User Model User-Adap Inter (2007) 17:183–214
DOI 10.1007/s11257-006-9017-2

ORIGINAL PAPER

Discovering stages in web navigation for problem-oriented navigation support

Vera Hollink · Maarten van Someren · Bob J Wielinga

Received: 1 November 2005/Accepted in revised form: 18 September 2006 /
Published online: 5 December 2006

Abstract Users of web sites often do not know exactly which information they are looking for nor what the site has to offer. The purpose of their interaction is not only to fulfill but also to articulate their information needs. In these cases users need to pass through a series of pages before they can use the information that will eventually answer their questions. Current systems that support navigation predict which pages are interesting for the users on the basis of commonalities in the contents or the usage of the pages. They do not take into account the order in which the pages must be visited. In this paper we propose a method to automatically divide the pages of a web site on the basis of user logs into sets of pages that correspond to navigation stages. The method searches for an optimal number of stages and assigns each page to a stage. The stages can be used in combination with the pages' topics to give better recommendations or to structure or adapt the site. The resulting navigation structures guide the users step by step through the site providing pages that do not only match the topic of the user's search, but also the current stage of the navigation process.

Keywords Navigation support · Information needs · Web usage mining · Navigation stages

V. Hollink(✉) · M. van Someren · B. J. Wielinga
Faculty of Science, University of Amsterdam, Amsterdam, The Netherlands
e-mail: vhollink@science.uva.nl

M. van Someren
e-mail: maarten@science.uva.nl

B. J. Wielinga
e-mail: wielinga@science.uva.nl

1 Introduction

In recent years web sites have evolved from small electronic leaflets to highly complex continually changing information systems. They are used not only to find well-specified information, but also to find answers to less articulate questions and to solve problems. This development poses higher demands on the structure of a web site and tools that support navigation.When solving a problem users often do not know exactly what solutions exist nor what the site has to offer as support in finding the solutions. If a user is not able to express her information needs as keywords, simple search and retrieval are not adequate Alpay et al. (2004). For these users a topic-based navigation structure is also not optimal as some of the pages about a topic will be relevant at an early stage of the search and others only after the user has acquired the knowledge that is needed to select a solution.

For example, consider a web site of an online shop that not only includes detailed information about products but also general information about the product types, their purposes and the conditions for their use. Maybe it even has pages describing possible combinations of products or explaining the products' terminology. For users who are looking up some detail of a specific product this extra information is not interesting. On the other hand, users who are wondering which product is the most suitable for them can benefit enormously from visiting the general information before reading about specific products. The general information does not directly contribute to their buying decisions but rather helps to reformulate and articulate their questions or tell them in which directions to look for a specific product.

Ezendam et al. (2005) and Alpay et al. (2005) showed that users who cannot accurately formulate their questions can be helped greatly by problem-oriented navigation structures that help them to view the information on the site in the right order. Problem-oriented navigation support is especially useful for sites with many incidental or one-time user with questions that need to be solved in a number or steps. Many of these questions first need to be reformulated or abstracted into the terminological and conceptual context of a domain before a solution can be given (see for example the classic work in the context of expert systems by Clancey (1985)). Despite these benefits at present not many sites provide problem-oriented navigation support. One reason is that it is hard to predict in advance with which questions users will come to the site and how this will influence navigation. Moreover, creating advanced navigation structures by hand is an extremely difficult and time consuming task.

Existing methods to automatically support user navigation or structure web sites do not offer problem-oriented navigation support. Recommender systems provide automated support by selecting a limited number of pages which they believe to be interesting for the user. Many systems, including Schwab and Pohl (1999), Zhu et al. (2003) and Mobasher et al. (2002), form clusters of pages with similar topics in such a way that users who are interested in some of the pages from a cluster have a high probability of also being interested in the other pages from that cluster. When a user visits a page, other pages from the cluster of the currently visited page are recommended. The recommendations act as shortcuts, which allow the user to reach his goal without passing through a series of less interesting pages. As we argued above, when navigation involves orientation and reformulation of problems, representing user interests as topic clusters is no longer sufficient. Two pages from the same cluster can be very similar in topic but one may contain introductory information and the

other a detailed solution. In this case the introductory page should be recommended first or appear first in a navigation structure.

In this paper, we propose a method to automatically create navigation components that indicate the preferred reading order for the pages of a web site. The sequential structures underlying these components consist of a number of so called navigation stages. The stages represent groups of pages that fulfill the same role the users' navigation processes. Input to the algorithm is the information stored in the site's log files. From the patterns found in the logs the optimal number of stages is determined and each page is assigned to a single stage. At the same time the algorithm minimizes the number of times the stage order is violated in the user logs.

