wide audience.

2. *Interactivity*: Apart from supporting automated sitemap generation, the system should be highly interactive on the administator as well as on the user side. This includes the configurability of the various sitemap generation methods.

3. *Ease of Use*: The sitemap system as well as the the resulting navigation maps should be designed to be easy to use.

4. *Robustness*: The system should be robust in a way that it accepts a large variety of input data and is not restricted to a certain application domain or site type.

5. *Independence*: Certainly, the system should be independent from Web authoring systems or server extensions, by operating on the raw data available on most Web servers.

6. *Extensibility*: In order to promote extensibility and interoperability, XML should be used for data storage and the system should be highly modular.

7. *Pleasant Visualization*: The sitemap visualization as well as the preview visualization for the site administrator should be visually appealing.

8. *Speed*: Last but not least, speed is an issue, since the sitemap generation should be available on common machines types.

### 2.4.2 Architecture

The *YWeb sitemap system* follows a *three-tier architecture* (see Figure 2.6) that can be characterized as follows.

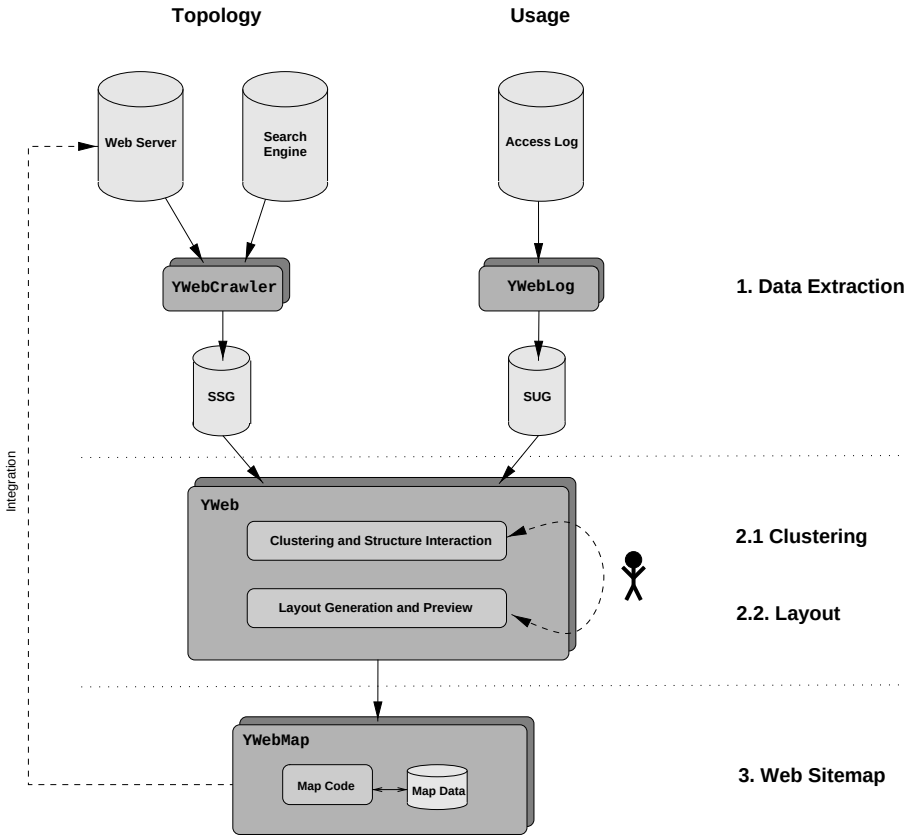


Figure 2.6: *YWeb* sitemap system architecture

The *first tier (data extraction)* is concerned with gathering the raw data from various sources on the Web and extracting relevant information that is usable for sitemap generation. Since this is a time-consuming process, the data is stored in an intermediate format that is later reused in the second tier. We distinguish between two different types of data:

*Structural data* is extracted from the Web server by crawling the Web pages and from specific search engines by submitting special queries. This is done by the *YWebCrawler* module that is also described in 3.1 and B.2.1. Upon successful extraction, the data is stored in a XML *site structure graph (SSG)* (see Appendix C).
*Usage data* is extracted from the server generated access logs by reconstructing user traversal paths. This is done by the *YWebLog* component (see 5.3 and B.2.2) that stores the result as a XML *site usage graph (SUG)* (see Appendix C).

The *second tier (clustering and layout generation)* is handled by the *YWeb* component. It provides means for clustering the extracted data and selecting different sitemap layout styles. The site administrator can therefore generate a pleasant visualization via an interaction loop with the two main sub-modules. The component also provides a one-click export of the specific sitemap as a Web-based visualization (which forms the next tier). Two clustering methods and corresponding visualizations have been implemented. The first method focusses on exploiting the hierarchies inherent in the URL paths to determine logical subdomains in the Web locality, while the second reconstructs common user traversal paths from the access log data that are further used to provide a more usage-based site view. Although the required input for both methods is complementary (with the first one requiring structure and the second usage data), each method will take advantage of further data if available.

The *third tier (Web-based sitemap)* is the sitemap component as it is presented to the visitor. Two different types of maps have been implemented: client-side image maps and Java applets. The map component itself is self-contained: it includes all the code and data necessary to provide a Web-based visualization without the first two tiers and can thus be integrated as a stand-alone component into the Web locality.

The *YWeb* system itself is implemented in *Java*, using the *y-files* framework (see B.1) that provides important features for viewing, editing, layouting and animating graph-like structures.

# Chapter 3

# Structure Data

In general, the structure data available on Web sites can be classified into *inter-page* and *intra-page* data. The inter-page structure captures the topology of the site via the links between pages whereas the intra-page structure refers to data associated with a single HTML page such as meta tags, size, date etc. Since the raw structure data is too large to be presented as a whole, clustering and abstraction has to be applied in order to extract meaningful patterns that can be used for exploration and visualization.

In this chapter, we present an extraction method for inter-page and intra-page data and a clustering technique based on finding logical domains in a Web page that serves as basis for hierarchical visualization.

## 3.1 Structure Data Extraction

In order to gain access to a site's structure data, a simple Web crawler was implemented that traverses the specified Web site from a defined starting page in breadth-first order.
Although there is a variety of free web-crawling robots available, most of them need special libraries and are tied to a specific operating system. Thus, the decision was made to use Java's extensive networking library to extract the data and save it as a *site structure graph* for further usage:

**Definition 6 (Site Structure Graph)** *We define the structure graph of a Web site $W$ as the directed graph (which may include multi-edges) $G_S = (V, E)$ where each $v \in V$ represents a single Web page $p$ and holds its extracted properties like URL, size, tags, date etc. Each edge $e = (u, v) \in E$ represents a hyperlink in page $u$ that points to page $v$ and also contains link properties like link title etc.*

Since the URL of a respective node or page is used for lookup and identification, we often use URL and node interchangeably to denote a node in a graph or path that represents a single URL-identified HTML page.

Ambiguities in the URL to file mapping (e.g. different URLs pointing to the same physical file) can occur because of soft-links in the file system: In addition, the URLs `http://www.test.de/test/index.html`, `http://www.test.de/test/` and `http://www.test.de/test` all refer to the same file on nearly all Web servers configurations. In order to prevent the crawler from introducing duplicate nodes, we convert each URL that is encountered into a *canonical URL format*:

**Definition 7 (Canonical URL)** *The canonical URL for a file that can be accessed under the following URLs: (1)* `/dir/index.html`*, (2)* `/dir` *or (3)* `/dir/` *is defined by (3).*

Another delicate subject regarding the raw structure data extraction is the handling of HTML frames. If a Web site uses frames, there is a difference between the links the user can follow and links in the page that the user actually selected. Since several HTML pages can be reused by different frame-sets in a Web page, there's no clear notion of which page "belongs" to which frame-set. In fact, the intensive use of frames can produce disorientation on the user side. Thus the use of HTML frames is discoured in the W3C web content accessability guidelines[1]. Nevertheless, since still a lot of sites make heavy use of frames (especially for online documentations generated with `javadoc` and alike) we decided to treat frame-sets as normal pages in the resulting structure graph.

### 3.1.1 Inter- and Intra-page Structure Extraction

A typical crawler (see [HN99]) consists of three different modules: The *URL frontier* holds and schedules the extracted URLs to be visited, the *download component* is responsible for fetching the page specified by the URL and the *extraction component* analysis and extracts the data and URLs from the downloaded page.

As we sequentially process each Web page $p \in W$ (starting with the main index page `/index.html`), the following information is extracted and stored in the respective node in the structure graph $G_S$: `URL`, `content size`, `content type` and `date` can be directly obtained through the HTTP header. The value of specific HTML tags like `<Title>` and `<Meta>` is extracted by parsing the content of the retrieved HTML page. In order to jump from $p$

[1] `http://www.w3.org/TR/WCAG10/`

to the next page in the Web site, `<A>`, `<Frame>` and `<Map>` tags are parsed for valid URLs which are constructed from the document base and converted into the canonical format if necessary. Each URL encountered, that has not yet been visited is then added to the URL frontier queue $U$ for BFS-like traversal. After the whole site has been traversed, the resulting structure graph is serialized to disk using a XML format (see Appendix C). Algorithm 1 describes the crawler actions in pseudo-code.

**Algorithm 1:** Web Crawler

**Data** : Start page URL $\mathcal{U}$
**Result**: Site structure graph $G_S$

$G_S = \emptyset$;
$U \leftarrow \{\mathcal{U}\}$;
**while** $U \neq \emptyset$ **do**
  $v \leftarrow$`firstElem(`$U$`)`;
  $p \leftarrow$ fetch URL $v$ from server via HTTP;
  extract HTTP header information;
  **if** *file type* = *"text/html"* **then**
    add new node $v$ to $G_S$;
    add new edge $e = (u, v)$ to $G_S$;
    parse tag information from $p$;
    store extracted information in $u$ and $e$;
    **for** *all URLs u found in p* **do**
      **if** $u \notin G_S$ **then** $U \leftarrow U \cup u$;
    **end**
  **end**
**end**

The *YWebCrawler* implemented for the purpose of structure extraction is also further described in B.2.1.

Although the main crawling algorithm is quite trivial, a considerable amount of time was spent in making the crawler robust enough for daily use. In particular, corrupt HTML code and lazy, non-conformant protocol implementations that are accepted by most Web browser made up a big source for special cases that had to be considered and tested. Besides, special care had to be taken to support network timeouts and allow crawling of pages available via the secure HTTP protocol (HTTPS, see B.1).

Pitfalls for the crawler that were not considered in the current implementation but deserve to be mentioned include:

*URL-embedded session identifiers* (see 5.1) are used by a number of sites for visitor tracking. Unfortunately, there's no way to automatically tell url-embedded session identifiers from dynamic HTML URLs.
*Crawler traps* are a set of URLs that cause the crawler to cycle forever (intentional or unintentional), caused by scripts or symbolic links.
*Forms and interactive content* as well as excessive use of scripting languages such as Javascript also pose a considerable challenge for automatic Web crawlers due to the required interactivity.

### 3.1.2 Global Connectivity Extraction

Another source for interesting topology information is provided by the link databases of major search engines (such as *Google*[2] or Altavista[3]). By querying the database for nodes that link to the specified page (replacing the common search string by `link:` and the respective URL), we can gain valuable insight about the gobal topology regarding the neighborhood of our Web locality.

In our implementation, we use the *Google* search engine to query incoming external link structure for every node $n \in G_S$ after the site structure graph has been constructed. The returned page for each node contains a list of incoming links to the query URL sorted by Google's ranking procedure (see [Hen00]). The result page is then parsed for a maximum user-defined number of in-links that are added to the structure graph $G_S$.

[2] `http://www.google.com`
[3] `http://www.altavista.com`

## 3.2 Logic Domain Clustering

The graphs obtained by the structure data extraction are usually fairly large, with the number of nodes ranging from several hundred to several thousands. In order to obtain structures suitable for visualization, the *logical domain clustering* that was first introduced by [LKVT00] for the organization of Web query results is used.

**Definition 8 (Logical Domain)** *A logical domain $D = (p_1, \ldots, p_n)$ with $p_i \in W$ is a set of Web pages defined by a specific semantic relation and a relating, syntactic structure. Every logical domain has a single domain entry page $e \in D$ which is meant to be the first page to be visited by users navigating that domain.*

As opposed to *physical domains* (see Definition 2) which are organized by domain name, logical domains are organized based on functionalities such as project, seminar, home page etc. Therefore, a logical domain can also be seen as a set of Web pages in a physical domain which as a whole provides a particular function or is self-contained as an atomic information unit.

Examples for logical domains are quite common: a university department may contain a logical domain that holds information about ongoing research. Other domains may be formed by the department's user home pages or online-manuals where the level of granularity specifies the total number of logical domains.

The logical domain clustering can be decomposed into two phases which will be described in the further sections. First, it identifies logical domain entry page candidates using link structure, URL path information, document meta-data and citation. The second phase then defines the boundary of each logical domain and merges specific sub-domains.

### 3.2.1 Entry Page Ranking

In order to better understand the properties of logical domains, we identify the following functions and give some examples:

*Personal site:* Personal home pages are usually located within a physical domain. `http://www-pr.informatik.uni-tuebingen.de/~eiglsper/` for example, is the entry page for a personal Web site which by itself is independent. Thus they are treated as logical domains.

*Topic site:* Web pages related to a particular topic are usually grouped under a directory. This includes product showcases, online manuals, project Web sites etc. For example, `http://www-pr.informatik.uni-tuebingen.de/yfiles/` and `http://www-pr.informatik.uni-tuebingen.de/Lehre/` can be treated as seperate logical domains.

*Popular Site:* It may occur that a page in a domain is more popular than the entry page of the respective domain. Such a popular page is usually indicated by a large number of external incoming links (which can be regarded as a form of citation) and thus serves as an entry page for a new logical sub-domain.

In order to determine the logical domain entry pages, a set of rules $\mathcal{R} = (R_1, \ldots, R_n)$ based on page title, URL, anchor text, link structure and citation popularity is used. Each rule $R_i$ has an associated user adjustable scoring function $R_i(p)$. Thus the total score $R(p)$ of page $p$ is given by the sum of all scoring functions for that page:

$$R(p) = \sum_{R_i \in \mathcal{R}} R_i(p) \tag{3.1}$$

The higher the score of a page, the more likely it is that the page is a logical domain entry page. After all (site-internal) pages $p \in G_S$ have been assigned a score, a portion of the top scored pages is used as entry page candidates for boundary definition of the logical domains.

Table 3.1 summarizes the single rules which are now presented in more detail. In general, we followed [LKVT00] for the rule and initial scoring function definition, with some minor alterations and extensions.

*Rule $R_1$:* If the URL ends with a user home directory of the form `user/`, the URL most probably refers to an entry page of a personal Web site that is treated as a logical domain.

*Rule $R_2$:* An URL path that contains a word given in the user-defined *topic word list* and is not under a user home page is probably a logical domain. Some standard words in the topic list include `faq`, `tutorial`, `research`, `people` etc.

*Rule $R_3$:* If an URL ends with a `/`, there is an index page (i.e. `index.html`) in that directory. Thus is it can be assumed that this URL is used as an entry page for that specific directory.

| Rule | Type | Description | Scoring |
|---|---|---|---|
| $R_1$ | URL | "/~[^/]*/?$" | +60 |
| $R_2$ | URL | "^[^~]*/(topic word)/$" | +30 |
| $R_3$ | URL | "^[^~]*/$" | +20 |
| $R_4$ | URL | "/cgi\-bin/" | -100 |
| $R_5$ | Title | "\bhome\b" $\vee$ \bweb\b.*\bpage\b | +10 |
| | | "\bwelcome\b" | +5 |
| $R_6$ | In-link | ^home$ | +5 |
| | text | \bgo\b.*\bhome\b | +5 |
| | | \breturn\b.*\bhome\b | +5 |
| $R_7$ | Out-link | ^home$ | -10 |
| | text | \bgo\b.*\bhome\b | -10 |
| | | \breturn\b.*\bhome\b | -10 |
| $R_8$ | Ext-link | # $>$ 0 | +20+20% |
| $R_9$ | Out-link | # $>$ 20 | +5 |

Table 3.1: Ranking rules (given in *grep*-like regular expressions) and initial scoring functions

*Rule $R_4$:* We do not consider dynamically created pages (indicated by a `cgi` or `bin` part in the URL) as possible candidate entry pages.

*Rule $R_5$:* A page that has the phrase `home`, `welcome` or `homepage` in the `<Title>` tag is probably a logical domain entry page.

*Rule $R_6$:* A link pointing to a page with the phrase `home`, `go home` or `return home` in the anchor text probably indicates that the page being pointed to is a logical domain entry page.

*Rule $R_7$:* This is the counterpart to $R_6$. A page is unlikely to be an entry page if it contains links that may point to a possible entry page.

*Rule $R_8$:* A page is likely to be an entry page if it has external incoming links from other physical domains. This is due to the fact that people tend to link the domain entry page rather than a specific page. The higher the number of external incoming links the higher to probablity of the page being a logical domain entry page.

*Rule $R_9$:* If a page contains more than 20 outgoing links, it most probably serves as a domain entry page.

It has to be mentioned that the initial scoring functions and rules described above can be extensively configured by the user to fit their specific needs (see Figure B.4). However, we already achieved more than acceptable results using the initial values mentioned.

### 3.2.2 Domain Boundary Definition

After all pages have been ranked, the boundaries of the logical domains are identified using the URL path information. The boundary definition starts by assigning all available pages under specific entry pages to form logic domains in a bottom-up fashion. Afterwards, adjustments are made to remove and merge small logical domains.

First, the $k$ top-scored pages $(p_1, \ldots, p_k) \in G_S$ are selected from the structure graph to form the set of entry page candidates $\mathcal{E}$:

$$\mathcal{E} = \{e_1, \ldots, e_k\} = \{p_1, \ldots, p_k | \forall p_i \in G_S : R(p_i) > R(p_{i+1})\} \tag{3.2}$$

These entry pages also define an initial set of respective logical domains $\mathcal{D} = (D_1, \ldots, D_k)$. Next, the parent-child relationships for the logical domains are

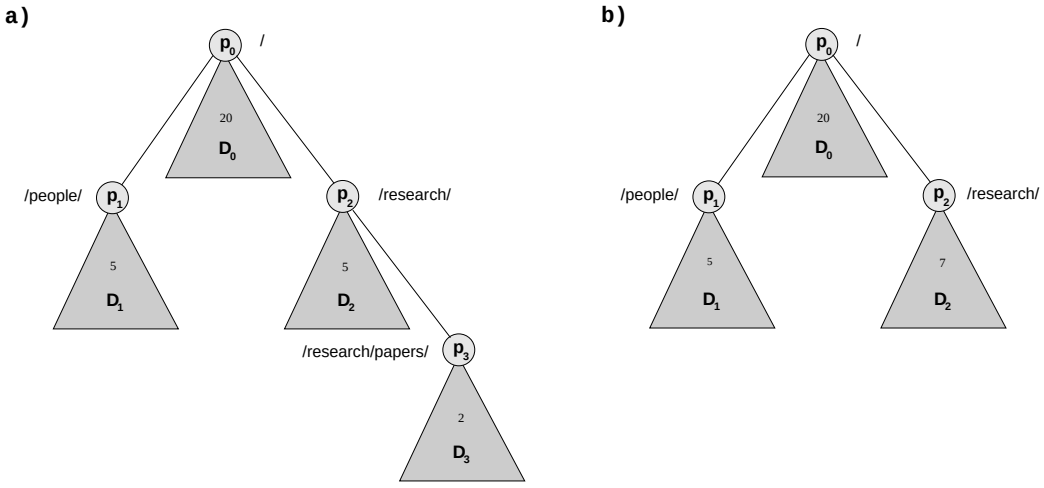


Figure 3.1: Logical domain cluster trees: a) initial assignment; b) after bottom-up merging

calculated and stored in a list. This is done using the URL path information

in the following way: It is assumed that only pages under the same directory root can be contained in the same logical domain, since this reflects the way in which people organize HTML files. Thus, a page must be located under the same directory as the entry page or in a subdirectory under the entry page. The initial logical domains are therefore ordered hierarchically using the URL path (the pure host directory without domain and filename) of their entry pages: The parent $p_i$ of a page $p_j$ is defined as the the page from the entry page set whose URL path has the longest prefix of the path of $p_j$.

Taking Figure 3.1a) as an example, $p_0$, $p_1$, $p_2$ and $p_3$ are the entry pages for the respective logical domains $D_0$, $D_1$, $D_2$ and $D_3$. The resulting parent-child relationships are then given by $(D_0, D_1)$, $(D_0, D_2)$, $(D_2, D_3)$.