The stages that are discovered can be combined with an (automatically constructed) content-based structure to construct problem-oriented navigation support. This support can be offered in the form of a menu in which pages are presented in the preferred order or recommendations that do not appear until the user has visited the relevant introductory pages. Other possible applications include filtering or ranking the results of a search engine so that the results match the current stage of the user's navigation process.

The stage discovery algorithm is applicable to sites where the users prefer to read the pages in a specific order, but where the initial navigation structures do not enforce a reading order. If the navigation structure influences the reading order too strongly, the discovered navigation patterns reflect the structure of the site instead of the users' preferences. This happens for instance when the algorithm is applied to sites that rely on in-text links as primary means of navigation. In-text links force users to click through series of pages before other pages can be reached. Consequently, the page order imposed by the link structure will appear as the dominant pattern in the log files.Examples of navigation means to which our method is applicable are topic-oriented menus. The menus show the user where the pages on some topic are located, but do not prescribe in which order the pages should be read.Other suitable structures are site search engines. The order of the pages in result lists indicate the pages' relevance but not their reading order.

The remainder of this paper is organized as follows. Section 2 specifies the task of discovering stages. Section 3 discusses related work. Section 4 describes the stage discovery algorithm. In Sects. 5 and 6, we evaluate the algorithm on log data collected in user experiments. In Sect. 7 artificial data is used to examine the sensitivity of the algorithm to characteristics of the log data. Sect. 8 demonstrates how a stage model can be used for building order sensitive menu structures. The last section contains conclusions and suggestions for further research.

2 The SeniorGezond site

In this section, we describe the navigation structure of the SeniorGezond site[1] which motivated us to create the stage model. The SeniorGezond site is a Dutch healthcare site developed by the Netherlands Organization for Applied Scientific Research (TNO) in cooperation with domain specialists from the Geriatric Network and the Leiden University Medical Center. It contains information for elderly people about the prevention of falling accidents (Alpay et al. 2005).

[1] http://www.SeniorGezond.nl/

Before the current navigation menu of the SeniorGezond site was developed, other menu structures were designed and tried out in various prototypes. In the first prototype a purely topic-oriented structure was used. Evaluation of this prototype showed that people experienced great difficulties in expressing their problems in terms of the site's topics (Alpay et al. 2004). This motivated the developers to build a navigation structure that is directed more at the viewpoint of the visitors.

The current navigation menu of the SeniorGezond site reflects the Precaution Adoption Process (PAP) model, a psychological model that describes how people become aware of their problems and translate their problems into actions (Ezendam et al. 2005; Alpay et al. 2005). The stages of the PAP model are translated into a menu structure with three layers. The first layer consists of problem descriptions, the second layer consists of descriptions of general solutions and the third layer consists of practical information about products and services that implement the general solutions. The *product* pages contain the information that in the end solves the users' problems. The other two layers help the users to articulate their problems and provide information about available solutions. A screenshot of the SeniorGezond site and the layered menu can be found in Fig. 1.

The problem-oriented menu of the SeniorGezond site has two dimensions. The horizontal axis or the layers of the menu represent navigation stages. On the vertical axis we see a number of topics such as dizziness and joint wear. Automatically finding clusters of pages with similar topics, as on the vertical axis, is a well known task that is worked on by many researchers, including Mobasher et al. (2002), Perkowitz and Etzioni (2000) and Pierrakos and Paliouras (2005). The emphasis of our work is on the other dimension: finding the stage structure.

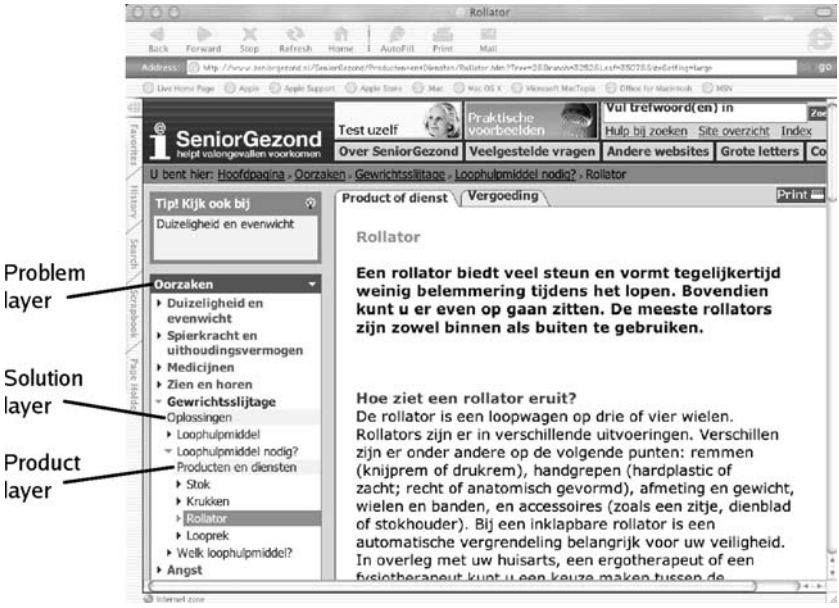

Fig. 1 A screenshot of the SeniourGezond site and its problem-oriented menu (in Dutch). The screenshot shows a *product* page about rollators. In the menu this product is connected to the solution 'Loophulpmiddel nodig?' (Need a walking aid?) and the problem 'Gewrichtsslijtage' (joint wear)