The same way that the logical domains were ordered, the remaining Web pages $p_j \in \{G_S \setminus \mathcal{E}\}$ are assigned to the matching logical domain via the longest URL path prefix. The small numbers in the domains in Figure 3.1 refer to the total number of pages assigned. Since all pages share at least one prefix, namely the root `/`, each page is assigned to a logical domain.

The whole logical domain clustering which has an upper bound of $O(n^2)$ with $n = |G_s|$ is described in pseudo-code in Algorithm 2. The final step

**Algorithm 2:** Logical domain clustering

**Data** : Site structure graph $G_S$
**Result**: Hierarchical clustering of logical domains

**for** $\forall p \in G_S$ **do** $R(p) = \sum_{R_i \in \mathcal{R}} R_i(p)$;
$\mathcal{E} \leftarrow \{p_1, \ldots, p_k | \forall p_i \in G_S : R(p_i) > R(p_{i+1})\}$;
**for** $\forall p_j \in \mathcal{E}$ **do**
  $p_i \leftarrow p \in \mathcal{E} | p.\mathsf{hostDir}$ is longest prefix of $p_j.\mathsf{hostDir}$;
  `setParentChildRelation(`$D_i, D_j$`)`;
**end**
**for** $\forall p_j \in \{G_S \setminus \mathcal{E}\}$ **do**
  $p_i \leftarrow p \in \mathcal{E} | p.\mathsf{hostDir}$ is longest prefix of $p_j.\mathsf{hostDir}$;
  $D_i \leftarrow D_i \cup p_j$;
**end**
$\mathcal{D}_M \leftarrow \{D_j \in \mathcal{D} : |D_j| < \mathsf{minimumDomainSize}\}$;
**while** $\mathcal{D}_M \neq \emptyset$ **do**
  $D_j \leftarrow$ `getDeepestDomain(`$\mathcal{D}_M$`)`;
  $D_i \leftarrow$ `getParentOf(`$D_j$`)`;
  $D_i \leftarrow D_i \cup D_j$;
  $D_j \leftarrow \emptyset$;
**end**

of the clustering consists of recursively merging the domains $D \in \mathcal{D}$ which have less than a user-defined amount of assigned pages. Therefore, the set of all merging candidates is calculated via

$$\mathcal{D}_M = \{D_j \in \mathcal{D} : |D_j| < \mathsf{minimumDomainSize}\} \tag{3.3}$$

Then, all $D_j \in \mathcal{D}_M$ are recursively merged with their respective parents $D_i$ in a bottom-up fashion until all domains contain more pages than the user-defined minimum domain size. The merging operation itself simply consists of moving all pages of the child to the parent domain and removing the domain itself afterwards (see Figure 3.1b).

Figure B.3 shows the *logical domain editor* which can be used to explore the resulting hierarchical domain clusters using a tree-view. It also serves as a basis to invoke the clustering and visualization functions.

## 3.3 Related Work

The first work on the structural analysis of hypertext was presented by [BRS92] who introduced several interesting metrics: The *Second-order connectedness* (SOC) (and *Back second-order connectedness* (BSOC)) of a node is given by the number of nodes that can be reached by following at most two links (or respective the number of nodes that can reach the node in at most two steps). The *relative out-centrality* is used as a special centrality metric that defines the central node as being the one whose distance (= path length) to all other nodes in the hypertext is small. Global metrics comprise *compactness* that indicates the interconnectedness of a hypertext and *stratum* that describes wether there is an order for reading the hypertext. He also identifies *index* and *reference* hypertext nodes by comparing the in- and out-degree of a single node with the mean of node in/out-degrees.

In [BS91], the same author proposes a clustering method based on aggregation. This is done by splitting the graph in connected components. The first clustering algorithm removes the outgoing edges from index nodes and incoming edges from reference nodes and then tries to find the biconnected components in the undirected graph. A reduced graph is then calculated (which forms a tree) and the algorithm iterates for each of the biconnected components. The second algorithm takes into consideration the direction of the links and thus operates with strongly connected components.

[MH97] uses the BSOC and SOC metric together with in/out-degree, access frequency and depth in the hierarchy to calculate an importance metric of a node that distinguishes it as a *landmark*. Using a *focus and context* technique, a *local context* is calculated which consists of the direct neighbors

of the currently selected nodes while the *global context* consists of the shortest paths to the landmark nodes that can reach the current node or are reachable from this node.

[Che97] proposes a framework for structuring and visualizing hypertext named *general similarity analysis*. A special graph is built from the existing hypertext using three different proximity measures for linkage, content and state transitions that are applied in combination to provide a similarity metric for each pair of nodes that is further used as edge weight. A minimum cost network called *pathfinder network* is then employed to select a subset of the graph for further visualization.

The *navigational view builder* (see [MFH95]) can generate several hierarchies from a hypertext document by using structure and content similarities among the nodes. Given a root node, the system partitions the graph into branches that equal clusters according to a specified metric which are then recursively subdivided to form a tree. The system is very interactive by prodiving support for selecting various attributes and filtering methods.

[PP97] and [TH98] both focus on using *co-citation analysis* for clustering: If node `A` has a link to both node `B` and `C`, it can be induced that `B` and `C` are somehow related although they are not directly linked.

Another interesting approach is presented by [PPR96] and uses structure, user accesses and text similarity metrics to assign pages to several categories like head, index, content and so on. They also build one graph for each similarity metric which is then used to calculate a *spreading activation* (which is in fact a kind of flow that decays with time) through the networks from a specified start node. As soon as the networks reach a quasi-equilibrium, the nodes with the highest "activation" are considered to be especially important and are used as roots for aggregation. This approach is heavily influenced by the theory of *information foraging*: The Web can be seen as an evolving ecology of information-bearing items or *memes*[4] competing for human attention in their information habitat. As the human *user-forager* hunts for information, he tries to maximize his information gain and thus follows the *information scent* through the Web.

[4]cultural knowledge units, analogeous to genes, that are transmitted and replicated

# Chapter 4

# Structure Visualization

In this chapter, three different visualization approaches for the extracted structural properties of a Web site are presented.
The *force-directed topology layout* displays the raw topology data (see 3.1) of a Web site as a directed graph that is layouted using a force-directed model. Since the structure data (raw as well as the result of the logical domain clustering) is highly hierarchical, two remotely tree-like visualizations were studied: The *2D cone-tree layout* is a variant of the traditional 3D cone-tree visualization scheme and is used on the logical domain data as well as bfs-processed raw structure data. The *tree-map layout* based upon a space-filling 2D embedding of a tree formed by the logical domain cluster data proved to be the most robust and navigationable solution and was thus also implemented as Web-based sitemap.

## 4.1 Force-Directed Topology Layout

The first structure visualization is based upon using the site structure graph $G_S$ (see 3.1) as a basis for visualization. In particular it enables an inspection of the Web site (and its close surroundings) on the topology level: edges in the graph are equivalent to hyperlinks while nodes represent single HTML pages.

The graph is subjected to the *Y*-built-in *force-directed layouter* (see Figure 4.1) which is based upon the following approach (see also [BETT99]): each edge $e = (u, v)$ in the graph is modeled as a "spring" that excerts a force on the respective nodes $u$ and $v$. The layout algorithm now iteratively places the nodes to find the local minimum energy configuration. This approach is widely used, because it produces mostly pleasant drawings that are highly symmetric and tends to distribute vertices evenly. As can be seen

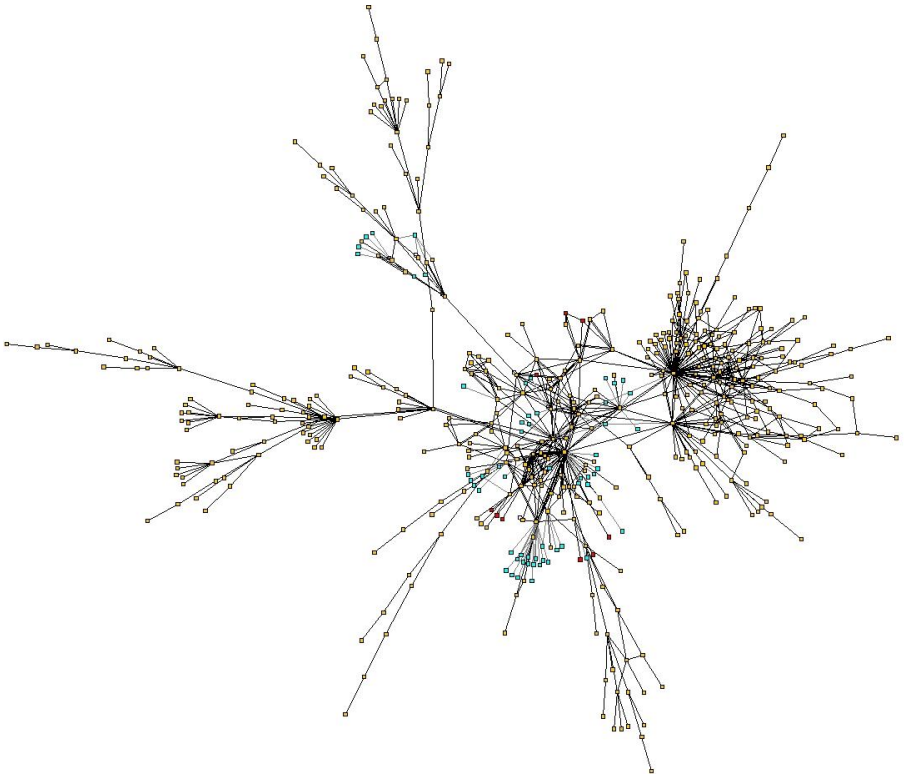

Figure 4.1: Force-directed topology layout of the site graph defined by a subset of the department Web site with 537 nodes and 837 edges

in Figure 4.2, colors are assigned to reflect the different types of nodes contained in $G_S$: error nodes (red), external referrers (cyan) and normal site nodes (yellow). Labels are also displayed for each node (URL or title) and edge (anchor text) in the graph.

Clearly, this visualization approach was not intended to serve as basis for navigation sitemaps. Nevertheless it proved highly valuable in order to gain an insight into the topological nature of the respective Web sites and do some hands-on exploration. Due to the vast amount of information displayed (the generated structure graphs had a maximum of several thousand nodes), a great deal of zooming operations is needed to gain valuable insight on large graphs. In addition, the force-directed layout algorithm scales very badly and thus can take several minutes to produce an acceptable diagram.

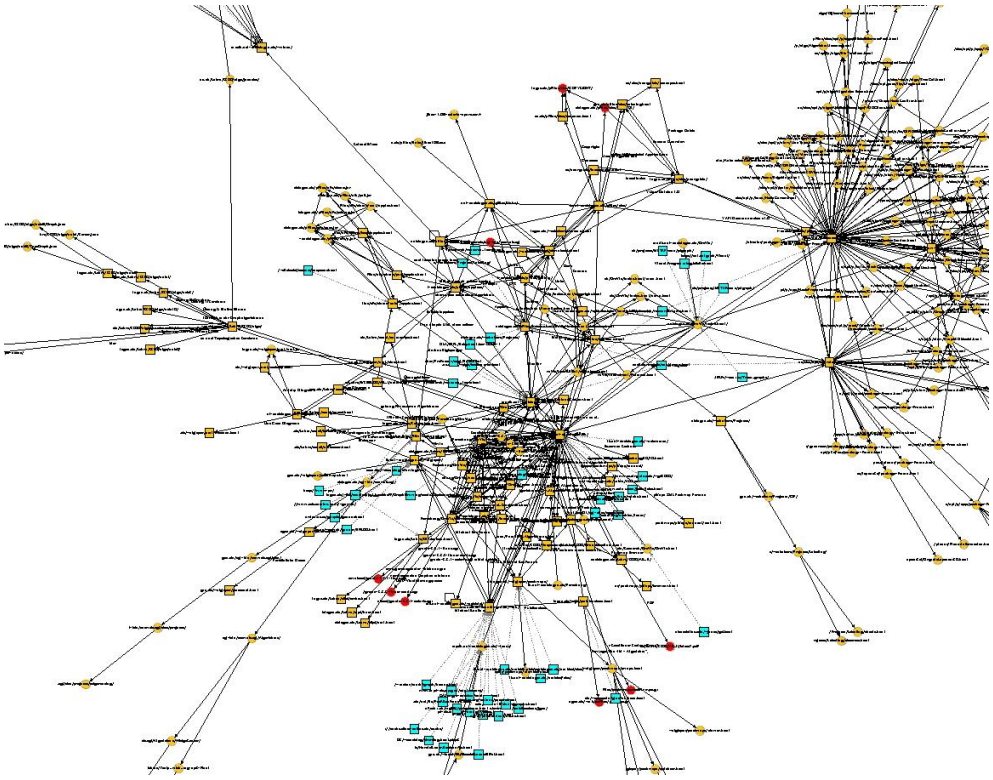

Figure 4.2: Zoomed force-directed topology layout of the Department Web site

## 4.2 2D Cone-Tree Layout

The second approach investigated the visualization of (possibly huge) hierarchies generated by a breadth first traversal of the graph $G_S$ or the logical domain clusters. In information visualization, *cone-trees* (see [CMS99]) have been used as a three dimensional representation of hierarchical information, where one node of a tree is located at the apex of a cone and all of its children are arranged around the circular base of the cone. Thus, the cone-tree can present a structure in a way that nearly the entire hierarchy is visible without scrolling.

[CK95] have presented a 2D embedding algorithm of the traditional cone-tree that takes care of possible cone overlaps and can effectively visualize very large hierarchies. The algorithm itself operates in two phases. During the first phase, the radius $r$ of the cones and the position given by the tree level and the angle $\theta$ are calculated in bottom-up fashion from the leaves to the root of the tree. In the second phase, the layout is generated by traversing the tree in top-down fashion, rendering the nodes with the assigned values.

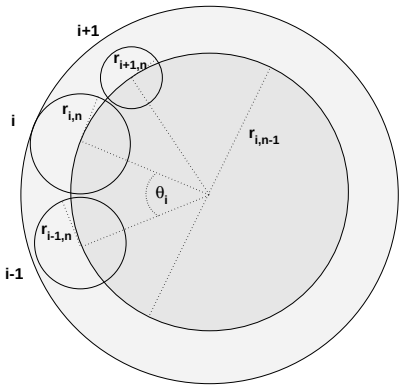


Figure 4.3: 2D cone-tree layout (taken from [CK95])

Apart from being quite space efficient compared to the previous layout, the algorithm performs very well since it is based on approximation.

Taking into consideration Figure 4.3, the circumference $C_n$ for level $n$ can be approximated via the sum of all diameters of the cones $i$ in level $n$:

$$C_n \approx 2 \sum_i r_{i,n} \tag{4.1}$$

The radius $r_{n-1}$ of the next cone in level $n-1$ can then be estimated via

$$r_{n-1} = \frac{C_n}{2\pi} + \delta \tag{4.2}$$

where $\delta$ refers to a user-defined factor that controls the "density" of the whole layout. If the arclength $s_i$ occupied by cone $i$ is approximated via $s_i \approx r_{i-1,n} + r_{i,n}$, its angle $\theta_i$ in respect to the center can then be expressed via

$$\theta_i = \frac{s_i}{r_n} \approx \frac{r_{i-1,n} + r_{i,n}}{r_n} \tag{4.3}$$

Figure 4.4 shows a 2D cone-tree layout of a BFS-tree generated from the structure graph $G_S$ of a Web site. The main problem with this visualization scheme results from the property of the displayed trees. Due to the approximation, the visualization works best for well-balanced trees with lots of child nodes. Since neither the resulting BFS-tree nor the logical domain cluster tree always exhibit this property, the layout easily degenerates as the tree approaches a linear list. Together with the fact that navigation to single nodes can sometimes be quite difficult and requires a fair amount of zooming due to vast differences in the cone size, we decided to abandon further experimentation with this layout type.

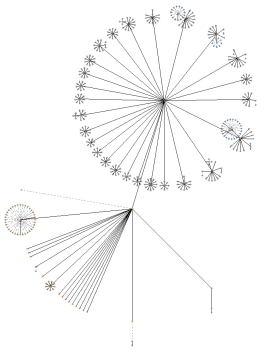

Figure 4.4: 2D cone-tree layout of the *Ouk* data set (413 nodes)

## 4.3 Tree-Map Layout

Our next layout approach focused on finding a suitable, space-constrained visualization for the logical domains alone, which led us to a different visual metaphor than the traditional *node and link metaphor* by depicting hierarchy through topological inclusion.

"Classical" graphs are usually composed of nodes, represented by dots or boxes, and edges represented by straight, polygonal or curved lines which can be either directed or undirected. Some less "classical" graphical notations are Venn diagrams and higraphs.
Harel [HY00] introduced the notation of a *higraph* that consists of *blobs* (rounded-corner rectilinear shapes) possibly connected by edges that are arranged in an inclusion hierarchy. Higraphs with non-intersecting blobs and without edges are also known as inclusion graphs or *tree-maps* [Shn92]. The general tree-map algorithm is quite simple: Given an attributed tree with the

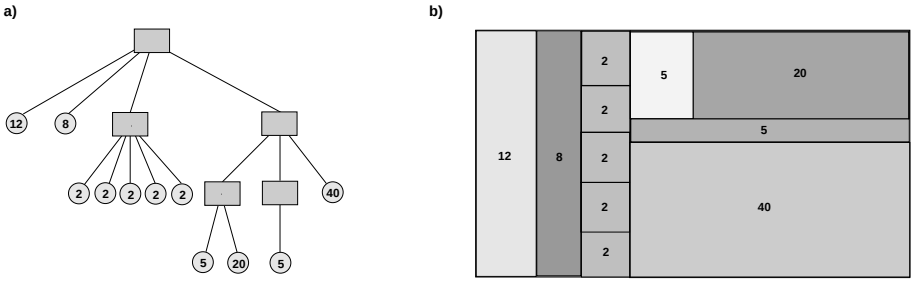


Figure 4.5: Conversion of an attributed tree a) into a tree-map b)

leaves of the tree being labeled according to a certain metric (e.g. size) as shown in Figure 4.5a, the corresponding tree-map is constructed via a "slice and dice" approach as follows. First, all non-leaf node attributes are assigned

the sum of their child attributes or scores in a bottom-up fashion (the root node in Figure 4.5a will thus receive the score 100). Then, a rectangle $r$ is given that defines the borders of the tree-map which is then rendered by traversing the tree from top to bottom. For each child node, the size and position is determined by the parent rectangle that is subdivided in relation of the child score to the parent score. The subdivision is carried out horizontally for all even levels in the tree and vertically for all odd levels (or vice versa). The tree-map node layout can thus be implemented recursively as shown in procedure `layoutNode(...)`.

The whole tree-map is rendered by a single call to the function `layoutNode`

**Procedure** `layoutNode`(*Node n, Orientation o, Rectangle r*)

```
paintNode(r);
for ∀m ∈childs(n) do
    switch o do
        case horizontal:
            width ← (score(m)/score(n)) * r.width;
            layoutNode(m, vertical, (r.x, r.y, width, r.height) );
            r.x ← r.x + width;
        case vertical:
            height ← (score(m)/score(n)) * r.height;
            layoutNode(m, horizontal, (r.x, r.y, r.width, height) );
            r.y ← r.y + height;
    end
end
```

with the parameters *rootNode*, *horizontal*, *boundingRectangle*) in $O(n)$, with $n$ being the number of nodes in the tree. As can be seen in the resulting map in Figure 4.5b, the visualization is space-filling and manages to place *all* leave nodes in the bounding rectangle.

Compared with the traditional tree visualization via the link and node metaphor, more information can be embedded on a constrained viewspace because links that otherwise easily clutter the display are omitted. In addition, due to the topological inclusion, hierarchies can be identified more easily. Just recently, new algorithms for *cushion tree-maps*[1] have been presented, that make use of advanced shading models from computer graphics to further enhance the identification of nested hierarchies

[1] "Kissenschlacht mit Disk-Inhalten", c't 2001, Heft 5 or `http://www.win.tue.nl/sequoiaview/`

On the other hand, the size of nodes in the tree-map can be difficult to compare since the score maps to the area covered by the node. A long thin node for example can have the same total area than a boxed-shaped one, but is considered as being smaller than the latter by the human perceptual system. Besides, users have to be instructed to be able to interpret (left to right, top to bottom) a tree-map if see it for the first time.

We now explore the application of tree-maps to the trees that are formed by the hierarchy of logical domains $\mathcal{D}$. First, the trees have to be converted so that logical domains only appear in the leaf nodes by inserting meta nodes and inserting the parent as the first child of the meta node (see Figure 4.6). In

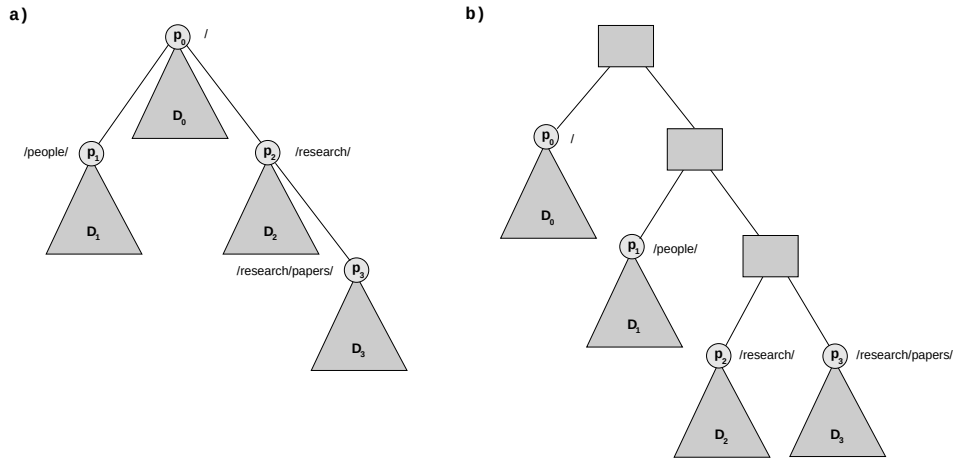


Figure 4.6: Tree conversion: a) logical domain tree; b) respective tree-map tree

addition, we augment the node layout algorithm described above to produce a *nested tree-map* by adding a user-defined offset and border when layouting the single nodes to make the hierarchical inclusion even more conspicious. The data used for the scoring in the logical domain nodes is explained in the next section (4.3.1).

The last step of the logical domain tree-map layout consists of a heuristic used for labeling the resulting tree-map nodes: Initially, the `<Title>` tag entry for the respective logical domain entry page is used as label string. The last character of the label string is then chopped off until the label fits into the node. For pages that do not posess a `<Title>` tag, the URL is taken as label string from which we iteratively chop of the first character (because the filename is located at the end). Labels for nodes that are arranged in horizontal order are rendered vertical (and vice versa) by inserting linefeeds after each character.

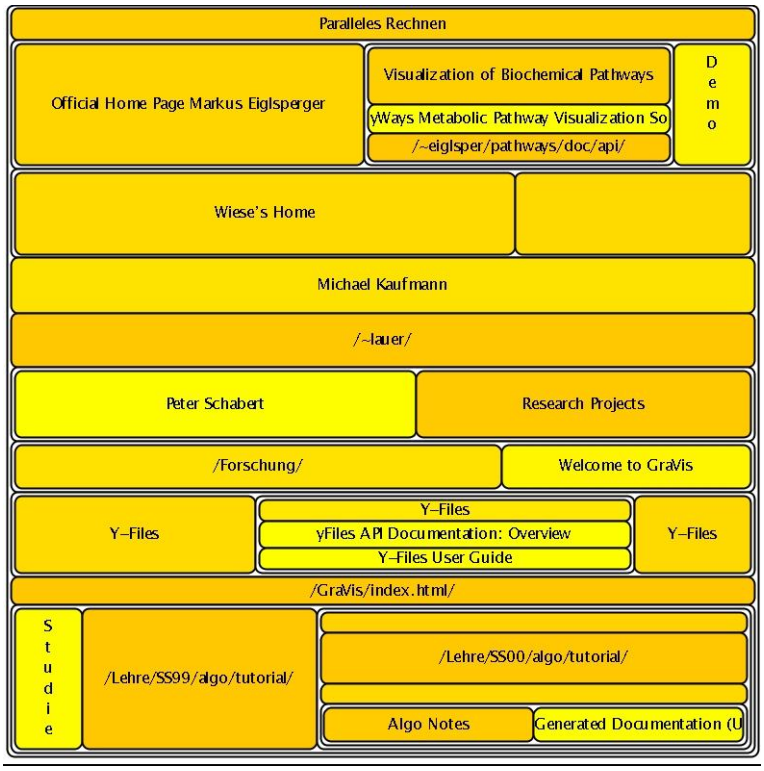


Figure 4.7: Logical domain treemap visualization of the *Department* data set with color mapped from the logical domain date and size from the entry node scores

Figure 4.7 shows the resulting tree-map for the hierarchic logical domain clusters that was generated by starting with a vertical sub-division. Thus, reading from top to bottom, we can identify the logical domains at depth one from the site's root (which is represented by the top-most node). If we select one of these horizontal nodes, we now read from left to right to identify his siblings (with the leftmost node indicating the parent node).

### 4.3.1 Visual Mapping

In general, two main *visual channels* are available for the tree-map nodes: *color* and *size*. The following properties can be selected by the user to either map to the color and/or size channel:

*Node Score*: The ranking value $R(e)$ of the entry node $e \in D$.

*Domain Score*: The mean ranking value $\frac{1}{|D|} \sum_{p \in D} R(p)$ of the domain nodes.

*Domain Size*: $|D|$ defined as the number of pages in the logical domain.

*Domain Date*: The mean date of all pages $p \in D$.

While the selected data for the node is directly assigned to the node attributes in the tree, the color is calculated via a linear interpolation between the maximum and minimum values available.

The mapping scheme allows the construction of navigation sitemaps that emphase different properties of the visible clusters. Figure 4.7 for example gives a quick overview about the most important domains by mapping the node score to the size channel, while newer parts of the site can also easily be recognized because they get assigned a brighter value by the color channel.

### 4.3.2 Interactivity

Apart from *tooltips* that display various information about the respective logical domain on mouse-over events, interactivity is only available for the *sitemap creator*.
The layout is highly configurable through user-definable colors (see Figure B.5), layout sizes and borders as well as the mapping to the visual channels as described above (see Figure B.5).
A one-click export of the designed logical domain tree-map as a Web-based visualization (which is explained in the next section) that can be readily added to an existing Web site is also provided.

### 4.3.3 Web-based Structure Visualization

Via a Web-based visualization, the previously described logical domain tree-map layout is made available to the site visitor as a navigation sitemap.
Our intention is to display the sitemap next to the browser so that both can be used at the same time without occluding each other (see Figure 4.8).

As stated in 2.2.2 an important design goal is the seamless integration into the existing Web site without requiring major changes to the server configuration or the existing HTML pages. We therefore simply replace the existing root `/index.html` by a HTML file that spawns the navigation sitemap in a new window next to the browser and loads the old index page in the current window.
Since the current structure visualization does not provide a lot of interactivity to the visitor, a sitemap based on a HTML *client-side imagemap* was chosen for several reasons: First, the whole layout can be easily exported as

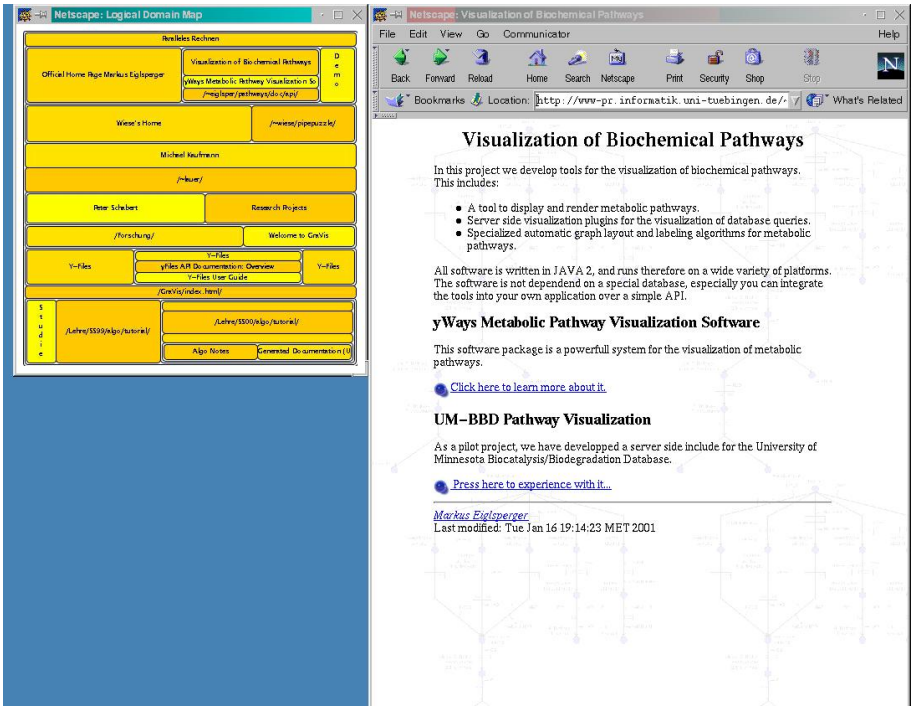


Figure 4.8: The Web-based logical domain tree-map visualization used for exploring the Department's Web site

a JPG or GIF graphics file from the generated diagram. Second, client-side imagemaps do not generate any additional load on the server (except for the network bandwith needed to transfer the picture) and thus scale very well even for a large number of users.

A combination of *Javascript* and *Cascading Stylesheets* (CSS) is used to implement cross-browser capable tool-tips whose positions are pre-calculated so that they don't occlude the mous pointer and then stored in the HTML file. The cross-browser implementation can be considered an especially tedious task since the employed object models (DOM used by *Internet Explorer* vs. *Netscape's* own object model) are far from being compatible and required a lot of testing.

The resulting navigation sitemap is presented to the user when he visits the site (its size being only about 100KB in total). He interacts with the navigation map by simply letting the mouse pointer float above potential interesting parts of the map in order to get a tooltip describing the domain's properties in more detail. By clicking on a logical domain, the other browser window is instructed to load the specific entry page for that domain. Because of the static map and the minimal interaction possibilities, the resulting nav-

igation sitemap can be regarded as very easy to comprehend by an occasional visitor of the site.

## 4.4 Related Work

Most existing visualiziations for Web structure data are based on the traditional *node and link metaphor* and scale very badly for hypertext of actual usable sizes. We now present a variety of approaches for structure visualization although most of them are not applicable given the navigation sitemap requirements stated in 2.2.2.

The *nicheworks* system ([Wil99]) presents several layout techniques that can be used for the structure visualization of very large graphs of 10.000-100.000 nodes in a Web context. Their *circular layout* simply place the nodes randomly on the periphery of a single circle while the *hexagonal grid layout* uses the grid points of a regular hexagonal grid. Both layout methods can be interactively refined. The *tree layout* is inspired by a readial tree layout where the nodes are placed with the root node in the center, then each connected node in a circle around that. The spacing between nodes reflects the number of nodes in the subtree originating from that node.

An even better visualization for large hierarchies induced from the Web structure graph is the *hyperbolic tree* that was already presented in 2.3.

The *general similarity analysis* ([Che97]) presented in 3.3 uses a simple spring model for the layout of the resulting graph for visualization (refer to 4.1) while the *navigational view builder* ([MFH95]) uses a 3D cone-tree like view and a 2D tree view.

Another important visual metaphor is the *landscape metaphor* employed by the *MAPA* system that was already presented in 2.3). Analogous to the real world, places, roads, crossings etc. are used as means of navigation because they are supposed to be highly familiar to the user.

[MH97] also uses the landscape metaphor by visualizing the local and global neighborhood of landmarks via 3D VRML models. Another view presents all landmarks of a site with 3D cubes and the relevant links between them analog to a city landscape.

An new visual metaphor called *auditorium* is presented by [TH98] for the visualization of *clan graphs* (a graph that groups together sets of related sites). It consists of concentric semi-circles with the center being the site to explore which are used to group sites into equivalence classes from most to least important. The system also allows dynamic ordering of the sites presented by preview thumbnails within the semi-circles depending on user selection. Progressive relevation of greater detail like in- and out-edges is

available upon user focus.

Radial layouts for whole Web sites called *disk trees* are used by [CPP00] to construct *time tubes* that equal stacks of disk trees for various dates in order to provide Web ecology and Web evolution visualization.

In his survey, [HGM00] also considers different *focus and context* approaches as a navigation technique complementary to zoom and pan. *Graphical fisheye views* are the most popular form which uses the fisheye lens effect in order to enlarge the area of interest (e.g. the focussed node), while showing other portions of the image with less detail. The hyperbolic tree described before is in fact a special application of a focus and context technique.

# Chapter 5

# Usage Data

Our main intention in considering usage data for the visualization of Web localities is to provide an alternative view of the site, that reflects the actual behaviour of the visitors. Since users have different goals, the original layout of pages in a Web site might hide the most important or frequently used pages in "unlikely" places, making it inconvenient for users to retrieve them. Thus, usage-based sitemaps can offer a form of *collaborative filtering* in a sense that they make use of preferences of other people to predict the documents which may be of interest to a particular user.
In this thesis, we focus on the analysis of *user paths* in a sense of coherent sequences of nodes followed by an individual, instead of single pages, since it can be assumed that distinct, common paths map to distinct navigation goals.

The current chapter describes an approach for extracting popular user paths from usage data in order to be used for visualization in user-centric sitemaps.

## 5.1 Data Sources for Usage Analysis

Several data sources that widely differ in availability, reliability and user acceptance are available for usage analysis. They are categorized by the location where the data is being collected:

*Client level* data sources provide the most reliable usage data since the data is collected right on the users machine. This is usually done by modified Web browsers that collect all user interactions and thus make the reconstruction of travelled user paths fairly easy. Another possibility is the use of remote agent tracking systems (see [FSMF99]) which are rarely used.

Since client level data collection requires the user to install and use special tools, its usage is limited to research prototypes and is therefore not applicable in our context.

At the *proxy level*, usage data is collected in between the users machine and the Web server. As shown in Figure 5.2, a user request often travels through various caches like corporate firewalls and proxies. Therefore some systems use the data recorded in the proxy logs to analyze user behaviour, sometimes combined with the need for the user to identify himself to the proxy via a login procedure. In addition, many users access the Web through a chain of proxies, which complicates the collection of coherent data. High-end usage mining and CRM systems like *Accrue Insight* [1] use advanced network monitoring and analysis techniques such as packet sniffing and TCP header analysis to infer single user behaviour from network traffic. Obviously, such a sophisticated techniques require a special infrastructure and is very expensive.

*Server level* data is easily collected on the Web server machine itself and therefore does not require the user to install special software or identify himself (except for systems that require a membership and login procedure to access the site content). Mostly, the Web server *access log* that captures all requests the server receives is used for further analysis. Since it only records IP addresses or hostnames for identification, several techniques were introduced to facilitate the tracking of single users.
A common practice is the use of *cookies* that were introduced as a solution for the problem of statelessness of the HTTP protocol: When a Web server receives a request for a page of content, it embeds a hidden directive instruction in the header of the returned document, initiating the client computer to store the attached identity data (known as "cookie") on its local disk and associate it with the specified Web page. The next time the user's browser visits the same Web page or another page within the originating site, it sends the stored cookie data back to the server along with the request for the page. Unfortunately, the use of cookies for more precise user identification has several drawbacks: First of all, the Web server itself has to be reconfigured to issue cookies on every request. Second, rising awareness about privacy issues has led many users to instruct their browser not to accept cookies or to employ "cookie-monsters" that automatically delete or obfuscate cookies as they are stored on the client (thus making identification impossible).
A more unobtrusive and reliable technique is the use of *URL-embedded ses-*

[1] `http://www.accrue.com`

*sion identifiers*. When a user visits a Web site, the server generates the page he requested on the fly, embedding a unique session identification string in all URLs[2]. By requesting such a link, the server receives the URL and parses the session id to assign the page request to the corresponding user. Unlike cookies, which are persistent, this technique fails if the user re-visits a site after he stopped browsing or switched of the computer because a new session id is generated for every request that does not already contains a session id in its URL. In addition, the Web server has to be reconfigured, in order to embed and parse the identifiers. Other techniques like *super cookies* or *omniscient observers* are mostly Java augmented 1-pixel GIFs that have to be manually embedded in every page (see *HitBox*[3] or *NedStat*[4] for example).

| Level | Quality | Availability | Unobtrusivness |
|---|---|---|---|
| Client | + | - | - |
| Proxy | - | - | + |
| Server | - | + | + |

Table 5.1: Comparison of different usage data sources

Table 5.1 summarizes the properties of the various data sources. Since our main goal is to make the usage data visualization available on large variety of machines, data availability and unobtrusiveness are our main concerns. Thus we concentrate on server level data sources for analysis. Because we do not want the Web administrator to be forced to make changes to the servers configuration, we decided to use the pure access logs.

## 5.2 Inferring User Paths from Web Server Logs

### 5.2.1 Web server access log data

In order to understand how visitors navigate a Web site, the access log file that is stored on the server is analysed. Each time a request reaches the Web server, one or several log entries are recording the response of the server. Due to the nature of the HTTP protocol, a separate connection is required

[2] i.e. `http://www.sun.com/research/;$sessionid$LT2JMAQAAAXRFAMTA1FU5YQ`

[3] `http://www.hitbox.com`

[4] `http://www.nedstad.com`

for every file that is requested from the server. Therefore a user's request for a particular page often results in several log entries since graphics, scripts, stylesheets etc. are downloaded in addition to the HTML file. Since every new access is appended at the end of the file, it is already ordered sequentially by access time.

### Combined Log Format

Figure 5.1 shows a sample entry of an access log in the widespread *NCSA combined log format*[5] which is supported for data analysis. It captures a

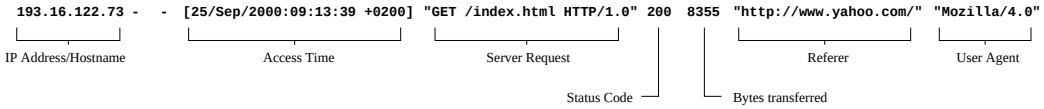


Figure 5.1: Sample Web server access log entry

request for the file `/index.html` from a client with the IP address `193.16.122.73` which is successfully transferred. The `referer` [sic!] and `user agent` fields suggest that the user previously visited `www.yahoo.com` and probably uses the Mozilla Web browser (or a derivate).

There are cases for which some of the fields are unavailable (which is indicated by a "`-`" character in the access log): users may explicitly configure their Web browsers not to send out header information to the Web server for privacy reasons which will result in an empty `referer` field. This is also the case for requests initiated by the user via entering the destination manually in the URL field of the browser instead of following a hyperlink or hitting the browser's reload button. Unfortunately the exact behaviour varies with every browser implementation.

Other formats that do not include the referrer and user agent fields were not considered for analysis, since they dramatically impair the chance of user identification and path reconstruction. The same applies to formats that record the referrer information in an extra log file, which implies the use of delicate synchronisation methods to match the corresponding entries.

## 5.2.2 Access Log Data Quality

Data recorded in the access log is mostly subjected to *temporal* (at what time of day do the most accesses occur ?) or *statistical* analysis (what are the top 10 accessed files ?) that can be generated quite easily by gathering

[5] See `http://httpd.apache.org/docs/mod/mod_log_config.html`

log line based metrics. However, *path* analysis requires the log entries to be grouped by user or session and single page requests by access order to obtain meaningful paths, which can be quite a tedious task due to the nature of the recorded data: Unfortunately the data gathered in the access logs is highly ambigous and incomplete.

In a typical Web content transaction there may be intermediate mechanisms that can obscure the origin and continuity of a Web browsing session (see Figure 5.2).

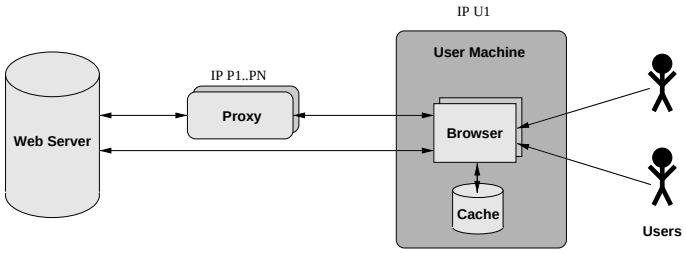


Figure 5.2: Chain of request during Web server access

**Caches** are widely used, especially in the browsing software itself (disk and memory), to speed up loading times of previously requested content. Unfortunately, a user request that is fulfilled by the local browser cache rather than the Web server does not result in an access log entry. Thus, depending on the caching level employed by the user, a large number of documents the user requested may be missing in the collected data which makes elaborate path reconstruction methods necessary (see 5.4).