We define a navigation stage as a group of pages that play similar roles in the users' navigation processes. The structure that our method searches for consists of a set of stages and a relative ordering of the stages. The stages are ordered in such a way that users generally prefer to visit pages from the first stage at the beginning of their sessions, then proceed to pages from the second stage, etc. There is no preferred visiting order for two pages within the same stage.

Ezendam et al. (2005) evaluated the layered structure of the SeniorGezond site by analysis of the log files and a usability study. She found that all three layers of the menu were visited frequently and that most transitions between pages occurred within layers or from problems to solutions or from solutions to products. Moreover, many users visited all three layers. The usability study showed that people recognized the layered structure and found it easy to use. In conclusion, Ezendam's results provide strong evidence that the problem-oriented structure is used as intended and that it provides better guidance than a topic-oriented menu.

There are many other domains in which a problem-oriented structure might be able to provide guidance. The study of Choo et al. (2000) shows that web users vary substantially in the extent to which they know what information they are searching for. It frequently happens that people want to solve a problem or answer a question and do not know beforehand what solutions exist and what the site has to offer. They can formulate their information needs in terms of the questions, but not in terms of solutions or answers. In these cases, it is important that the site is structured around the viewpoints of the users rather that the viewpoints of the providers of the content.

Another example is someone who wants to ask the local government permission to build a shed. In the end her question will be answered by a web page that contains the address of an organization she needs to write to or an application form for a building permit. However, when she visits a governmental site about building legislation she cannot search for these organizations and permits, because she may not know that they exist or whether they apply. Here, a problem-oriented menu could be of great help. Instead of referring her to the application form directly, it would first provide general information about building legislation, then refer her to some application procedures and finally to a downloadable application form. A similar situation occurs when someone wants to buy some product he is not familiar with. A topic-oriented menu orders the available products according to some product features. If the visitor does not know exactly what he needs he might not be able to select the features that are most appropriate for his situation. A problem-oriented structure starts with offering more general information about product types and product features. In the next step this information can be used to select a product.

In spite of the potential benefit of problem-oriented menus, not many sites offer this service. No doubt one of the reasons for this is the considerable effort needed to created these structures. In this work we present a method to automate this process. It allows site owners to learn the order in which users want to read the pages of a site and it creates a problem-oriented navigation structure. The method saves site owners the effort of restructuring the contents of the site and saves the users the effort of tracking the relevant information through the site's structure.

3 Related work

In recent years much research has been devoted to the automatic construction and adaptation of navigation structures. Probably the most notable in this respect is the

work of Perkowitz and Etzioni (2000). They developed PageGather, an algorithm to automatically create index pages for web sites. PageGather creates a graph representing pages and their co-occurrences in the users' sessions. The connected components in the graph form the basis of the index pages. More recently Pierrakos and Paliouras (2005) invented an algorithm to select parts of a web directory that are interesting for a group of users. Their *Community Directory Miner* employs probabilistic latent semantic analysis to extract clusters of users with common interests and to select page categories that correspond to these interests. The MONTAGE system (Anderson and Horvitz 2002) automatically assembles personalized start pages (montages) for web users. The montages contain links to pages that the user has visited in contexts similar to his current situation. By providing shortcuts to frequently accessed web content the MONTAGE system facilitates routine web browsing. In Hollink et al. (2005b) we presented an algorithm that uses the information gain criterion to optimize a navigation menu in terms of the number of steps that users need to reach their target information. Web personalization refers to a large family of methods to adapt web navigation structures or web content to individual users. For a survey of usage-based personalization methods we refer to Pierrakos et al. (2003). A more broad view on adaptive navigation support in the context of adaptive hypermedia is given in Brusilovsky (2001).

A large majority of the research on navigation adaptation for both groups of users and individuals, including the ones mentioned above, focuses on the selection of interesting content. Much less attention has been paid to the order in which the pages should be presented to the users. An exception is the work on educational hypermedia, e.g. De Bra and Calvi (1998) and Brusilovsky et al. (1998). However, here the preferred order of the pages, or more general the content chunks, is specified by hand. Although it is generally agreed that the need to specify these relations is one of the main drawbacks of these systems, to our knowledge no attempts have been made to automatically learn the prerequisites from user behavior.