**Proxies.** Another level of caching occurs at the corporate or internet service provider (ISP) level. Typically, the HTTP requests of users are passed through a proxy server that provides caching service, thus reducing the number of retransmission requests for popular sites.
Although techniques like *cache busting* (advising the browser and server not to use cached copies), can be applied by the site administrator, defeating the caching system is widely considered unacceptable by the network community and is not guaranteed to work seamlessly and should therefore be neglected as an option.

Another problem different from caching arises for user identification: When a proxy receives an URL request that is currently not cached, it initiates a HTTP request using its *own* IP number instead of the IP number of the client. Thus, to the Web server, all users from a proxy-cached ISP appear (in the access logs) to be aliased to the few IP numbers of the ISP's proxy servers. Therefore we can not easily establish a one-to-one mapping between

user and IP address. This is even complicated by the fact that larger ISPs (i.e. AOL) assign consecutive requests from the same client to different proxies for load balancing reasons. The identification problem described above also arises with the use of corporate firewalls and routers.

Due to the sole recordable interaction between the user and the Web server being the request of a document it is also not obvious when a user stops a navigation session. It is quite common that a user pauses several minutes while browsing the Web to complete other duties, later resuming his browsing session thus making it ambigious wether his behaviour should count as a new visit or browsing session or not, which points up to the necessity of a kind of user session grouping.

Also, depending on the site structure and access characteristics, the sheer size of the access logs that can easily reach 1 GB/month constitutes a significant challenge for analysis algorithms. Therefore, memory and speed efficiency have to be considered while developing methods that are able to cope with such high-volume input data.

We now present an approach that solves most of the problems described in this chapter and has been successfully applied to several large access log files (see Appendix A).

## 5.3 Path Extraction Algorithm

As stated before, our goal is to extract popular paths travelled by users solely from the Web server's access log files.

The ideal path set consists of long traversal sequences, that capture the users' navigations as they enter the site (or even before by considering the external referrer) and that reflect the level of popularity of the respective part of the Web locality.

The suggested algorithm consists of four subsequent stages that will be described in more detail in the following sections:

1. *Data Cleaning* (5.3.2):
   All entries that do not contribute to the path analysis are discarded.

2. *User/Session Identification* (5.3.3):
   Identify users and sessions and group accesses.

3. *Path Reconstruction* (5.3.4):
   Reconstruct complete paths from single accesses and extract all available paths from the data.

4. *Popular Path Extraction* (5.3.5):
   Extract the most popular paths from all available paths.

### 5.3.1 Path Model

When users travel a path through a Web page, they generate *clickstream data* that consists of single consecutive request entries that can be encoded as n-grams. Thus we can make the following definition:

**Definition 9 (User Path)** *A user path* $\mathcal{U} = <p_1, .., p_N>$ *is a time-ordered list of all Web pages* $p_1..p_n$ *with* $p_i \in W$ *that were requested by a single user during a browsing session.*

In order to facilitate the path analysis and make it more robust, [WYB98] constructs *maximum forward paths* from each user path which we adopt in this context:

**Definition 10 (Maximum Forward Path)** *The maximum forward path represents the largest prefix* $\mathcal{M} \subseteq \mathcal{U}$ *of a user path* $\mathcal{U}$*, such that* $\forall p_i, p_j \in \mathcal{M}; i \neq j : p_i \neq p_j$.

Thus the maximum forward path captures the largest subset of a user path until the first visit to a previously requested page occurs. This model is based on the assumption that backward navigations are only made for ease of travelling but do not contribute to the main "navigation intent" of the user. The back-tracking case is handled by constructing a new maximum forward path $\mathcal{M}_2$ that consists of the prefix of $\mathcal{M}_1$ till the backtracking point and appending the rest of the user path to it that is not contained in $\mathcal{M}_1$ and applying that rule recursively (this is explained in more detail in 5.3.4). In Figure 5.3 the user travels the sequence $\mathcal{U}$=<A,B,C,B,D>. The corresponding maximal forward references are $\mathcal{M}_1$=<A,B,C> and $\mathcal{M}_2$=<A,B,D> which can be interpreted the following way: Instead of navigating through <A,B,C> in order to visit page C and then continuing via <C,B,D> to visit page D, the user could also have visited <A,B,C> followed by <A,B,D> to achieve the same result.
By employing the maximum forward path model, we eleminate any cycles in the user navigation which makes the further path analysis step much easier.

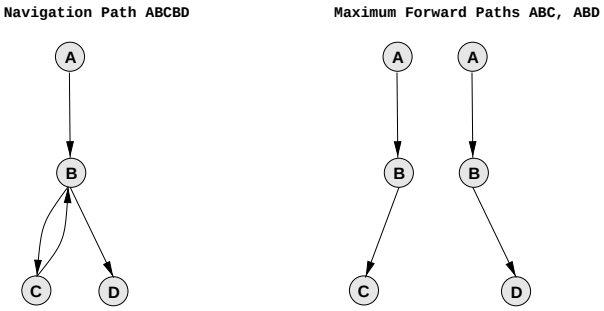


Figure 5.3: Maximum forward reference path construction

### 5.3.2 Data Cleaning

During this preprocessing step a large amount of data is skipped while sequentially analyzing the log file because it does not contribute to the analysis or may falsify the result. The following items are discarded:

**Non-HTML files.** Since the main intent is to get a picture of the user's behaviour, it does not make sense to include file requests that the user did not explicitly initiated or that do not contribute to the user's navigation. Therefore all log entries with URL filename suffixes indicating such items are removed. The default list of suffixes includes `.jpg`, `.gif`, `.ps`, `.css`, `.js` etc. and can be easily modified.

**Invalid Entries.** The access log also contains a large number of error requests that did not result in a file being transmitted to the user. Thus we remove all log entries containing malformed URLs (entered by the user), status codes other than `HTTP_OK` or a transfer size $< 0$ bytes. As previously stated, not all fields of an entry may be recorded. Since accesses that do not contain the `Hostname`, `Request` and `User Agent` field are of little use in the path extraction they are discarded in the preprocessing step. The same applies to entries that record other actions than retrieving a file via the `GET` command (namely `HEAD` for requesting header information and `PUT` for sending data to the server).

**Accesses from internal hosts.** Depending on the Web locality, many requests to information such as online API documentation or tutorials originate from users within the corresponding organization. Since they access the site more often and intensly than the average user, these requests may dominate the analysis and shift the result to areas of the site that are normally not of interest to visitors from the "outside". Thus a host/ip filter list was imple-

mented that discards any log entries containing the corresponding string.

**Accesses from Robots.** According to [TK00], roughly 10% of the recorded accesses account for robots, search engine crawlers and other software agents that automatically index the Web, search for information etc. Many of these robots adopt a breadth-first retrieval strategy to visit the whole Web site or a large subset and therefore generate a large number of false (and long !) paths in the analysis.
A simple strategy that was implemented in order to filter out these accesses is to ban all users (ip/user agent pairs) that accessed the file `/robots.txt` that is used by the site administrator to define the parts of the locality he doesn't want to be visited by robots via the *Robots Exclusion Protocol*[6]. Unfortunately the protocol is only implemented by a small number of "well-behaving" robots which made it necessary to implement a bot filter list that contains suspicious substrings that are likely to appear in robot names (e.g. `bot`, `crawl`, `spider`, `get`) and compares them with the `User Agent` field.

### 5.3.3 Session and User Identification

The most simplistic assumption to make about users is that each distinct IP address or domain name respresents a unique user. However, this is hardly practicable if we consider the problems described in 5.2.2.
Therefore we adopted a heuristic called *IP-Agent-Timeout* (presented by [PP99]) that uses a combination of the IP field, user agent information and access time. While sequentially processing the access log, the following cases are handled:

1. a new IP address is encountered: assume this is a new user
2. an already processed IP address is encountered:
   (a) the user agent matches previous requests: assume this is the same user
   (b) the user agent does not match previous requests from the same IP address: assume this is a new user
   (c) the difference to the last access with the same IP address larger than a defined timeout value: assume this is a new user

[6] see `http://info.webcrawler.com/mak/projects/robots/norobots.html`

[Pit98] has suggested a timeout value of 25 minutes based on his survey of interface event sequences in Web browsing sessions (with a mean time difference of 9.3 minutes between interface events plus 1.5 standard deviations).

The method described above reduces the problem of several users accessing a Web site through the same IP address that often use different browser versions (or builds) and thus can be detected via the agent field.
However, the case of one user having different IP addresses is not handled. These accesses result in paths that consist of only one node and are later removed in a step that eleminates all single entry paths.
One method for dealing with changing machine names that occur when users are accessing a site through load balancing proxies like AOL is known as *host munging*: based upon IP class chop the suffix off the IP addresses and the prefix off the domain names. Unfortunately this method could not be implemented due to time constraints.

[PP99] presents an empirical study that demonstrates the impact of different session identification methods (including *IP*, *IP-Agent*, *IP-Agent-Timeout* and *IP-Agent-Cookie-Timeout*) on the path reconstruction results of a single site. However, they fail to make a recommendation considering the reliability of this methods.

For our goal of path analysis, session and user identification is mainly used for determining the specific session, the next entry belongs to and not to directly model any users. The approach is described in more detail in Algorithm 4.

### 5.3.4 Path Reconstruction

The previous section described how to distinguish different users via a heuristic using agent, ip and time information. We are now interested in reconstructing meaningful maximum forward paths (see 5.3.1) from the single accesses. While sequentially parsing the access log, we want to reconstruct a maximum number of paths. In the current log line, we extract the *time*, $IP$, *agent* and URL information ($u_{req}$ here denotes the requested URL and $u_{ref}$ the referrer field). We also introduce an augmented version of the maximum forward model paths that are used to store our results:

**Definition 11 (Traversal Path)** *We define the traversal path as the tuple* $T = (\mathcal{M}, ip, agent, t)$*, where* $\mathcal{M}$ *is a maximum forward path,* $t$ *is a list of access times for each entry in* $\mathcal{M}$ *and ip and agent are strings containing the corresponding ip and user agent information.*

Thus, a single user browsing session may result in several traversal paths. While sequentially stepping through the log data, a set of *active traversal*

*paths* $T^A = (T_1, \ldots, T_n)$ for all users is maintained. This list is updated every step by removing the entries $T$ with $time - time_T > 25\ minutes$ and adding them to the result list $\hat{T}$.
Subsequently, a set of *candidate traversal paths* $T^C \subseteq T^A$ is generated from $T^A$ by selecting all paths that have the same IP and agent information than the current log entry. The path reconstruction is now reduced to the problem which path $T$ to chose from the candidate set and where to append the new entry.
For this task, the request-referrer access pair described above is used: There are four principal possibilities how a new access pair $(u_{ref}, u_{req})$ can relate to an existing traversal path $T$. These are also shown in Figure 5.4, along with the resulting traversal paths. The following cases are:

1) $(u_{req} \notin T) \wedge (u_{ref} = \texttt{lastElem}(T))$:
   The the access pair fits at the end of the path, so we just append it.

2) $(u_{req} \notin T) \wedge (u_{ref} \in T) \wedge (u_{ref} \neq \texttt{lastElem}(T))$:
   This case is supposed to occur after a backtracking navigation. The resulting tree is split in two paths with the same prefix (see also 5.3.1).

3) $(u_{req} \in T) \wedge (u_{ref} = \texttt{lastElem}(T))$:
   If the access pair equals a backtracking navigation from the the end of the path to a previously visited page, we just omit it and wait until case 2) occurs.

4) $(u_{req} \notin T) \wedge (u_{ref} \notin T)$:
   If the access pair and the path $T$ are disjunct, we generate a new traversal path.

For every $T \in T^C$, the access pair match is now calculated considering the cases above. Since this results in many possible paths, matches for case 1) are considered to be more "valuable" for reconstruction (since they correspond to a forward navigation), than 2) and 3) (which correspond to a backtracking/split navigation). And naturally, 2) and 3) are more "valuable" than 4) (which results in a completely new path to be generated). If there are still any ambiguities, the path is chosen via a *most recently used* (MRU) policy from the available set.
A simplified version of the path extraction which has an upper bound of $O(L^2)$ with $L$ being the number of log file entries is described in pseudocode in Algorithm 4.

However, there are many subtle special cases that had to be considered in the implementation. One example is the handling of HTML frames: As

**Algorithm 4:** Path reconstruction

**Data** : access log $f$

**Result**: list of all extracted traversal paths $\hat{T}$

**while** *!eof(*f*)* **do**
  parseNextLogLine $(f, time, ip, agent, u_{req}, u_{ref})$;
  **if** *!filterEntry(*$ip, agent, u_{req}, u_{ref}$*)* **then**
    $\hat{T} \leftarrow \{T \in T^A | time - time_T < \mathsf{timeout}\}$;
    $T^A \leftarrow T^A \setminus \hat{T}$;
    $T^C \leftarrow \{T \in T^A | (ip_T = ip) \wedge (agent_T = agent)\}$;
    **if** $(T^C = \emptyset) \vee (ip \notin T^C) \vee (u_{ref} = \emptyset) \vee$ *isExtern(*$u_{ref}$*)* **then**
      $t \leftarrow$ createNewTraversalPath$(time, ip, agent)$;
      $T^A \leftarrow T^A \cup t$;
    **else**
      **if** $\{T \in T^C | u_{req} \notin T\} \neq \emptyset$ **then**
        $t_E \leftarrow \min_{T \in T^C}\{time - time_T | u_{ref} =$ lastElem$(T)\}$;
        $t_M \leftarrow \min_{T \in T^C}\{time - time_T | u_{ref} \neq$ lastElem$(T) \wedge$
        $u_{ref} \in T\}$;
        **if** $t_E \neq \emptyset$ **then**
          $t \leftarrow t_E$;
        **else**
          **if** $t_M \neq \emptyset$ **then**
            $t \leftarrow$ createSplitTraversalPath$(t_M)$;
          **else**
            $t \leftarrow$ createNewTraversalPath$(time, ip, agent)$;
          **end**
          $T^A \leftarrow T^A \cup t$;
        **end**
      **else**
        **continue** ;
      **end**
    **end**
    appendPathEntry$(t, time, u_{req}, u_{ref})$;
  **end**
**end**

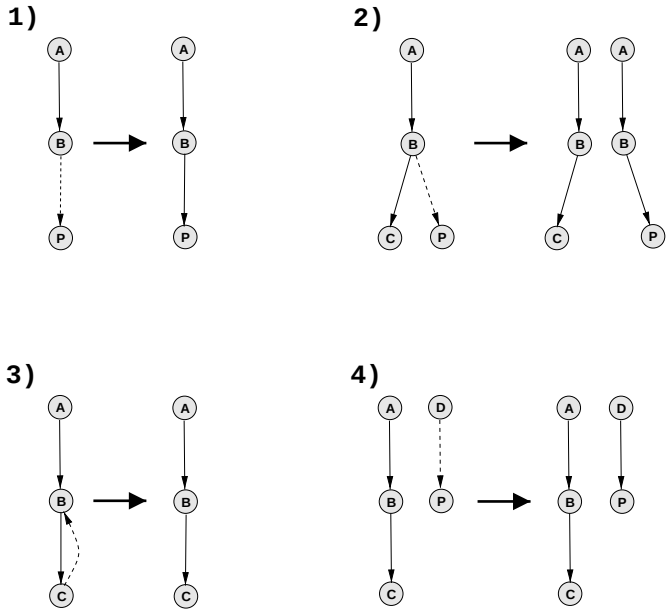


Figure 5.4: Path reconstruction cases with resulting traversal paths (dashed lines indicate the referer-access pair and solid lines an already existing path).

stated before, the access log records all HTML files that are included in a single frame. Thus we obtain a set of HTML page requests that fall into a very small time slot for a single frame request. The implementation therefore considers subsequent requests with a time difference below 1 second as the result of a HTML frame and arranges them in linear order. Special care has also been taken about the fact that the frame pages can have different, the same, or even no referrer information which forces us to make some unaesthetic adjustements in the path completion procedure.

### 5.3.5 Popular Path Extraction

After having reconstructed all available traversal paths $\hat{T}$ from the access log, the last stage of the algorithm is concerned with extracting a set of most popular paths $\bar{T}$ from $\hat{T}$ that can be used for visualization. Therefore, the extracted paths should not capture short navigation fragments that result if we employ standard data mining methods (see 5.4). Our goal is to obtain well traveled, long paths that capture navigations from the site entry on. We therefore introduce the following definition:

**Definition 12 (Most Popular Paths)** *The set of most popular paths* $\bar{T}$

*is a union of all the paths $T \in \hat{T}$ from the extracted path set that capture possibly long, well travelled navigation sequences starting with the user's site entry. Thus, it captures the areas of the Web locality that are of high interest to the users.*

Using the reconstructed traversal paths $\hat{T}$ from the last stage, we first construct a *path graph* $G_{\hat{T}}$ that captures the accumulated traversal sequences similar to a suffix tree and which is later used for analysis.

**Definition 13 (Path Graph)** *A path graph is a DAG defined by the triple $G_{\hat{T}} = (V, E, \Phi)$ with $V, E \in \hat{T}$. For every node $v \in V$, a list $\Phi_v = \{(T_{pr}, \Phi(T_{pr})) \mid T_{pr} = (p_1, \ldots, p_{m-1}) \subset T = (p1, \ldots, p_{m-1}, p_m = v, \ldots, p_n) \in \hat{T}\}$ is maintained with $\Phi(T)$ being the occurence of path $T$. Thus, the function $\Phi_v(T_{pr}) : \hat{T} \mapsto I\!N$ returns the total number of traversal paths that intersect in node $v$ and have the prefix $T_{pr}$. Figure 5.6 illustrates the concept of the path graph.*

Starting with $G_{\hat{T}} = \emptyset$, the path graph is built as follows. Every traversal path $T \in \hat{T}$ is inserted into $G_{\hat{T}}$, by adding its nodes and directed edges to the graph if they do not exist yet. An additional edge is also created from a meta-node called `SON` (start of navigation) to the first node in the path and another one from the last path entry to `EON` (end of navigation). Then, for every $v \in T = (v_1, \ldots, v_{m-1}, v_m = v, \ldots, v_n)$, $\Phi_v(T_{pr} = (v_1, \ldots, v_{m-1}))$ is incremented by one.

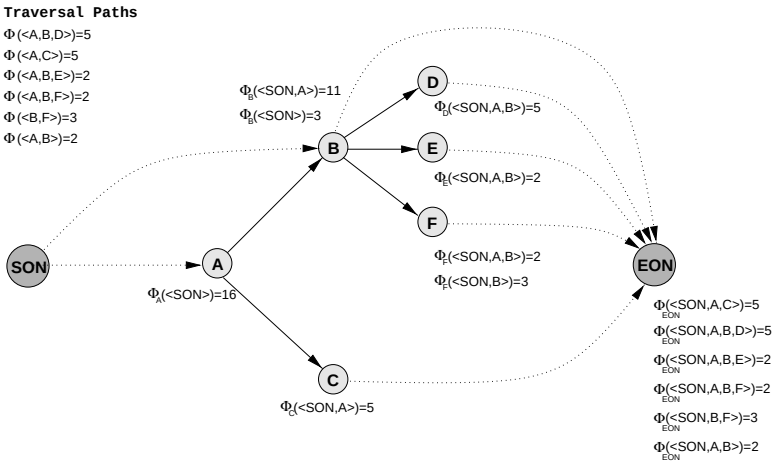


Figure 5.5: A simple path graph example

During the insert operation, information that is not needed like IP, agent and time information is ommited, since we are only interested in the actual

navigation sequence $\mathcal{M}$. In addition, we store external referrers for each node in a structure separate from $G_{\hat{T}}$ for later use and ommit referrers that contain an URL query part containing a "?" since they mostly do not represent static links but are the result of search engine queries that lead to our site. Figure 5.6 presents a path graph of the Department Website containing several hundreds of nodes. Several strategies to extract the most popular paths

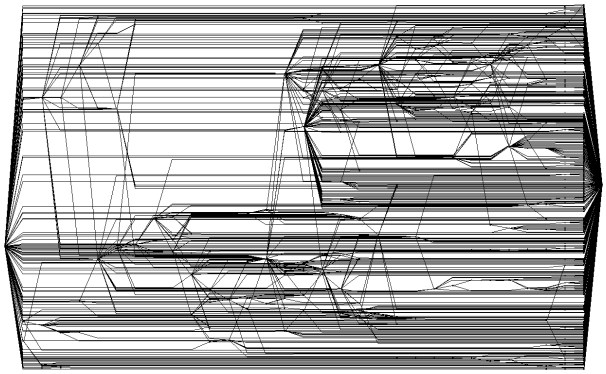

Figure 5.6: Real path graph of the Department's data set

from the path graph have been tried (see end of section). The most promising approach can be characterized as a *greedy path graph traversal* with an adaptive cutoff value. Put briefly, we begin at the start of navigation meta node and build the most popular path $P$ by traversing the graph following the most popular traversal choice with the prefix $P$ thus adding each visited node to $P$ until `EON` is reached. The extracted path is then removed from the path graph and the next most popular path is calculated using the new path graph until $G_{\hat{T}} = \emptyset$ or the number of extracted paths exceeds a certain limit. The path set $\bar{T}$ is then ordered by occurence. The whole extraction procedure which is depicted in pseudo code in Algorithm 5 will now be described in more detail.