A variety of machine learning techniques have been applied to the task of learning a model of web usage. Again, most models only include page relevance and not page order. A type of models that do include page order are Markov models e.g. Pitkow and Pirolli (1999), Sarukkai (2000) and Deshpande and Karypis (2004). Markov models make predictions about the next step of a user using the observed frequencies of sequences of pages in the log files. They contain information about sequences of individual pages, but do not specify relations between larger units such as page clusters. This lack of a large scale structure means that they do not provide insights in the behavior of the users and cannot serve as a basis for automatically created navigation structures.

In Anderson et al. (2001) and Cadez et al. (2003) mixtures of Markov models are used to find clusters of users with similar browsing patterns. A limitation of this work is that the patterns that characterize the clusters are restricted to sequences of manually assigned page categories. These sequences can be viewed as navigation stages, but the possible stages are limited to the predefined page categories.

Ypma and Heskes (2002) overcome this problem by representing web user behavior as a hidden Markov model (HMM). In HMMs the hard-coded page categories are replaced by a number of unobservable states. The current state of a user determines the probability of visiting pages and moving to other states. In theory, the states of a HMM can contain pages with similar topics as well as similar stages. However, inspection of the states produced in Ypma and Heskes (2002) reveals that in practice the pages are primarily grouped by topic. Moreover, the simultaneous optimization

of stages and topics makes learning HMMs computationally expensive. Furthermore, for the creation of navigation structures HMMs yield the same problem as Markov models, albeit to a lesser extent. The states of a HMM provide some structure, but it is not completely clear how they can be translated into a navigation structure.

A more clear interpretation can be given to the model presented in Jin et al. (2005). Just like Pierrakos and Paliouras (2005) they use probabilistic latent semantic analysis to detect clusters of users who have visited similar sets of pages (in the paper called *tasks*). Once the clusters have been fixed, the authors find task-level usage patterns by computing the most likely tasks in each step of the users' navigation. These patterns provide information about tasks that are frequently performed subsequently in a session. A drawback of this approach is that tasks can only be distinguished if they are frequently performed in isolation. Series of subtasks that are almost always performed together are viewed as one task. This makes the method appropriate for finding top-level tasks, but not for dividing the navigation within tasks into stages.

Another interesting model of web usage is the information foraging theory first introduced in Pirolli and Fu (2003). The theory describes how people decide when to keep browsing in the current information source (web site) and when to go searching for a better source. According to Pirolli and Fu, people estimate the amount of relevant information that can be found on a site based on the site's *information scent*.When the information scent becomes too low they switch to another site with a higher scent. Pirolli and Fu model the sequential behavior of users who are trying to fulfill an information need. The model explains how an information need gets satisfied when information is found, but not how an information need is changed by the information found so far. As a result, navigation assistance systems based on the information foraging theory, such as the one in Herder (2004), can help users to find trails along the most informative pages, but the theory does not prescribe in which order the pages should be on the trail.

In sum, an extensive amount of work has been devoted to the automatic creation of link structures, but in this area presenting the links in the right order has received little or no attention. Possibly, this is due to the lack of order sensitive models that are comprehensible enough to be used in navigation structures. In Hollink et al. (2005a) we presented a method for finding a simple and understandable model of web usage in which the main order characteristics are preserved. In the current work this algorithm is refined and evaluated in more depth. Moreover, here we go beyond building the model and show how it can be used to automatically create sequential navigation structures.

At the algorithmic level the method presented in this work bears some resemblance to scaling methods such as uni-dimensional preference scaling (Carroll 1972) and ordinal utility revelation Domshlak and Joachims (this issue). Like our method, these methods seek to convey the underlying structure in a set of items by scaling them onto one dimension. Scaling methods make use of observed relations between items. If we apply scaling to web usage data, the items are pages and the relation between them is 'is visited (directly) before.' A problem occurs when two sets of pages occur almost never in the same sessions. In this case the scaling algorithms have no accurate information about the relative positions of the two sets of pages. Our algorithm overcomes this problem by using the positions of the pages in the sessions which makes all pages comparable.

Algorithm 4.1: DISCOVER_STAGES(log_files)

$sessions \leftarrow$ PREPROCESS(log_files)
$f_{ARP} \leftarrow$ INITIALIZE($sessions$) (i)
$f_{stage} \leftarrow$ CONSTRUCT_STAGES($f_{ARP}, sessions$) (ii)
$f_{stage} \leftarrow$ OPTIMIZE_STAGES($f_{stage}, sessions$) (iii)
return (f_{stage})

Fig. 2 The top level of the stage discovery algorithm

4 The stage discovery algorithm

In this section, we present an algorithm that automatically divides the pages of a web site into navigation stages. Each stage represents a group of pages for which the order of their requests cannot be accurately predicted, but that as a group can be ordered relative to other groups. The pages in a stage may not have similar topics, but play similar roles in the users' navigation processes.