First the successor of `SON` that has the largest number of traversal paths $\Phi_n(\texttt{SON})$ is selected as current node $n$. In the next step, $n$ is appended to the (currently empty) path $P$ which we denote by the operation $P \oplus n$. For the current node $n$, all successors $\texttt{succ}(n, P)$ that contain a path starting with the prefix $P$ are retrieved. We then use this set to calculate a cutoff value $\bar{x}$ as the mean of all paths that traverse the set of nodes:

$$\bar{x} = \frac{1}{|\texttt{succ}(n, P)|} \sum_{m \in \texttt{succ}(n,P)} \Phi_m(P) \tag{5.1}$$

If the total number of paths with prefix P that traverse $n$ and go directly to the `EON` node is larger than $\bar{x}$, we chose $n =$ `EON` as our new path traversal step. Otherwise, the successor node with the maximum number of path traversals is chosen:

$$n = \arg \max_{m \in \texttt{succ}(n,P)} \{\Phi_m(P)\} \tag{5.2}$$

The procedure above is repeated until we reach the `EON` node. Then, the constructed path $P$ represents the most popular path extracted from $G_{\hat{T}}$ and is therefore added to our list of most popular paths $\bar{T}$.
In the next step, $G_{\hat{T}}$ is modified by removing $P$. Therefore, the whole graph is examined to find all paths that have the prefix $P$ which are stored in the set $D$. For every path $T \in D$, $\Phi_n(T_p)$ is decreased by $\Phi(T)$ for each node touched and $T_p$ is successively assigned the whole sub-path range $(T_1, \ldots, T_{n-1}, T_n = T)$. If necessary, respective nodes and edges in the graph are also removed. By selecting all paths $D$ with a prefix $P$ for deletion, we collapsed all derivates of $P$ onto this path. Thus, if the total sum of the occurences of all deleted paths is larger than the occurence of the popular path $P$, we have to adjust

$$\Phi(P) = \sum_{T \in D} \Phi(T) \tag{5.3}$$

to reflect this fact. The whole path extraction procedure is then started anew till the graph $G_{\hat{T}}$ is empty. During the last step, the result set $\bar{T}$ is ordered by path occurence, such that $i < j$ for $\Phi(P_i) > \Phi(P_j)$ with $P_i, P_j \in \hat{T}$.
Figure 5.5 provides a very simplified example of some extracted traversal paths $\hat{T}$ and the resulting path graph. The most popular paths will be $\bar{T}_1$ =(`<A,B,D>`, $\Phi(\bar{T}_1) = 5$), $\bar{T}_0$ =(`<A,B>`, $\Phi(\bar{T}_0) = 6$) and $\bar{T}_2$ =(`<A,C>`, $\Phi(\bar{T}_2) = 5$).
The implemented cutoff idea is based on the assumption that for a given position $n$ in the most popular path $P$, the "popularity" or significance of the next node choice can be assessed by comparing the number of path traversals with the mean traversals of all options. Although this method is quite robust, it often results in very similar popular paths that mostly follow the main route in the site. In order to gain popular paths that are widely spread over the site, we adjust the cutoff value to be below $\bar{x}$ for shorter paths (thus considering more path options there) and to approach the original value as the path length $|P|$ increases. Thus, we now use the slightly modified cutoff value $\tilde{x}$ instead of $\bar{x}$ that is calculated via:

$$\tilde{x} = \bar{x} * \left(1 - \frac{1}{(\beta * |P|) + 1}\right) \qquad , \beta \in ]0, \infty[ \tag{5.4}$$

Since the number of extracted paths $|\bar{T}|$ is highly dependant on the specific site structure, but we want it to be in a certain range, we introduce the variable $\beta$ in the formula above in order to make the path extraction adaptive (Figure 5.7 shows the cutoff modification function for different values of $\beta$).

The path extraction algorithm is then supplemented by an outer loop that inspects the popular path set $\bar{T}$. If the number of popular paths is not in the user defined range $|\bar{T}| \in [\texttt{minPaths}, \texttt{maxPaths}]$, we adjust $\beta$ by adding $\Delta_\beta$ if $|\bar{T}| > \texttt{maxPaths}$ (and substracting otherwise). The default values are $\beta = 1.0$, $\Delta_\beta = 0.1$ with the desired range of popular paths being $[50, 100]$. The inner loop of the path extraction algorithm (see Algorithm 5) also runs in $O(L^2)$ (see before), if we consider $O(\hat{T}) = O(L)$.

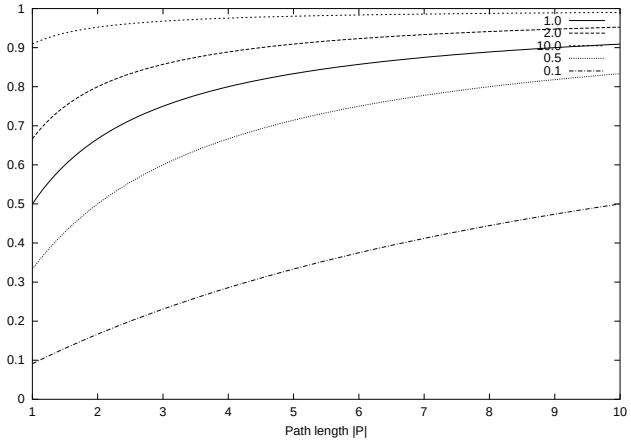


Figure 5.7: Cutoff modification function for different values of $\beta$

Finally, the whole popular path set $\bar{T}$ including the external referrers is serialized to disk using a straight-forward XML file format (see AppendixC) and can then be used for usage visualization in *YWeb* which is described in the next chapter.

### Other extraction strategies

Another basic extraction strategy was implemented that tried to obtain most popular paths simply by selecting the paths $P_i \in \hat{T}$ with the highest number of traversals $\Phi(P_i)$. However, this always resulted in very short prefix-like paths that crowded about the same space of the Web locality.

A further approch tried to rank each path by assigning a weight that captures the divergence of the actual usage from a model that would regard

**Algorithm 5:** Popular Path Extraction

**Data** : list of all reconstructed paths $\hat{T}$

**Result**: ordered list of most popular paths $\bar{T}$

$G_{\hat{T}} = \emptyset$;

**for** $\forall\, T \in \hat{T}$ **do** insert $T$ into $G_{\hat{T}}$;

$\bar{T} \leftarrow \emptyset$;

**while** $|\bar{T}| \notin [\mathit{minPaths}, \mathit{maxPaths}]$ **do**

  $G_{\hat{T}} \leftarrow$ restore original $G_{\hat{T}}$;

  $\beta \leftarrow \beta \pm \Delta_\beta$;

  $\bar{T} \leftarrow \emptyset$;

  **while** $G_{\hat{T}} \neq \emptyset$ **do**

    P $\leftarrow \emptyset$;

    $n \leftarrow \arg\max_{m \in \texttt{childs(}\mathit{SON}\texttt{)}} \{\Phi_m(\texttt{SON})\}$;

    **while** $n \neq \mathit{EON}$ **do**

      $P \leftarrow P \oplus n$;

      $\bar{x} \leftarrow \frac{1}{|\texttt{succ}(n, P)_)|} \sum_{m \in \texttt{succ}(n, P)} \Phi_m(P)$ ;

      $\tilde{x} \leftarrow \bar{x} * \left(1 - \frac{1}{(\beta * |P|) + 1}\right)$;

      **if** $\Phi_{EON}(P) > \tilde{x}$ **then**

        $n \leftarrow \texttt{EON}$;

      **else**

        $n \leftarrow \arg\max_{m \in \texttt{succ}(n, P)} \{\Phi_m(P)\}$;

      **end**

    **end**

    $\bar{T} \leftarrow \bar{T} \cup P$;

    $D \leftarrow$ find all paths with prefix $P$ in $G_{\hat{T}}$;

    **for** $\forall\, T \in D$ **do** remove $T$ from $G_{\hat{T}}$;

    **if** $\sum_{T \in D} \Phi(T) > \Phi(P)$ **then** $\Phi(P) \leftarrow \sum_{T \in D} \Phi(T)$;

  **end**

  $\bar{T} \leftarrow \forall P_i, P_j \in \bar{T};\ \ i < j \ \ :\ \ \Phi(P_i) > \Phi(P_j)$;

**end**

the link following as a normal distribution:

$$W(P) = \frac{\Phi(P)}{E(\Phi(P))} = \frac{\Phi(P) \prod_{n \in P} \texttt{outDegree}(n)}{\Phi(n_1 \in P)} \tag{5.5}$$

Unfortunately, the results were heavily distorted for paths leading through nodes of especially high out-degree (for example *Javadoc* navigation frames !).

## 5.4 Related Work

There is a great amount of access log analyzers available to the interested user via the Web. Unfortunately, most of them just focus on calculating simple statistics based on questionable metrics such as hits or page views which are quite ambigous as has been explained before and are thus less interesting in our context.

In the recent years, the data mining community has also embraced the Web as a target for knowledge discovery in the sense of *Web content mining* and *Web usage mining*. Publications such as [MS99] contain a wealth of information on this subject, but they are focussed on finding association rules in the access data that are used to make predictions like: 90% of the users that accessed `(A,B)` will access `(D,E)`. Although this kind of analysis may be useful for understanding specific navigation properties, it is not suitable for visualization.

Even fewer systems tackle the problem of extracting user paths that can be used for visualization. *Speedtracer* ([WYB98]) for example, a Web usage mining and analysis tool by IBM, uses the maximum forward path model described here, but focusses on finding frequenctly occuring consecutive subsequences. These however, do not necessarily capture large navigation paths starting with the user's site entry.

[CPP00] uses a notion of *longest repeating subsequences* (LRS) similar to the maximum forward sequences for path analysis. However, they obtain their data from cookie-enhanced log files that are more accurate than the plain ones used here.

Other path extraction prototypes like [PPR96] use the site's topology to make guesses about the best candidate path to append the current entry to.

# Chapter 6

# Usage Visualization

In this chapter, we focus on a visualization for the paths that was designed in such a way that the user can quickly grasp the usage of the site as a whole and gain an answer to the following questions: Where did most people enter the site ? Where did they travel in the site ? Where did they come from ? Since the visualization is based almost entirely on the site's access data, we can describe it as a form of *usage based layout* (UBL) that provides an alternative representation compared to the structure-based logical domains previously described.
In contrast to most existing sitemaps, we also exploit the direct neighborhood of the site in order to provide a further contextual cue for the user. In addition, interactive techniques for dynamic filtering and querying are presented for this layout together with Web-based implementation based on Java's applet technology.

## 6.1 Popular User Path Layout

The basis for user path layout is the site path model.

**Definition 14 (Site Path Model)** *The site path model* $M = (\bar{T}_M, \mathcal{C})$ *consists of a prefix of the popular path set* $\bar{T}_M \subset \bar{T}$, $\bar{T}_M = \bigcup_{i<\gamma_M} T_i \in \bar{T}$ *with* $\gamma_M$ *being user-defined.* $\mathcal{C}$ *denotes a set of configuration constants for the layout that will be introduced throughout the text.*

The basic idea is that the site administrator or creator of the usage-based map selects an appropriate range of the most popular paths and an appropriate appearance via the layout configuration. The model $M$ which is later also used in the Web-based visualization generates a view that can then be

interactively explored by the user. Following the well-known model-view-controller paradigm, we calculate a new model graph $G_M$ from $M$ and $\mathcal{C}$ after every user interaction, that is then layouted, labeled and colored.

The main part of the *popular user path layout* consists of the graph (the view) induced by the visible paths $\bar{T}_L$,

$$G_M \stackrel{\text{def}}{=} \bar{T}_L = \bigcup_{i<\gamma_L} T_i \in \bar{T}_M \tag{6.1}$$

with $\gamma_L = |\bar{T}_L|$ being the maximum number of *visible* paths in the layout. Since we want the whole path layout to fit into a limited screen estate of $\mathcal{C}_{Width} \times \mathcal{C}_{Height}$, the resulting graph $G_M$ is then subjected to a width-restricted path layout, labeling and coloring. Depending on the node selected by the user, a subgraph of $G_M$ is calculated for highlighting and external referrer nodes are added on top of the previous layout. The single layout steps described above are now further explained in the subsequent sections.

### 6.1.1 Width-Restricted Path Layout

The main part of the layout is used to display the user paths in the model via the graph $G_M$. Since the union of the popular user paths through a Web site shows a highly hierarchical property, an appropriate layout style had to be chosen:
The well-known *Sugiyama layout* (see [BETT99])is often used for the hierarchical visualization of digraphs because it is highly intuitive. The approach consists of the following three phases:
In a preprocessing step, the input digraph $G = (V, E)$ is made acyclic by temporarily reversing a subset of its edges.
During the *layer assignment phase*, $V$ is partitioned into subsets $L(G) = \{L_1, \ldots, L_h\}$ such that for every edge $(u, v) \in E$ with $u \in L_i$ and $v \in L_j$ the constraint $i > j$ is satisfied. In a further step, "dummy vertices" are inserted as follows: Each edge $(u, v)$ of span $i - j > 1$ is replaced with a path $(u = v_1, \ldots, v_k = v)$. Afterwards, every vertex $v \in L_i$ from the layered digraph is assigned the same $y$-coordinate $y_i$. The subsequent *crossing reduction phase* then orders the vertices within each layer $L$ to reduce the number of edge crossings. Finally, the *horizontal coordinate assignment phase* calculates an $x$-coordinate for each vertex such that the edge length is minimized.

The current Sugiyama implementation in the *y-files* framework that uses the polyline drawing convention, produces pleasant layouts when applied to the path graph $G_M$. However it does not optimize for the layout size which often results in huge graphs that can only be explored using zoom and pan.

Therefore, the following changes were made to the existing implementation of the algorithm:

In the first phase, the layering based on topological sorting was replaced by a layering method to minimize the width $W = \max_{1 \leq i \leq h} |L_i|$ of the largest layer: If we assume that the size of the dummy nodes is considerably smaller than the space occupied by the real nodes, $Width_{dummy} \ll Width_{real}$, (which is true in most cases since the size of the dummy nodes equals the width of the edges in the drawing), the *coffman-graham-layering* also described in [BETT99] can be used to obtain a width-minimized layering. The algorithm takes a reduced digraph $G_{red}$ and the maximum number of real nodes per layer $W$ as input and produces a layering of $G$ with width at most $W$. The reduced digraph $G_{red}$ is obtained via the algorithm presented in [Meh84] that calculates the reflexive, transitive closure $G^* = (V, E^*)$ where $E^* = \{(i, j) \in E | i \rightsquigarrow j\}$ via the closure of the reduced graph $G^* = G^*_{red}$.

Thereafter, the first phase of the coffmann-graham-layering is executed: the vertices $u \in V$ are ordered by assigning a positive integer label $\theta(u)$ to each vertex, starting with the sources in the graph. After the labels $1, 2, \ldots, k-1$ have been assigned, the next vertex $v$ is chosen such that: (1) no labels have been assigned to $v$ yet, (2) labels have been assigned for all of its predecessors $u$ with $(u, v) \in E$ and (3) the set of labels of the predecessors of $v$ $\{\theta(u) | (u, v) \in E\}$ is minimized in lexicographic order.

The second phase of the coffmann-graham-layering fills the layer with vertices, ensuring that no layer gets more than $W$ vertices. The algorithm proceeds from the bottom layer $L_1$ to top layer $L_h$ in the following fashion: For layer $L_k$, we chose a vertex that has not been placed yet and for which all successors have been placed in one of the layers $L_1, \ldots, L_k$. In case of a tie, the vertex with the largest label is selected. If there are no such vertices or the layer is full ($|L_k| = W$), the next layer $L_{k+1}$ is selected.
It has been shown ([BETT99]) that the resulting height $h$ of the layering has an upper bound of $h \leq (2 - \frac{1}{W})h_{min}$ with $h_{min}$ being the minimum possible layout height of the graph without any width restrictions.

The next modification affects the third phase of the sugiyama layout. In the current *Y* implementation, an integer linear program (ILP) is used to solve the following optimization problem that tries to minimize the space used by each layer and draw the edges as vertical as possible.
Let $C_n$ be the $x$-coordinate of vertex $n \in V$, $D_e = |C_u - C_v|$ with $e = (u, v) \in E$ be the difference in the $x$-coordinates of nodes $u$ and $v$ respectively and $w(v) = \mathcal{C}_{NodeWidth}$ the width of vertex v in pixels units. The objective function of the ILP can then be formulated as

$$\min \sum D_e \qquad , \forall e \in E \tag{6.2}$$

with the following set of constraints:

$$0 < C_n < \infty, \ \ \forall n \in V; \qquad 0 < D_e < \infty, \ \ \forall e \in E \tag{6.3}$$

$$C_v - C_u \leq D_e, \ \ \forall e \in E; \qquad C_u - C_v \leq D_e, \ \ \forall e \in E \tag{6.4}$$

$\forall L \in L(G): \ \ \forall i, j \in L : \pi(i) = \pi(j) + 1 :$

$$C_i - C_j \geq \frac{w(i)}{2} + \frac{w(j)}{2} + \delta_r = d_{ij} \tag{6.5}$$

While (6.3) restricts the $x$-coordinate, (6.4) minimizes the $x$ distance between connected nodes in separate layers. With $\delta_r = \mathcal{C}_{RealNodeDist}$ being the minimal node distance, (6.5) minimizes the space occupied by the nodes in each layer $L$ (where $\pi(i)$ denotes the index of node $i$).

In order to use this algorithm with the coffmann-graham-layering, the following modifications were introduced:
First of all, since the drawer did not take into account the (small, but existent) size of the dummy nodes, (6.5) was adjusted to take care of the four different cases regarding the sequence of dummy nodes ($V_d$) and real nodes ($V_r$):

$$C_i - C_j \geq d_{ij} = \begin{cases} 2w(j) + 2w(i) + \delta_d & : \ i \in V_d \ \wedge \ j \in V_d \\ 2w(j) + \frac{w(i)}{2} + \delta_d & : \ i \in V_r \ \wedge \ j \in V_d \\ \frac{w(j)}{2} + 2w(i) + \delta_d & : \ i \in V_d \ \wedge \ j \in V_r \\ \frac{w(i)}{2} + \frac{w(j)}{2} + \delta_r & : \ i \in V_r \ \wedge \ j \in V_r \end{cases} \tag{6.6}$$

Here, $\delta_d = \mathcal{C}_{DummyNodeDist}$ denotes the minimal distance between the dummy nodes, which otherwise are likely to be placed on the same position.
The second adjustment is the restriction of the $x$-coordinates $C_n$ by the size of the largest layer (since otherwise, the LP solver did not minimize the position correctly !). Thus the first part of equation (6.3) is replaced by:

$$0 < C_n < \max_{L \in L(G)} \left\{ \sum_{(i,j) \in L} d_{ij} \right\} \qquad , \forall n \in V \tag{6.7}$$

The aesthetic appearance of the resulting layout (see Figure 6.1) is greatly determined by the maximum number of nodes per layer $W$ and the overall

available layout size $\mathcal{C}_{Width} \times \mathcal{C}_{Height}$. For a mostly pleasant initial configuration, we estimate $W$ via

$$W = \left\lfloor \frac{\mathcal{C}_{Width'}}{\mathcal{C}_{NodeWidth} + \mathcal{C}_{RealNodeDist}} * (1 - \mathcal{C}_{\eta}) \right\rfloor \tag{6.8}$$

where $\mathcal{C}_{Width'}$ is the adjusted width taking into consideration a suitable border and $\mathcal{C}_{\eta} \in [0, 1]$ is an additional node spacing modifier which influences the "density" of the layout: If the maximum possible number of nodes per layer was chosen, the layout would result in a rectangular equidistant grid that makes it very difficult to identify tree-like properties in the graph.

In the unlikely case that the ILP can not be solved, because of a breakdown in the assumption $Width_{dummy} \ll Width_{real}$ (which may occur if $W$ is too small and thus, far too many dummy nodes are inserted), the user is notified to change the configuration $\mathcal{C}$ accordingly.
As mentioned in [BETT99], the problem of finding a layering with minimum width subject to having minimum height is NP-complete. Here, we only focus on finding a minimum width layering since the horizontal space constraint is probably more present as the map is going to be displayed next to the browser. Thus, after the width restricted layout is completed, we compare the resulting graph height with $\mathcal{C}_{Height}$ and notify the user if it is not possible to embed the model graph with the settings $\mathcal{C}$.

The path layout described above is only calculated once for the largest model graph $G_M$ with $\gamma_L = \gamma_M$. For all $\gamma_L < \gamma_M$ we use the pre-calculated node and edge positions. This has the drawback that the layout still occupies the same space when the user reduces the number of paths displayed (see 6.1.3). On the other hand, using fixed node positions greatly improves the user's ability to preserve his *mental map* of the site and makes the visualization more responsive to interaction events.

### 6.1.2 Visual Mapping

In addition to calculating node and edge positions, the appearance of the nodes and edges is chosen to reflect a mapping of information to various visual attributes like color and shape.

As can be seen in Figure 6.1, the shape of the nodes is used to distinguish between internal (round rectangles) and external (round) pages. The color of the nodes does not imply a particular information whereas the thickness of the border serves as a flag for available META information for the respective page.
The shape of the edges is also used to distinguish between internal (solid)

and external (dotted) user paths. Since we want the color of the edges to visualize the aggregate traversals of the particular link, we enhance the previous traversal definition from the last chapter as follows:

$$\Phi(e) = \sum_{\{P \in \bar{T}_L | e \in P\}} \Phi(P) \qquad , \forall e \in G_M \tag{6.9}$$

Now the traversals $\Phi(e)$ of a particular edge is given by the accumulated traversals of all *visible* paths that contain this edge. The edge color is then calculated via a linear interpolation between the current edge traversals and the maximum edge traversal value in the graph:

$$Color(e) = \frac{\Phi(e)}{\max_{e \in G_M}\{\Phi(e)\}} * (\mathcal{C}_{ColorHigh} - \mathcal{C}_{ColorLow}) \tag{6.10}$$

Finally, a special highlighting color and edge shape is used to display paths selected by the user.
The whole mapping is highly configurable as can be seen in the screenshots in Appendix B.2.3.

### 6.1.3 Interactivity

Concerning the interactivity of the layout, we distinguish between two different user types:

#### Sitemap Creator

First, the *sitemap creator* has a wide range of possibilities to modify the appearance of the layout via the *site path model editor* (see Figure B.6). As stated before, he creates and modifies the site path model $M$ that the user will later be able to explore. This includes the selection of the maximum number of nodes in the model which directly affects the maximum number of paths $\gamma_M$ from which the model is built.
By modifying the layout configuration $\mathcal{C}$, the appearance of the layout can be altered ranging from size and colors to density (B.7). He also has the possibility to export the sitemap and generate the necessary HTML code to include it into his Web presence via a single mouse click.

#### Sitemap user

Second, the interactivity for the *sitemap user* was developed for easy exploration of the site path model $M$. It mainly consists of three elements that are also available to the sitemap creator:

1. *Path Display Adjustement:* The user can dynamically adjust the number of paths $\gamma_L$ that are used to construct the layout graph $G_M$ from the model. Since the feedback from the layout is presented in under 1 sec., this feature can be used to interactively explore the most popular paths and link traversals at different levels of granularity (see Figure 6.1).

2. *Node-Path Exploration:* A further exploration of user paths is available by clicking on a specific node. The layout is then modified to emphase the subgraph induced by all paths through this node. In a second step, additional nodes and edges are added to provide information about external referrers leading to the selected node and the first node of the most popular path through it, thus providing useful context information about the user behaviour. The details of this layout augmentation are described in 6.1.4.

3. *Tooltips:* Some information of like URL, title, meta tags etc. are made available on mouse-over events by displaying a tooltip next to the corresponding node.

### 6.1.4 Node-Path Exploration Layout

During this step, the previously generated, static path layout of $G_M$ is augmented by a dynamic component that depends on user interaction.
The user selects a node $N_s \in G_M$ in order to explore its neighborhood via the popular paths taken by other users that lead through that node.

Thus in the first part of the layout augmentation, a subgraph $G_s \subseteq G_M$ is calculated as follows: A set of paths $\bar{T}_s = \{T_i \in \bar{T}_L | N_s \in T_i\}$ is defined by all visible paths that include the selected node. This set induces a subgraph $G_s = \{n, e \in G_M | n, e \in \bar{T}_s\}$ that contains only the nodes and edges of the "important" paths. In order to emphase the neighborhood $G_s$ created by this operation, the complement $G_M \setminus G_s$ is modified by using an alpha-blending technique: The RGB color values of the corresponding nodes, edges and labels are adjusted by a linear interpolation between the original and the background values.

The purpose of the second layout augmentation is to provide the user with a context about the traveled paths that goes beyond the scope of the examined Web locality: By displaying the most prominent external referrer nodes that lead to the selected node $N_s$ and the entry node of the most popular path

$T_p$, the user receives some sort of semantic context for the part of the Web presence he currently explores.
The most popular path $T_p$ of the selected path set is calculated via

$$T_p = \arg\max_{T_i \in \bar{T}_s} \{\Phi(T_i)\} \tag{6.11}$$

We also define $N_p$ as the first node of the path $T_p$. Then, two sets of external referrers that were determined during the log analysis phase are formed. $R_p$ represents the referrer nodes that are leading to the node $N_p$, while $R_s$ is formed by all referrers leading to the selected node $N_s$. In case of $N_p = N_s$, only one set $R_p = R_s$ is created.
The layout of $G_M$ from the last step is now extended to include these new nodes via a simple placement algorithm. The two sets are cropped to a size defined by the sitemap creator by removing the less visited nodes. Then the width $W$ of the bounding box for $R_p$ and $R_s$ is calculated via

$$W(R_p) = W(R_s) = |R_{p/s}| * \mathcal{C}_{NodeWidth} + (|R_{p/s}| - 1) * \mathcal{C}_{RealNodeDist} \tag{6.12}$$

The initial horizontal placement position $X$ for the referer boxes is right above the respective node since we want to minimize the edge length: $X(R_s) = X(N_s)$ and $X(R_p) = X(N_p)$. The position is then adjusted if it lies beyond the layout bounds. If the boxes intersect, we move $R_s$ into the direction of the smallest overlap. We prefer $R_s$ since we want the referrers of the most popular path entry node $N_p$ to be displayed as close to the node as possible for aesthetic reasons. If one of the referrer boxes still intersects with the layout bounds, we shift them both together by the same value such that no overlaps exist.
The vertical position of the nodes is then given by $Y(R_s) = Y(R_p) = \mathcal{C}_{YLayoutStart} - \mathcal{C}_{NodeHeight} - \mathcal{C}_{LayerDist}$. Finally, edges between each node $n \in R_{s/p}$ and $N_{s/p}$ are inserted into $G_M$ and the edge colors are calculated via the scheme described in 6.1.2.

Afterwards, the most popular path of the set $\bar{T}_s$ is highlighted using a special color and line style (see Figure 6.1).
The resulting graph is finally labeled via a greedy maximum independent set labeling provided by the *y-files* framework, that has been extended to discard candidate label positions that are not contained in the total layout bounds. The label text is determined via the heuristic presented in 4.3.
Figure 6.1 shows the complete path layout as it is generated by Algorithm 6. The initial layout generation is quite slow, due to the LP being NP-complete. In practice, this doesn't pose a problem since we only treat graphs with less than 50 nodes here. The second part runs in interactive rates and has an upper bound of $O(n)$, with $n$ being the number of nodes in the layout.

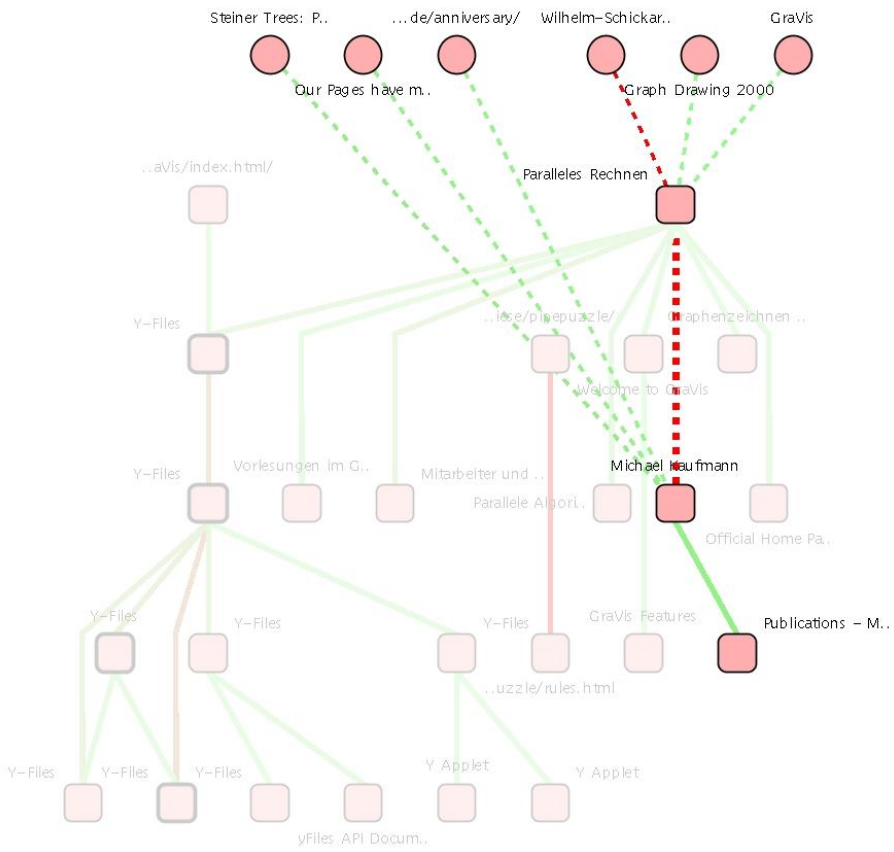


Figure 6.1: Popular path layout of the Department's site with a node selected

**Algorithm 6:** Simplified version of the popular user path layout

**Data** : Popular user path set $\bar{T}$
**Result**: Popular path layout

$\gamma_M \leftarrow$ get number of paths selected by map creator;
$G_M \leftarrow \bar{T}_M \leftarrow \bigcup_{i<\gamma_M} T_i \in \bar{T}$;
**while** *user interaction available* **do**
  $\gamma_L \leftarrow$ get number of paths selected by user;
  $G_M \leftarrow \bar{T}_L \leftarrow \bigcup_{i<\gamma_L} T_i \in \bar{T}_M$;
  calculate width-restricted layout for $G_M$;
  $N_s \leftarrow$ get node selected by user;
  **if** $N_s \neq \emptyset$ **then**
    $\bar{T}_s \leftarrow \{T_i \in \bar{T}_L | N_s \in T_i\}$;
    $G_s \leftarrow \{n, e \in G_M | n, e \in \bar{T}_s\}$;
    retrieve calculated layout positions;
    calculate edge colors;
    fade colors of $G_M \setminus G_s$;
    $T_p \leftarrow \arg\max_{T_i \in \bar{T}_s}\{\Phi(T_i)\}$;
    $N_p \leftarrow$ first node of $T_p$;
    $R_p \leftarrow$ get referers to $N_p$;
    $R_s \leftarrow$ get referers to $N_s$;
    calculate positions for $R_p$, $R_s$ and insert into $G_M$;
  **end**
  render $G_M$ with colors and layout positions;
**end**

## 6.2 Web-based Path Visualization

After an appropriate site path model $M$ has been chosen by the map creator, it is serialized to disk using an own XML format (see Appendix C).

Since the interactivity of the layout does not permit the use of pre-calculated image maps as in 4.3.3, there are basically two options to make the path visualization layout available to the user on the Web if we do not want him to install special software:

1. *Server-generated image maps* are client-side image maps that are generated on the server as the user interacts with the map. Technically, this can be realized by using a servlet engine with the installed Web server. Unfortunately, this collides with our constraint that the visualization should not require significant changes to the server setup. In addition, depending on the number of users, the process for generating

the maps can generate considerable load and thus does not scale very well.

2. *Applets* in constrast are executed on the client machine and are supported by all common browser types. The drawback is that the whole map application and data has to be sent to the user first which can result in considerable network traffic and a start-up delay before the visualization can be used.

Considering the facts above, the decision was made to provide the Web-based path visualization via an applet solution. The *YWeblet* was implemented

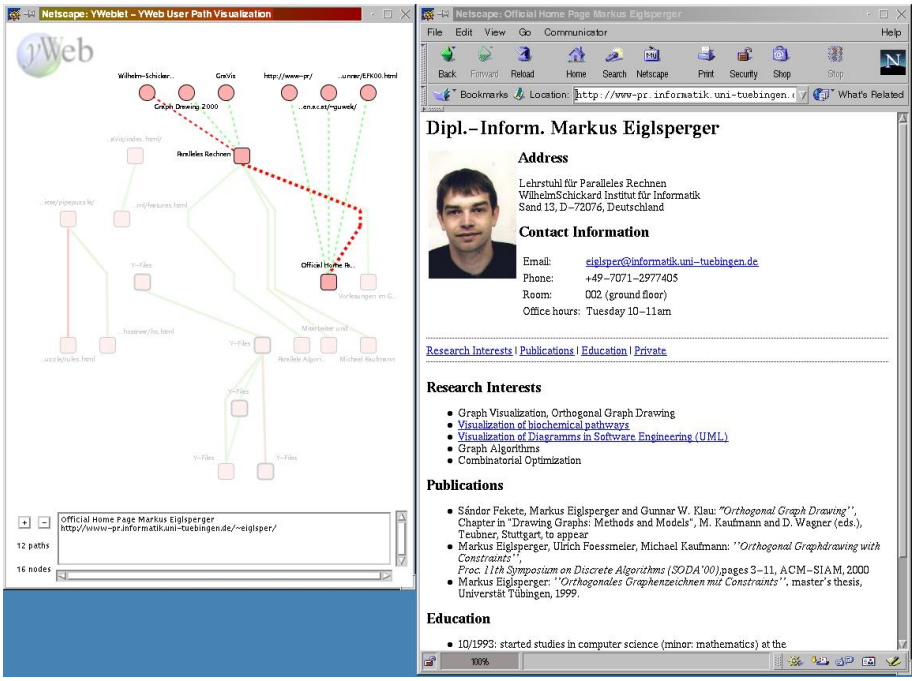


Figure 6.2: The *YWeblet* used for exploring the department Web site

using the Java 1.2 API which currently makes the one-time installation of the required plug-in necessary. Nevertheless, special care was taken to make the migration to version 1.1 (which does not require a plug-in) fairly easy, if desired. The layout and site path model code used is in fact the same that is employed in the *YWeb* application itself, with some added wrapper classes. The whole sitemap is of course delivered in a JAR archive in order to preserve download time. Figure 6.2 shows a screenshot of the applet in use alongside the Web browser.

The interactivity features do not differ a lot from the ones described in 6.1.3. The path display adjustement as well as the node-path exploration is

made available too, while tooltips have been discarded in favor of a status area at the bottom of the applet. By clicking on a node in the layout, the other browser window is instructed to load the specific Web-page.

## 6.3 Related Work

Compared with the field of hypertext and Web structure visualization, there have been only a few publications touching the visualization of user paths in a Web environment. Some work has been done on the visualization of single user paths as a graphical history and browsing feature (see [Gal00] for an overview), but this is quite different from the approach of using the accumulated paths of all users for usability analysis and navigation.

The earliest visualization for access logs *WebViz* ([PB94]) does not distinguish between paths, but presents a usage graph which is drawn via a randomized scheme using the "depth" in the unix file hierarchy and a space partitioning heuristic. The graph itself represents the whole site, with the transition frequency between nodes being mapped on the edge width. The system also allows a visualization of the traffic patterns over time.

In a recent paper ([CPP00]), the same authors use *dome trees* to visualize large Web sites and allow the overlay display of single user paths. They also introduce a usage-based layout that is operates on building a tree using the site structure and a priority-based traversal based on usage data that is later displayed as dome or disk tree. This method has a sharp focus on site analysis and is not suitable as single user navigation help since the amount of displayed information is overwhelming and requires a great deal of interface events.

[CS99] presents a 3D visualization tool called *VISVIP* that can be used by site administrators to explore paths traveled by individual subjects. The path data is not extracted from the access logs, but recorded by an instrumented browser during usability experiments. For visualization, a 2D layout using a force-directed algorithm is used, with the path of the subject being superimposed on the drawing via a spline. The third dimension is occupied by a bar for each node, representing the time the user spent on each node. They also provide an animated representation of progress along a path through the Web site.

*INsite* [FSMF99] uses the connectivity matrices defined by the aggregates of several user paths through clusters of pages for presentation. Unfortunately, this compact form of visualization is hard to interpret for users that are not familiar with the respective site.

The only system that also supplies a Web-based visualization for aggre-

gate user paths is *Footprints* [WM97], which is a set of tools that explores interaction history to generate several of layouts for site navigation that are based on a physical-world metaphor. Their path map depicts typical paths through the current document in a stair-step fashion. This is done via a Java applet that encodes the degree of use as lines of varying width through the single nodes that should be thought of like cities, with the paths being highways through the nodes.

Unfortunately most of the systems described above are research prototypes that are not available to the public and have been tested only on a very limited data set.

# Chapter 7

# Conclusions and Future Work

We have presented an automatic navigation sitemap generation system that fulfills all of our self-imposed design requirements established in 2.4.1. Nevertheless, there is still a lot of work do be done in terms of robustness, interactivity and visualization.
However, in comparison to most systems described in 2.3, our prototype can be considered to be novel in respect to the following features:

1. *Real cross-plattform* capability by solely relying on the *Java* and *y-files* API.

2. *Open system* through XML data exchange standards and code availability.

3. *New structure-based navigation sitemap approach* by using logical domains as the basis of visualization and interaction in navigation sitemaps.

4. *New usage-based navigation sitemap approach* by using popular user paths through the locality as the basis for visualization and interaction.

5. *Large Web localities supported* with up to 15.000 nodes.

6. *Robustness* by using a variety of different Web site types for testing (as opposed to only educational type sites used by most research prototypes).

We now present some of the results and problems concerning the data extraction, the clustering/structuring, the visualization and the Web site map tier (see 2.6):

### Data Extraction

Regarding the structure extraction tier, the simple *YWebCrawler* is currently taking up to several hours for a large site of several thousand nodes in order to extract structural data from the Web locality. This could of course be improved by providing a real multi-threaded implementation. In the current version, the crawler also does not support the crawling of hyperlinks embedded in *Javascript* and other scripting languages. Nevertheless, it proved to be highly reliable concerning the crawling of dynamically generated HTML pages as long as no interactive content like forms etc. was encountered.

The *YWeblog* popular path extraction component provided the source for useful insight into the site's usage. In order to further improve its reliability, techniques like IP and host munging should be employed. Besides, we did not yet present metrics to actually assess the quality of the extracted paths. This was done by manually examining the log files and inferring the user patterns and popular paths through correlation with the site structure.
Nevertheless, the path extraction was successfully used, with access log data files $>$ 1GB taking about half an hour to complete.

Concerning the server access logs as a data source, it has to be mentioned however that reliable usage data will be harder to come by in the future due to the increasing use of anonymizer services (such as crowd proxies or *SafeWeb*[1]). Also, future implementations of the IPv6 protocol will support the dynamic change of IP addresses during accesses, thus making identification even more complicated.

### Clustering

Currently, only the logical domain component uses some kind of clustering technique for data aggregation. We observed that the empirically determined scoring functions work surprisingly well for a large variety of different sites. Nevertheless, it could be worth while to establish presets for different genres of sites such as academic, personal or commercial ones in order to improve its accuracy. Another interesting approach would be the use of adaptive techniques such as machine learning for determining a suitable standard configuration. Both tasks however will require a large number of Web localities to be sampled in order to obtain general results.
Due to its straight forward clustering procedure, the logical domain assignment takes less than five minutes even for the largest site examined.