The stage discovery algorithm needs as input a set of log files of the site for which a stage structure is created. To collect these logs the site must have been online for some time. Moreover, while the server logs are collected, the site's navigation structure must not force the users to visit the pages in a specific order. As we discussed in the introduction, many commonly used navigation structures fulfill this requirement, including topic-based menus and site search engines.

The stage discovery algorithm does not make use of the pages' contents. Content-based methods use word similarity to cluster pages with related topics. These clusters may not correspond to stage structures because pages with similar roles do not necessarily contain similar words. As a result one topic cluster can contain both generic introductory pages and pages with highly specific information.

Figure 2 shows the top level of the algorithm in pseudocode. The algorithm is composed of three main steps that are discussed in detail below:

1. *Initialization* The pages are scaled along one dimension (i).
2. *Stage construction* By clustering the scale is divided into an optimal number of stages and the pages are assigned to the stages (ii).
3. *Stage optimization* The page assignments are optimized through bootstrapping (iii).

4.1 Initialization

Before the actual initialization starts, the sessions of individual users are extracted from the server logs. Here a session is defined as the sequence of pages that a user has viewed during her visit to the site. When users are required to login to the site, the requests of individual users can be uniquely identified. Otherwise, the sessions need to be restored from the IP addresses and browser information that are available in standard log files. Log files of sites with dynamically created pages sometimes contain large numbers of URLs pointing to pages with almost the same contents. In this case pages with very similar contents need to mapped onto one URL. A wealth of techniques has been developed to improve the quality of restored sessions when proxies and caching are used, e.g., Cooley et al. (1999), but a discussion of these techniques is

Algorithm 4.2: INITIALIZE($sessions$)

```
for each s ∈ sessions
  do { for i ← 1 to |s|
       do { p ← the page on position i in s
            Add RELATIVE_POSITION(i, |s|) to RP_list_p
for each p ∈ pages
  do f_ARP(p) = AVERAGE(RP_list_p)
return (f_ARP)

procedure RELATIVE_POSITION(position, session_length)
 if session_length = 1
   then return (0.5)
   else return (position − 1)/(session_length − 1)
```

Fig. 3 The first step of the stage discovery algorithm: initialization

beyond the scope of this article. From now on we assume that the sessions are restored and represented as lists of consecutively visited pages.

The second preprocessing step is the removal of all revisits from the restored sessions. Users who are visiting pages from one stage might sometimes go back to a page from the previous stage that they have already visited to look up details they do not remember accurately. To prevent the algorithm from incorrectly inferring that these pages belong to the later stage, we remove all revisits from the sessions. Another advantage of removing the revisits is that it removes the difference between sessions of users who use browser caching and sessions of users who do not use browser caching.

After preprocessing the actual initialization step starts. In this step the pages are laid out on a one-dimensional scale that reflects the parts of the sessions in which they are visited most often. The initialization process is summarized in Fig. 3.

The algorithm starts with collecting the positions of the pages in each session and normalizing the positions by dividing them by the length of the session. We define the relative position (RP) of a page p at the$k$th place in a session consisting of m page visits as:

$$RP(p) = (k-1)/(m-1)$$

The position of a page in a session with only one page is defined as 0.5. The *average relative position* (ARP) of a page is the average over its relative positions in all sessions in which it appears. In the pseudocode in Fig. 3 the ARPs of the pages are represented as a function f_{ARP} that maps pages onto ARP values.

The reason for introducing the concept of average relative position is that it allows us to lay down all pages onto a one-dimensional scale.The position on this scale reflects the part of the sessions in which the page is visited most often. Pages with low ARP values are visited mainly in the beginning of sessions, while pages with high ARPs belong to the end of sessions. This insight is formalized in the second step of the stage discovery algorithm where the stages are constructed.

4.2 Stage construction

In the stage construction step the ARP values of the pages are clustered. The resulting page clusters form the initial stages. In addition, in this step we determine in how many

Algorithm 4.3: CONSTRUCT_STAGES($f_{ARP}, sessions$)

```
best_fitness ← 0
for no_stages ← 1 to max_no_stages
do { likelihood, gaussians ← EM(no_stages, RANGE(f_ARP))                    (i)
     ARP_regions ← INTERSECT_POINTS(gaussians)                              (ii)
     ARP_regions ← TIGHTEN_REGIONS(0.7, ARP_regions)                        (iii)
     for each p ∈ pages
     do { f_stage(p) ←?
          for c ← 1 to no_stages
          do { if f_ARP(p) within ARP_regions[c]
               then f_stage(p) = c
     fitness ← (1 − α) ∗ likelihood +                                       (iv)
               α ∗ PROPORTION_REGULAR(sessions, f_stage)
     if fitness >= best_fitness
     then { best_f_stage ← f_stage
            best_fitness ← fitness
return (best_f_stage)
```

Fig. 4 The second step of the stage discovery algorithm: stage construction

stages the navigation can be decomposed. Figure 4 shows the construction process in pseudocode. In the coming paragraphs we first explain how we transform the ARP values into stages when the number of stages is known (in Fig. 4 starting at (i)) and then we explain how the optimal number of stages can be estimated (iv).