[1] `http://www.safeweb.com`

### Layout and Visualization

The logical domain tree-map layout proved to be highly interesting during informal usability studies mostly because of its ease of use and speed of interaction. However, it could still be improved in order to provide further interactivity by zooming into selected logical domains, making it possible to alter the level of granularity or use dynamic filtering techniques. Another approach would be to present several pre-calculated domain maps to the visitor that were generated according to different metrics in order to support distinct navigation tasks like finding the newest content on a site, finding the largest sub-content, the hottest product etc. Some work regarding a more visual appealing layout could also be done, especially in respect to vertical labeling.

On the other hand, the popular path layout provided a first excursion into the realm of usage based layouts. It proved to be of special interest by providing information about the close external neighborhood where users actually travel from to this site by visualizing the external referrers. Another highly valued feature is the dynamic filtering technique that allows the user to select a level of granularity by modifying the number of visible paths.

The quality of layout of the graph induced by the selected popular paths is hard to assess since there is (to our knowledge) no other visualization available that uses the same data in this context. For an immediate improvement however, the coffmann graham layering could be adjusted to consider the size of the dummy nodes. Another interesting width restricted layering method that considers the dummy nodes is also presented in [BELM00]. During implementation, several short-comings of the *y-files*, especially the fixed draw order regarding nodes and edges proved to be challenging in providing visually appealing layouts.

It has to be stated that both layout generation methods actually take less than a minute, with the popular path layout being available at interactive rates after the calculation of the initial layout.

### Web-based Sitemaps

The most appealing features of the Web-based sitemaps are the high configurability and the fast "one-click" generation and export that are especially important for the "casual" site administrator.
Still, a lot of work could be done in order to preserve more bandwith and reduce the startup times of the visualization, especially in respect to the path layout applet. The migration to Java 1.1 compatible API functions should also be considered since it is supported by every browser type. However, in

this context, the *y-files* framework would first have to be migrated because it makes extensive use of Java 1.2 collection classes.

### Future Work

Several important topics for future work on the navigation sitemap generation system as a whole can be identified: First of all, the problem of URL ambiguity through soft links has to be resolved by calculating a fingerprint (for example the MD5 hash) of the specific page. In addition, new techniques for the handling of HTML frames have to be investigated that perhaps use a kind of clustering technique in order to group pages together that are related by one HTML frame. Another important topic for research is the support for crawling dynamically generated content that includes interactive items like forms etc. which is certainly a challenge for itself.
Concerning the usage data extraction, metrics will have to be developed and evaluated that assess the actual quality of the extracted paths and the resulting popular user paths in order to compare different path reconstruction methods. The system itself offers a great deal of further possibilities in increasing interactivity during the sitemap construction process. As an example, visual editing procedures like manually deleting and inserting logical domains could be added.

On the sitemap and visualization side, formal usability tests should be conducted in order to determine their real value to the end user. In addition, new visual metaphors could be investigated that allow the user to assess which parts of the page he already visited and present pages that are of possible interest to him by including a recommendation system based on clustered user sessions. Besides, the further use of animation techniques is proposed in order to make the layouts more immersive and visually appealing.

Regarding the field of Web site mapping as a whole, the following topics are especially interesting for investigation:
First, new visual metaphors and clustering approaches will have to be developed in order to gain further insight into useful and navigationable representations for Web localities. Particular interesting could be the application of techniques from *social network analysis* ([Zah01]) for the analysis of intra- and inter-site structure. Another hot topic is the visualization of time-dependant data e.g. the evolution of site structures or traffic through time ([CPP00]).

With the advent of Web sites providing automated services for other sites and software agents via the Web (see *Microsoft's .NET* framework[2]), the Web is evolving towards a huge distributed computing platform. More powerful visualization techniques will be needed in order to cope with the growing complexity of the resulting *semantic Web* and the ever increasing volume of inter- and intra-site data.

[2] `http://msdn.microsoft.com/net/`

# Appendix A

# Web Locality Data Sets

The evaluation of the presented algorithms was conducted on a *SUN Ultra-Sparc 10* with 300 MB of memory, using five highly diverse Web locality data sets that will now be described in order to present their different properties.

### *Ouk Magazine*

The *Ouk Magazine*[1] Web site is a "net-mag" of monthly appearing Ouk Magazines for electronic music enthusiasts. Most of the site contains the actual monthly issues, while a small part is reserved for a discussion group and news announcements. The locality itself is quite small with currently about 400 pages and 700 hyperlinks and does not include advanced features like frames or Javascript. The corresponding access logs that were used for analysis range from 5 to 8 MB per month.

### *Parallel Computing Group*

*The University of Tübingen Parallel Computing Group* Web site[2] is a typical academic locality that does not include much manually created content. In fact, a huge part of the site consists of *JavaDoc*-generated class documentations for the group's graph visualization framework called *y-files* that extensively uses HTML frames. The whole site therefore contains about 2000 pages in a whole. The access log data used for path analysis was actually quite small with about 3 MB/month.

### *Department of Computer Architecture*

The *University of Tübingen's Dept. of Computer Architecture* Web site[3] is an advanced academic site that also contains a large online tutorial about the SNNS software as well as several online books (these were not included in the analysis). The site which makes extensive use of frames contains about 4400 nodes and 11204 edges and provides about 60 MB of access log data per month.

### *Simon V's Homepage*

*Simon V's Homepage* Web site[4] is an ambitious artist homepage that provides a lot of music and picture downloads as well as several discussion boards. It can be categorized as highly dynamic because it makes heavy use of scripting mechanisms and generates most pages on the fly. It consists of about 1300 pages with 12000 hyperlinks and provides about 4 MB of access log data per month.

### *Native Instruments Software Synthesis GmbH*

The *Native Instruments* Web site[5] is a bilingual (english/german), commercial locality that consists of several product displays, a large community, trial downloads and of course an online shop.
The site is only about 800 nodes and 4000 edges in extent, but provides 1.5 GB of access log data per month. It makes heavy use of Javascript and does require the HTTPS protocol for some pages.

[1] `http://www.ouk.de`
[2] `http://www-pr.informatik.uni-tuebingen.de`
[3] `http://www-ra.informatik.uni-tuebingen.de`
[4] `www.simonv.com`
[5] `http://www.native-instruments.de`

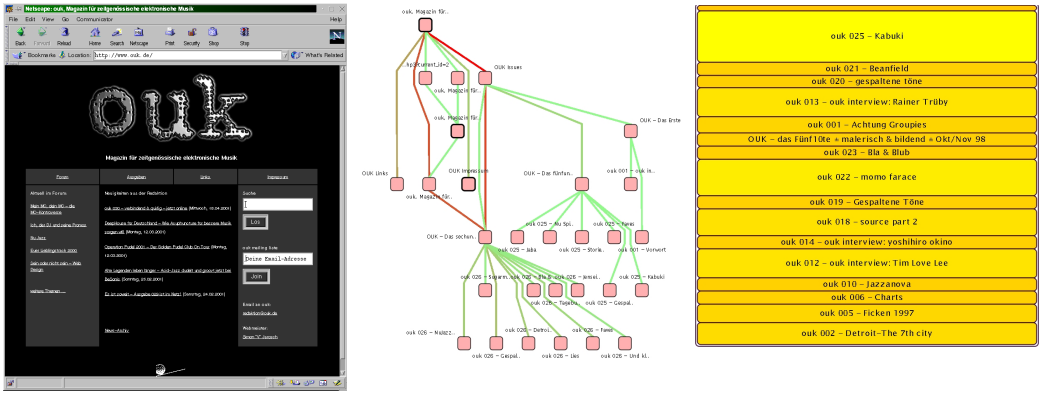


Figure A.1: *Ouk Magazine* Web site and sample sitemaps

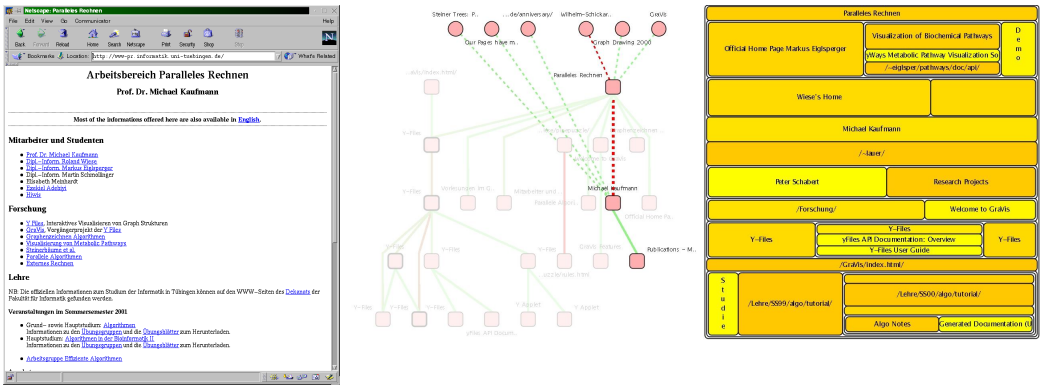


Figure A.2: *Parallel Computing Group* Web site and sample sitemaps

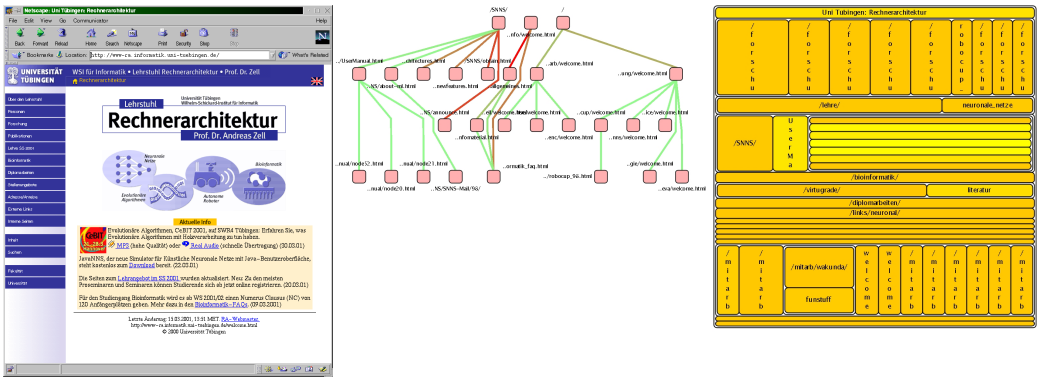


Figure A.3: *Dept. of Computer Architecture* Web site and sample sitemaps

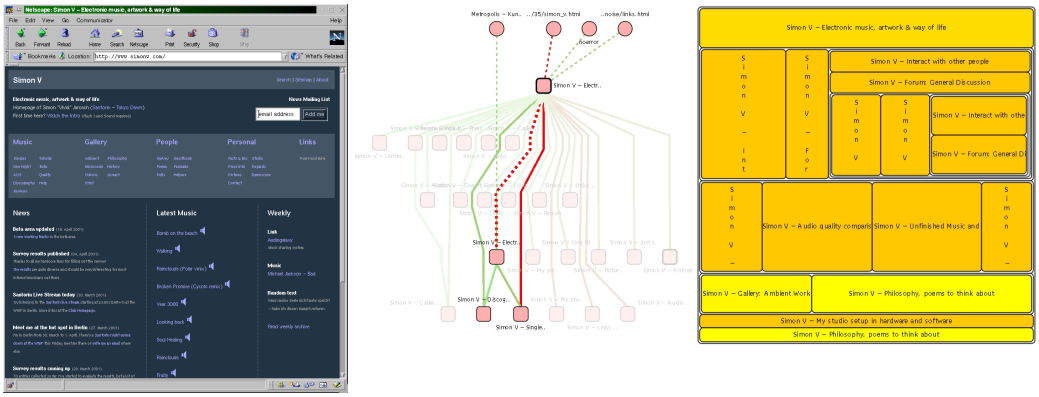


Figure A.4: *Simon V's Homepage* Web site and sample sitemaps

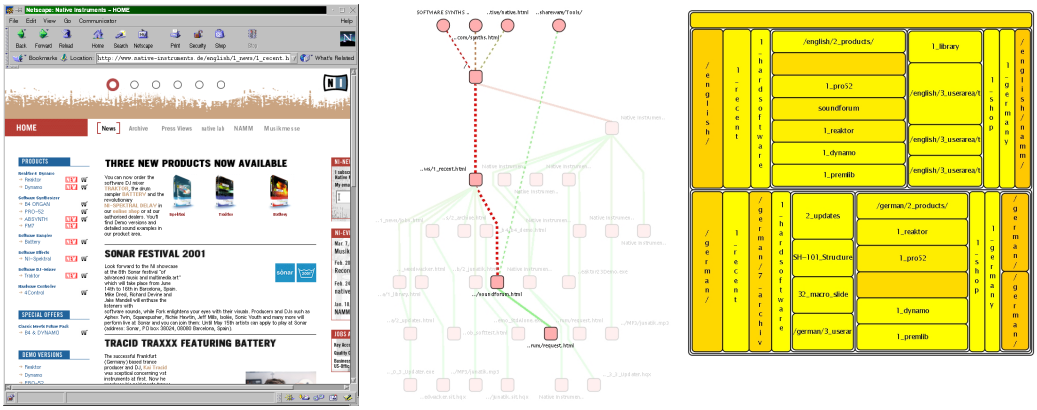


Figure A.5: *Native Instruments* Web site and sample sitemaps

# Appendix B

# Software Package Description

## B.1 Required Software Packages

The following software is required by various parts of the *YWeb* system:

#### ***Y-Files*** **Framework**

*Y-Files*[1] is a freely available, powerful library for viewing, editing, layouting and animating graph-like structures developed at the Parallel Computing Group at the University of Tübingen. The library is written in Java and runs on every computer with a Java2 VM properly installed.

#### **Apache *Xerces* XML parser**

*Xerces*[2] is a freely available, powerful XML parser that is used for loading the various SUG and SSG file formats.

#### **Java Secure Sockets Extension (J2SEE)**

The *Java Secure Sockets Extension*[3], available for Windows and Solaris type machines is required in order to enable the crawling of sites via the secure HTTPS protocol which is suggested when analyzing online shops.

[1] `http://www-pr.informatik.uni-tuebingen.de/yfiles/`
[2] `http://xml.apache.org/xerces-j/`
[3] `http://java.sun.com/products/jsse/`

## B.2 Components

The *YWeb Sitemap Generation System* mainly consists of three seperate software components: The *YWebCrawler* is used for extracting the structural properties of the specific Web locality, while the *YWebLog* component extracts and analyzes usage data. Finally, the *YWeb* component itself provides an interactive application for exploring the data and generating suitable navigation sitemaps based on structure and usage.

### B.2.1 *YWebCrawler*

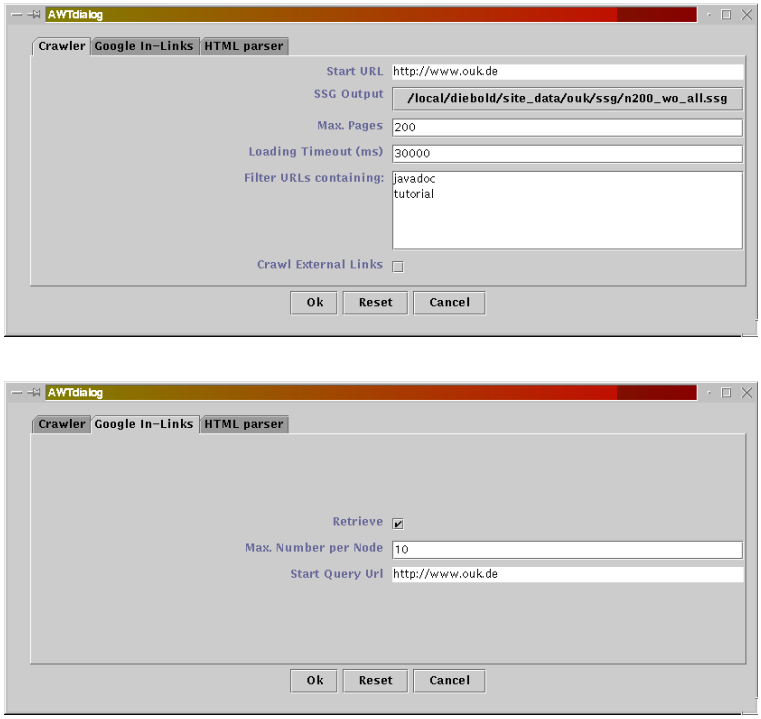


Figure B.1: Crawler (top) and Google link extraction (bottom) dialogs

### B.2.2 *YWebLog*

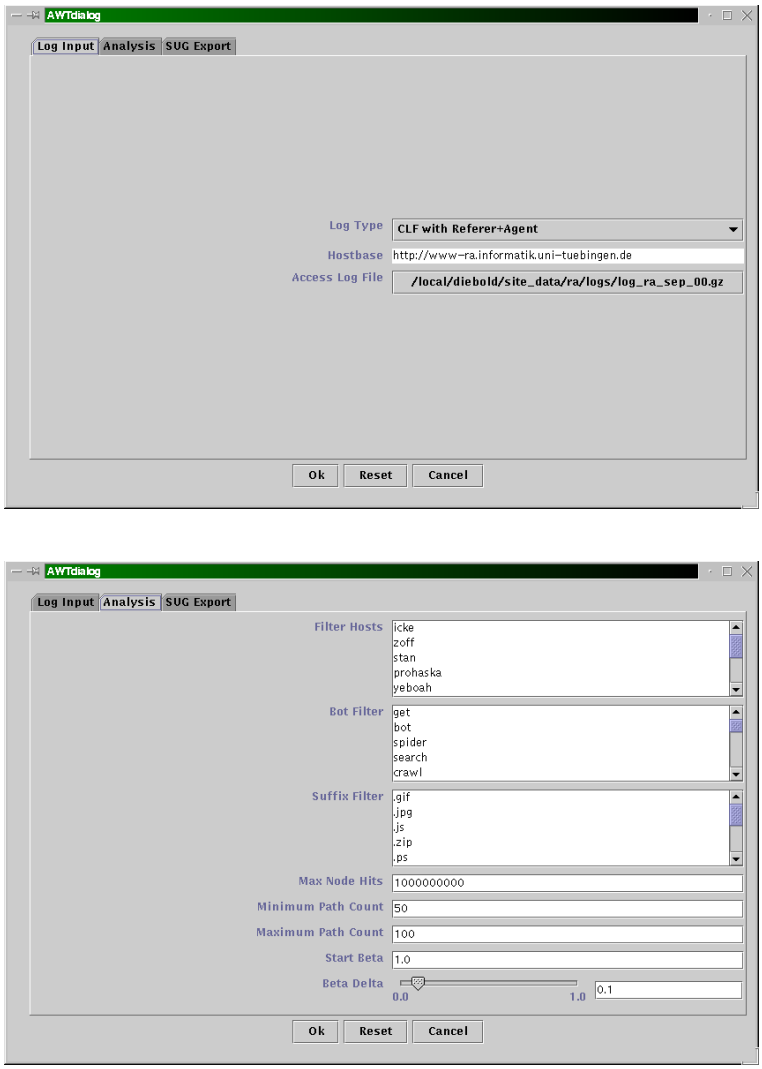


Figure B.2: Log input (top) and analysis (bottom) dialogs

### B.2.3 *YWeb*

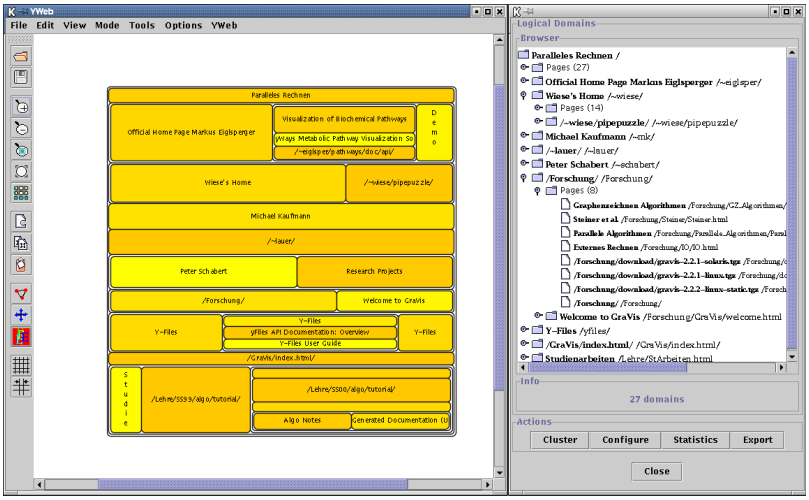


Figure B.3: *YWeb* application operating in logical domain mapping mode with the logical domain browser and editor displayed to the right