4.2.1 Constructing a fixed number of stages

To divide the ARP scale into *n* clusters we apply the Expectation Maximization (EM) algorithm (Dempster et al. 1977). The EM algorithm fits a mixture of *n* one-dimensional Gaussians to the ARP values (i). In the resulting mixture each Gaussian corresponds to a cluster of ARP values. To transform the Gaussians into stages we compute for each Gaussian in which interval of the ARP scale the Gaussian is the most likely component. In other words, we compute the intersection points of the Gaussians in the mixture (ii). The intersection points divide the ARP scale into a number of regions that correspond to the stages.

Now each page can be assigned to the stage in which ARP region the page's ARP value falls. However, the assignment of pages with ARPs close to the region boundaries is very insecure. Therefore, for each stage we increase the lower boundary of its region and decrease the upper boundary of its region until only 70% of the stage's original ARP region remains (iii). Pages with ARPs within these intervals are assigned to the corresponding stages. The assignment of pages with ARP values outside the stage boundaries is postponed to the last step of the stage discovery algorithm. In Fig. 4 the stage assignments are represented by the function f_{stage} that maps pages onto stages.

4.2.2 Determining the number of stages

As can be seen in Fig. 4 the optimal number of stages is determined by generating and evaluating models with increasing numbers of stages and selecting the best performing

model. Intuitively, the best performing model is the one that most accurately describes the user behavior that is found in the log data. The quality of a model's fit to the ARP data is expressed by the average log-likelihood of the ARP values given the Gaussian mixture. However, this measure alone does not suffice to compare the performance of mixture models as models with more components always have the potential to fit the data at least as good as models with fewer components. Consequently, using only the log-likelihood could lead to severe overfitting.

To prevent the selection of overly complex models the likelihood needs to be combined with a measure that favors models with smaller numbers of stages. One possibility is to penalize models relative to the number of model components. However, for this problem a more meaningful solution is at hand. The *proportion regular transitions* reflects the extent to which the individual user sessions follow the stage pattern prescribed by the model. The more sessions follow the pattern, the better the model. The proportion regular transitions is defined as the proportion of the page transitions made in the user sessions in the log file that are regular according to a model. A transition between two consecutively visited pages is regular if the pages are from the same stage or if the stage of the first page directly precedes the stage of the second page.

The proportion regular transitions is generally smaller when a model with fewer components is used, because smaller models place fewer restrictions on the page order. According to a model with only one stage all transitions occur within one stage and thus all transitions are regular. In the other extreme, a model that assigns each page to a separate stage prescribes a complete page ordering. With this model all deviations from the prescribed path are marked as irregular.

As a final measure of model performance, we define the *fitness* of a model as a linear combination of its average log-likelihood and its proportion regular transitions:

$$Fitness(n|S) = (1 - \alpha) * Likelihood(S|n) + \alpha * Proportion\_regular(S|n)$$

Here $Likelihood(S|n)$ is the average log-likelihood of the sessions S given the model with n stages. $Proportion\_regular(S|n)$ is the proportion of the page transitions in S that are regular according to the model with n stages. α is a weighting parameter.

In the current version of the algorithm we test all models with a number ofstages smaller than some user specified value. The model in this set with the highest fitness is used to create the initial stage assignment that is passed on to the next step of the stage discovery algorithm.Another possibility is to start with a model with one stage and test continually larger models until the fitness no longer increases. In this paper we chose to implement the former method. It requires a little more computation, but is less sensitive to variations in the fitness.

The EM algorithm does not always result in a fit that can be interpreted as a valid stage model. Sometimes one of the model components 'dies', its prior probability becomes zero. In this case the model effectively has become a model with a smaller number of stages. Another possibility is that one Gaussian is superimposed on another Gaussian, so that the regions between the Gaussians' intersection points do not include the means of the distributions. In both situation we consider the fitted model to be an invalid stage model and assign it a fitness of zero. Note that both problems to not occur with one stage models. This means that a one stage model (a model without a division in stages) is correctly marked as the preferred choice when no other valid stage model can be found for a data set.

4.3 Stage optimization

In the previous section the pages were assigned to stages on the basis of the parts of the sessions in which they occurred most. Here we improve the classification by looking at the context in which the pages occur in the individual sessions (see Fig. 5 for the pseudocode).