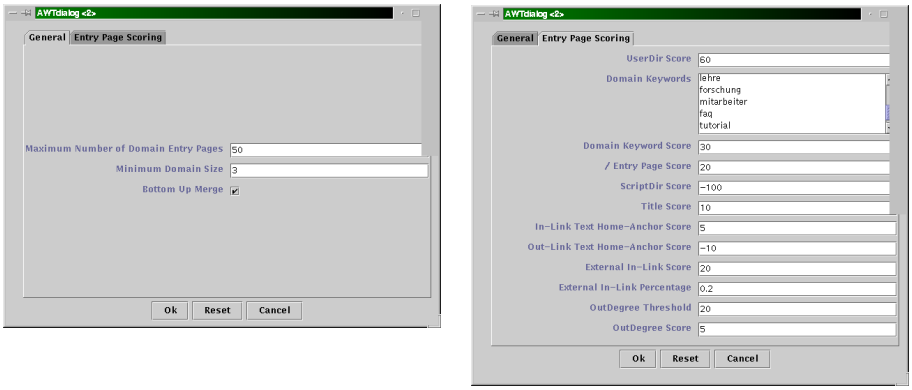


Figure B.4: Logical domain clustering dialogs

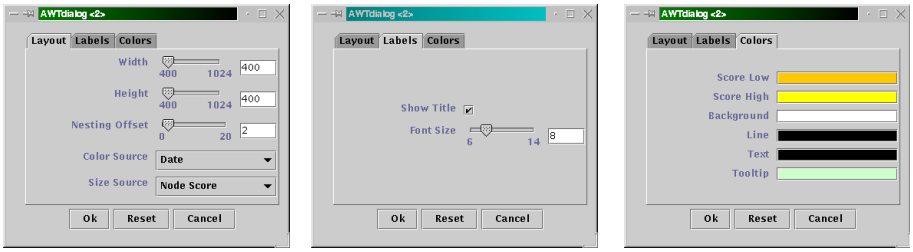


Figure B.5: Logical domain mapping configuration dialogs

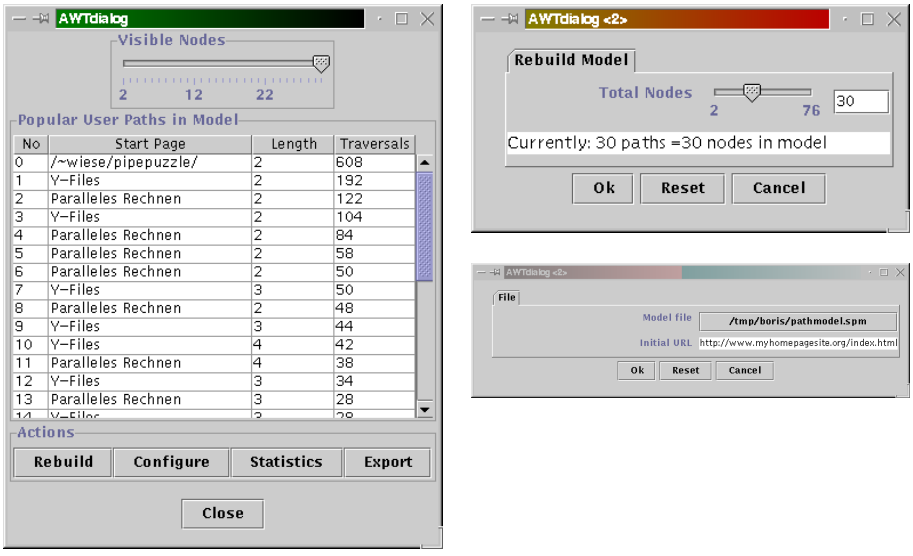


Figure B.6: Popular path layout editor, build and export dialogs

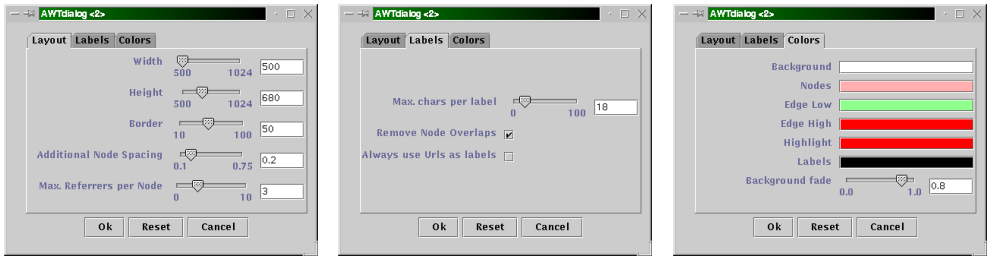


Figure B.7: Popular path layout mapping configuration dialogs

# Appendix C

# Document Type Definitions

## Site Structure Graph (SSG.DTD)

```
<!-- SITESTUCTUREGRAPH DTD v1.2 -->
<!ELEMENT SITESTRUCTUREGRAPH (NODELIST,EDGELIST)>
<!ELEMENT NODELIST (NODE*)>
<!ELEMENT EDGELIST (EDGE*)>

<!-- NODE -->
<!ELEMENT NODE (URL,CSIZE,CDATE?,CTYPE,EXTRN,ELINK?,MDESC?,MKEYS?,TITLE?)>
<!ATTLIST NODE
   NODEID ID #REQUIRED
   TYPE CDATA #REQUIRED>

<!-- GENERIC NODE ELEMENTS -->
<!ELEMENT URL (#PCDATA)>
<!ELEMENT CSIZE (#PCDATA)>
<!ELEMENT CTYPE (#PCDATA)>
<!ELEMENT CDATE (#PCDATA)>
<!ELEMENT EXTRN (#PCDATA)>
<!ELEMENT ELINK (#PCDATA)>

<!-- HTML NODE ELEMENTS -->
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT MKEYS (#PCDATA)>
<!ELEMENT MDESC (#PCDATA)>

<!-- EDGE -->
<!ELEMENT EDGE (NAME)>
<!ATTLIST EDGE
   EDGEID ID #REQUIRED
   SOURCE IDREF #REQUIRED
   DEST IDREF #REQUIRED>

<!-- EDGE ELEMENTS -->
<!ELEMENT NAME (#PCDATA)>
```

## Site Usage Graph (SUG.DTD)

```
<!-- SITE USAGE GRAPH DTD v1.1 -->
<!ELEMENT SITEUSAGEGRAPH (NODELIST,EDGELIST,(PATHLIST?))>
<!ELEMENT NODELIST (NODE*)>
<!ELEMENT EDGELIST (EDGE*)>
<!ELEMENT PATHLIST (PATH*)>

<!-- NODE -->
<!ELEMENT NODE (URL,EXTRN,HITS)>
<!ATTLIST NODE
   NODEID ID #REQUIRED>

<!-- GENERIC NODE ELEMENTS -->
<!ELEMENT URL (#PCDATA)>
<!ELEMENT EXTRN (#PCDATA)>
<!ELEMENT HITS (#PCDATA)>

<!-- EDGE -->
<!ELEMENT EDGE (HITS)>
<!ATTLIST EDGE
   EDGEID ID #REQUIRED
   SOURCE IDREF #REQUIRED
   DEST IDREF #REQUIRED>

<!-- PATH PATH -->
<!ELEMENT PATH (TRAV,NODES)>

<!ELEMENT TRAV (#PCDATA)>
<!ELEMENT NODES (NR*)>
<!ELEMENT NR (#PCDATA)>
```

## Site Path Model (SPM.DTD)

```
<!-- SITE PATH MODEL DTD v1.1 -->
<!ELEMENT SITEPATHMODEL (CONFIG,NODELIST,REFERERLIST,PATHLIST)>

<!ELEMENT NODELIST (NODE*)>
<!ELEMENT REFERERLIST (EDGE*)>
<!ELEMENT PATHLIST (PATH*)>
<!ELEMENT CONFIG (LAYOUTWIDTH,LAYOUTHEIGHT,LAYOUTBORDER,NODESPACING,
                  MAXIMUMREFERERNODE,LABELCHARS,REMOVENODEOVERLAPS,
                  USEURLLABELS,BACKGROUND_C,NODE_C,EDGES_C,EDGEE_C,
                  HIGH_C,LABEL_C,FADE)>

<!-- CONFIG ELEMENTS -->
<!ELEMENT LAYOUTWIDTH (#PCDATA)>
<!ELEMENT LAYOUTHEIGHT (#PCDATA)>
<!ELEMENT LAYOUTBORDER (#PCDATA)>
<!ELEMENT NODESPACING (#PCDATA)>
<!ELEMENT MAXIMUMREFERERNODE (#PCDATA)>
<!ELEMENT LABELCHARS (#PCDATA)>
<!ELEMENT REMOVENODEOVERLAPS (#PCDATA)>
<!ELEMENT USEURLLABELS (#PCDATA)>
<!ELEMENT BACKGROUND_C (#PCDATA)>
<!ELEMENT NODE_C (#PCDATA)>
<!ELEMENT EDGES_C (#PCDATA)>
```

```
<!ELEMENT EDGEE_C (#PCDATA)>
<!ELEMENT HIGH_C (#PCDATA)>
<!ELEMENT LABEL_C (#PCDATA)>
<!ELEMENT FADE (#PCDATA)>

<!-- NODE -->
<!ELEMENT NODE (TITLE,URL,HITS*,MKEYS*,MDESC*)>
<!ATTLIST NODE NODEID ID #REQUIRED>

<!-- GENERIC NODE ELEMENTS -->
<!ELEMENT URL (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT HITS (#PCDATA)>
<!ELEMENT MKEYS (#PCDATA)>
<!ELEMENT MDESC (#PCDATA)>

<!-- REFERERS  -->
<!ELEMENT REFERERS (NODES)>
<!ATTLIST REFERERS SRCID ID #REQUIRED>

<!-- PATH PATH -->
<!ELEMENT PATH (TRAV,NODES)>

<!ELEMENT TRAV (#PCDATA)>
<!ELEMENT NODES (NR*)>
<!ELEMENT NR (#PCDATA)>
```

# List of Figures


2.1 PowerMapper . . . . . 21
2.2 *SiteBrain* sitemap integrated in a Web page . . . . . 21
2.3 Sitemap generated by *Inxight Tree Studio* . . . . . 23
2.4 Sitemap generated by *WebToc* . . . . . 23
2.5 Sitemap generated by *Mapa* . . . . . 24
2.6 *YWeb* sitemap system architecture . . . . . 27

3.1 Logical domain cluster tree . . . . . 36

4.1 Force-directed topology layout . . . . . 42
4.2 Zoomed force-directed topology layout . . . . . 43
4.3 2D cone-tree layout schema . . . . . 44
4.4 2D cone-tree layout of the *ouk* data set . . . . . 45
4.5 Conversion of an attributed tree into a tree-map . . . . . 45
4.6 Logical domain tree conversion . . . . . 47
4.7 Logical domain treemap for the *Department* data set . . . . . 48
4.8 Web-based logical domain tree-map . . . . . 50

5.1 Sample Web server access log entry . . . . . 56
5.2 Chain of request during Web server access . . . . . 57
5.3 Maximum forward reference path construction . . . . . 60
5.4 Path reconstruction cases . . . . . 65
5.5 A simple path graph example . . . . . 66
5.6 Real path graph of the Department's data set . . . . . 67
5.7 Cutoff modification function . . . . . 69

6.1 Popular path layout . . . . . 81
6.2 The *YWeblet* used for exploring the department Web site . . . 83

A.1 *Ouk Magazine* Web site and sample sitemaps . . . . . 95
A.2 *Parallel Computing Group* Web site and sample sitemaps . . . 95
A.3 *Dept. of Computer Architecture* Web site and sample sitemaps 95
A.4 *Simon V's Homepage* Web site and sample sitemaps . . . . . 96

A.5 *Native Instruments* Web site and sample sitemaps . . . . . 96

B.1 Crawler and Google link extraction dialogs . . . . . 98
B.2 Log input and analysis dialogs . . . . . 99
B.3 *YWeb* application operating in logical domain mapping mode 101
B.4 Logical domain clustering dialogs . . . . . 102
B.5 Logical domain mapping configuration dialogs . . . . . 102
B.6 Popular path layout editor, build and export dialogs . . . . . 102
B.7 Popular path layout mapping configuration dialogs . . . . . 103

# List of Algorithms


1 Web Crawler . . . . . 31
2 Logical domain clustering . . . . . 37
3 layoutNode(Node $n$, Orientation $o$, Rectangle $r$) . . . . . 46
4 Path reconstruction . . . . . 64
5 Popular Path Extraction . . . . . 70
6 Simplified version of the popular user path layout . . . . . 82

# Bibliography


[Abr97] D. Abrams. How People use WWW Bookmarks. In *Procceedings of the ACM Conference on Computer-Human Interaction*, 1997.

[BELM00] J. Branke, P. Eades, S. Leppert, and M. Middendorf. Width Restricted Layering of Acyclical Digraphs with Consideration of Dummy Nodes. Technical Report No 403, Insitute AIFB, University of Karlsruhe, 2000.

[BETT99] G. Battista, P. Eades, R. Tamassia, and I. Tollis. *Graph Drawing - Algorithms For The Visualization of Graphs*. Prentice Hall, 1999.

[BRS92] R. Botafogo, E. Rivlin, and B. Shneiderman. Structural Analysis of Hypertexts: Identifying Hierarchies and Useful Metrics. *ACM Transactions on Information Systems*, 10(2):142–180, April 1992.

[BS91] R. Botafogo and B. Shneiderman. Identifying Aggregates in Hypertext Structures. In *Proceedings of the third annual ACM conference on Hypertext*, pages 63–74, San Antonio, TX USA, 1991.

[Che97] C. Chen. Structuring and Visualising the WWW by Generalised Similarity Analysis. In *Proceedings of the Eigth ACM Conference on Hypertext and Hypermedia*, pages 177–186, Southhampton, UK, 1997.

[CK95] J. Carriere and R. Kazman. Interacting with Huge Hierarchies: Beyond Cone Trees. In *Proceedings of the IEEE Information Visualization Conference '95*, pages 74–81, 1995.

[CMS99] S. Card, J. Mackinley, and B. Shneiderman. *Readings in Information Visualization - Using Vision to Think*. Morgan Kaufmann Publishers, 1999.

[Con87] J. Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20:17–41, 1987.

[CPP00] E. Chi, P. Pirolli, and J. Pitkow. The Scent of a Site: A System for Analyzing and Predicting Information Scent, Usage, and Usability of a site. In *CHI 2000 - Human Factors in Computing Systems*, pages 161–168, NYC, USA, 2000.

[CS99] J. Cugini and J. Scholtz. VISVIP: 3D Visualization of Paths through Web Sites. In *Proceedings of the International Workshop on Web-Based Information Visualization*, pages 259–263, Florence, Italy, 1999.

[Dod00] M. Dodge. *Cyber-Geography Research.* Centre for Advanced Spatial Analysis, University College London, Octobre 2000. Available at `http://www.cybergeography.org` [as of 11-03-2000].

[FSMF99] A. Faisal, C. Shahabi, M. McLaughlin, and F.Betz. INsite: Introduction to a generic paradigm for interpreting user-web space interaction. In *Proceedings of the second international workshop on Web information and data management*, pages 53–58, Kansas City, MO USA, 1999.

[Gal00] Martin Gall. Y-Nansen: Tool zur graphischen Darstellung benutzerbedingter Internet-Histographien (Surf-Maps). Master's thesis, University Tübingen, WSI Computer Science Dept., February 2000.

[Har69] F. Harary. *Graph Theory.* Addison-Wesley, 1969.

[Hen00] M. Henzinger. Web Information Retrieval - an Algorithmic Perspective. In *Algorithms-ESA 2000: Proceedings of the 8th annual European Symposium*, LNCS 1879, pages 1–8, Saarbrücken, Germany, September 2000.

[HGM00] I. Herman and M. Marshall G. Melançon. Graph Visualization and Navigation in Information Visualisation: A Survey. *IEEE Transactions on Visualization & Computer Graphics*, 6:24–43, 2000.

[HN99] A. Heydon and M. Najork. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web*, 2(4):219–229, December 1999.

[HY00] D. Harel and G. Yashchin. An Algorithm for Blob Hierarchy Layout. In *Proceedings of the 2000 ACM Conference on Advanced Visual Interfaces*, pages 29–40, Palermo, Italy, May 2000.

[LG99] S. Lawrence and C. Giles. Accessibility of Information on the Web. *Nature*, 400:107–109, 1999.

[LKVT00] W. Li, O. Kolak, Q. Vu, and H. Takano. Defining Logical Domains in a Web Site. In *Proceedings of the 2000 ACM Hypertext Conference*, pages 123–132, San Antonio, TX, August 2000.

[LR96] L. Lamping and R. Rao. The Hyperbolic Browser: A focus + context technique for visualizing large hierarchies. *Journal of Visual Languages and Computing*, 1:33–55, 1996.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, pages 7–10. EATCS, Monographs on Theoretical Computer Science. Springer Press, 1984.

[MFH95] S. Mukherjea, J. Foley, and S. Hudson. Visualizing Complex Hypermedia Networks through Multiple Hierarchical Views. In *Conference proceedings on Human Factors in Computing Systems*, pages 331–337, Denver, CO USA, 1995.

[MH97] S. Mukherjea and Y. Hara. Focus+Context Views of World-Wide Web Nodes. In *Proceedings of the Eigth ACM Conference on Hypertext and Hypermedia*, pages 187–196, Southhampton, UK, 1997.

[MM97] D. Modjeska and A. Marsh. Structure and Memorability of Web sites. Working Paper in the Department of Industrial Engineering, University of Toronto, 1997.

[MS99] B. Masand and M. Spiliopoulou, editors. *International WEBKDD99 WorkShop: Web Usage Analysis and User Profiling*, Lecture Notes in Artificial Intelligence (LNCS 1836), 1999.

[Muk99] S. Mukherjea. Information Visualization for Hypermedia Systems. *ACM Computing Surveys*, 31(4), December 1999.

[PB94] J. Pitkow and K. Bharat. WebViz: A Tool For World-Wide Web Access Log Analysis. In *Advance Proceedings First International World-Wide Web Conference*, pages 271–277, Geneva, Switzerland, 1994.

[Pit98] James E. Pitkow. Summary of WWW characterizations. *Computer Networks and ISDN Systems*, 30(1-7):551–558, 1998.

[PL98] C. Pilgrim and Y. Leung. Designing WWW Site Map Systems. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, pages 253–258, Florence, Italy, 1998.

[PP97] J. Pitkow and P. Pirolli. Life, Death and Lawfulness on the Electronic Frontier. In *Conference Proceedings on Human Factors in Computing Systems (CHI)*, pages 383–390, Atlanta, GA USA, 1997.

[PP99] P. Pirolli and J. Pitkow. Distribution of Surfers Paths through the Wold Wide Web: Empirical Characterizations. *World Wide Web*, 2(1-2):29–45, 1999.

[PPR96] P. Pirolli, J. Pitkow, and R. Rao. Silk from a Sow's Eear: Extracting Usable Structures from the Web. In *Conference proceedings on Human factors in computing systems*, pages 118–125, Vancouver, Canada, 1996.

[Shn92] B. Shneiderman. Tree visualization with tree-maps: A 2-D space-filling approach. *ACM Transactions on Graphics*, 11(1):92–99, 1992.

[TH98] L. Terveen and W. Hill. Finding and Visualizing Inter-site Clan Graphs. In *Proceedings of the ACM Conference on Human-Computer Interaction*, pages 448–455, 1998.

[TK00] P. Tan and V. Kumar. Modeling of Web Robot Navigational Patterns. In *Proceedings of the WebKDD 2000: Web Mining for E-Commerce*, pages Boston, MA, USA, August 2000.

[Ver00] D. Verton. *Big-Name Web Sites Lack Basic Guidelines*. Computerworld, Octobre 2000. Available at `http://www.computerworld.com` [as of 11-03-2000].

[Wil99] G. Wills. NicheWorks — interactive visualization of very large graphs. *Journal of Computational and Graphical Statistics*, 8(2):190–204, 1999.

[WM97] A. Wexelblatt and P. Maes. Footprints: History-Rich Tools for Information Foraging. In *Proceedings of the Conference for Computer-Assisted Information Retrieval*, pages 75–84, 1997.

[WYB98] K. Wu, P. Yu, and A. Ballmann. Speedtracer: A Web usage mining and analysis tool. *IBM Systems Journal*, 37(1), 1998.

[Zah01] Keyan Zahedi. Analysis and Visualisation of Social Networks. Master's thesis, University Tübingen, WSI Computer Science Dept., April 2001.