In our model stages are strictly ordered, so that most navigation steps occur within one stage or from a page from one stage to a page from the next stage. As a consequence, a page which occurs in the sessions mostly between two pages from stage *s* has a high probability of belonging to stage *s*. We use this idea to correct the classification of pages that are initially assigned to an incorrect stage. For each page *p* and each stage *s* we count the number of times *p* occurs between two pages of stage *s*. We define the *evidence of misclassification* of *p* as the difference between the number of times *p* occurs in its current stage and the maximum number of times *p* occurs in some other stage (ii). The pages with the highest evidence of misclassification are reassigned to the stage in which they occur most (iii). With the new classifications for each page the evidence of misclassification is recomputed and again the stages of the pages with the highest evidence are changed. This bootstrapping process is continued until no more stage changes are made or until a maximum number of cycles is reached.

Because the bootstrapping process can be sensitive to sessions of users who did not follow the stage structure very accurately, it is embedded in a larger cycle. The

Algorithm 4.4: OPTIMIZE_STAGES($f_{stage}, sessions$)

```
min_regular ← 0.9
while min_regular >= 0.5
do { reg_sessions ← ∅
     for each s ∈ sessions
       do { if PROPORTION_REGULAR(s, f_stage) >= min_regular   (i)
            then Add s to set reg_sessions
     for cycle ← 1 to max_no_cycles
       do { miss_evidence ← Max_{p∈pages, c∈RANGE(f_stage)}   (ii)
              (EVIDENCE(p, c, f_stage, reg_sessions)
              −EVIDENCE(p, f_stage(p), f_stage, reg_sessions))
            if miss_evidence = 0
              then break
            f_stage ← REPAIR_STAGES(miss_evidence, f_stage)   (iii)
     min_regular ← min_regular − 0.1                          (iv)
return (f_stage)

procedure EVIDENCE(page, stage, f_stage, sessions)
 evidence ← 0
 for each s ∈ sessions
   do { for i ← 2 to |s| − 1
        do { p_i ← the page on position i in s
             if p_i = page and f_stage(p_{i−1}) = f_stage(p_{i+1}) =
               stage
               then evidence ← evidence + 1
 return (evidence)
```

Fig. 5 The third step of the stage discovery algorithm: stage optimization

first time the bootstrapping process is called we use only sessions that have a least 90% regular page transitions (i). In later cycles this restriction is gradually relaxed until all sessions are used with at most 50% regular page transitions (iv). In this way we improve the quality of the data that is used during bootstrapping and reduce the chance that the process drifts towards a suboptimal classification.

4.4 Complexity

The stage discovery algorithm is designed to run offline. There is no need to rerun the algorithm each time a user requests a page, as the discovered stage structures are typically stable behavior patterns that do not change on a daily basis. As a result, the running time of the algorithm is not a major issue. Nevertheless, in this section we briefly discuss the algorithm's space and time requirements as scalability is essential when using web log data.

The initialization phase involves one pass through the log file. The time and memory complexity of this phase are linear in the length of the log file. For stage construction more resources are needed. The time complexity of one iteration of the EM algorithm is linear in the size of the data set and the number of model components. The data set consists of one data point per page so that the complexity becomes $O(p.n)$, where p is the number of pages on the site and n is the number of stages. The number of cycles that the algorithm needs to converge depends on the distribution of the data and is hard to predict in advance. However, in practice for many data sets the number of cycles appears to be approximately constant under varying amounts of data and model components, e.g., Cadez et al. (2003). In these cases, the total time complexity is linear in the number of data points and the number of components. Our experiments suggest that also for the stage discovery algorithm this relation is roughly linear. To determine the optimal number of stages the construction process is run with various numbers of stages. For each number of stages EM is called and the log file is traversed to determine the proportion regular transitions. As a result, the time complexity of the stage construction phase is $O(N.(p.\frac{1}{2}.(N+1)+s))$, where s is the size of the log file and N is the maximum number of stages. The memory requirements of the EM algorithm are modest as only the values of the current cycle need to be stored. The space complexity is $O(p.n)$.

In the stage optimization phase for each regularity level the algorithm makes one pass through the log file to select the regular sessions. With these sessions a number of bootstrapping cycles are performed. The time needed for one bootstrapping cycle is $O(r)$, where r is the number of regular sessions. Our experiments indicate that the number of bootstrapping cycles does not increase with increasing numbers of sessions. Consequently, the time needed to perform the bootstrapping process is linear in the number of regular sessions. In total, the time complexity of the optimization phase is $O(s+b.r)$ per regularity level, where b is the number of bootstrapping cycles. The space complexity is $O(r+p)$.

The time requirements of the stage optimization phase can be problematic when s and r are very large. Fortunately, the time can easily be reduced by increasing the required proportion of regular transitions. The minimum proportion of regular transitions can thus be used as a parameter to control the computational costs. Increasing this parameter reduces both the number ofbootstrapping cycles and the number of sessions included in bootstrapping.

Because the total time and space requirements of the stage discovery algorithm are linear in the length of the log file and the number of pages of the site, the algorithm can be used on large data sets. To give an indication of the practical time and memory costs: running the algorithm on the log data from the SeniorGezond experiment (244 sessions with in total 5057 server requests, see Sect. 5) takes 7 seconds on a normal desktop computer using an implementation that was not extensively optimized.

5 Discovering stages for the SeniorGezond site

To evaluate whether the stage discovery algorithm is able to produce useful stage structures, we test it on two different domains. The first domain is the SeniorGezond site that was discussed in Sect. 2. The second domain is described in Sect. 6. In both cases the structure of the web sites is used as a gold standard to which the classification made by the algorithm is compared.

In this section, we apply the stage discovery algorithm to log data from the SeniorGezond site to see whether the algorithm is able to reconstruct the site's stage structure from the users' navigation patterns. In this experiment we used a simplified version of the site. Instead of the stage-oriented menu this version contained only a single large menu with a long list of links to all pages of the site. Furthermore, all external links, in-text links and other means of navigation were removed.

Thirty participants performed each ten search tasks on the modified SeniorGezond site. In each task the participants were asked to play the role of an elderly person in a problematic situation who searched the SeniorGezond site for a solution. The formulation of the problem descriptions was on purpose a little vague. We wanted to simulate users who felt they had a problem, but were not able to clearly articulate their problem. An example of a problem description can be found in Appendix A. The participants were mainly computer science students. None of them knew the purpose of the experiment.

The participants accessed the modified site through a login page. During the search assignments all clicks were recorded. For each assignment of each participant we listed the pages that were viewed consecutively during the performance of the assignment. This resulted in 244 lists of pages (sessions) with an average length of 7.3 page views (after revisit removal). As shown in Table 1, 90 of the 120 pages were visited at least once. In the following discussion we will only consider the 90 visited pages, since the algorithm has no information about the remaining 30 pages. This does not affect the scope of our conclusions, because in a real application we can safely assume that all web pages are visited.

Table 1 The number of pages and page visits per page type in the SeniorGezond experiment after removal of revisits

Page type	No. pages	No. visited pages	No. visits	Avg. no. visits per visited page
Problems	10	10	355	35.5
Solutions	27	20	732	36.6
Products	83	60	685	11.4
Total	120	90	1772	19.7

First, we analyzed the behavior of the subjects by hand to see whether they followed the expected pattern *problems* → *solutions* → *products*. In this analysis we made use of the types of the pages. Of course, this information was not available to the discovery algorithm. The transition matrix in Table 2 shows for each page type how many times someone went from a page of this type to a page of each other type. From the matrix it is clear that by far most transitions occur within stages or go from one stage to the next stage. This confirms that the different page types are used during different navigation stages. Furthermore, the transition frequencies in Table 2 are very similar to the ones found for the online version of the SeniorGezond site (Ezendam et al. 2005). Apparently, the tasks used in the experiment elicit behavior that closely resembles that of the real users.

The navigation stages can be seen even more clearly from the ARP distributions of the three page types shown in Fig. 6. The figure clearly shows that the *problem* pages are visited mostly in the beginning of the sessions, the *solution* pages in the middle and the *product* pages in the end. From these results we conclude that for our users the three page types of the SeniorGezond site indeed form navigation stages: the *problem* pages form the first stage, the solutions the second stage and the products the third stage.

Above we showed that the three navigation stages of the SeniorGezond site can be seen clearly when the types of the pages are known. We will now demonstrate that the stage discovery algorithm can find the stages without requiring knowledge about the types of the pages.

Table 2 Relative frequency of transitions between the page types of the SeniorGezond site in percentages

		To type			
		Problems	Solutions	Products	Stop
From type	Start	**75.4**	20.1	4.5	–
	Problems	**38.9**	**52.1**	4.8	4.2
	Solutions	3.0	**63.5**	**25.0**	8.5
	Products	1.6	4.8	**69.2**	**24.4**

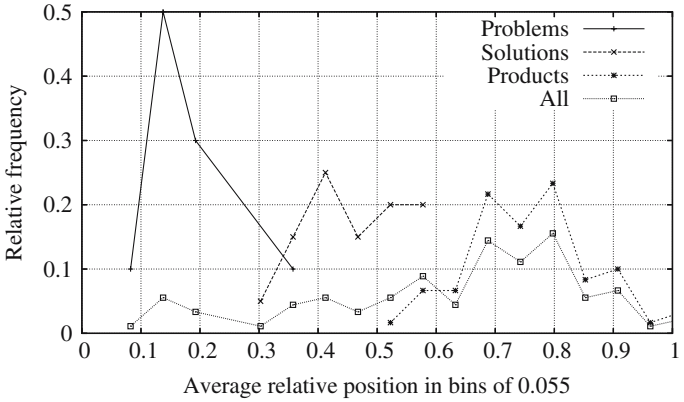

Fig. 6 The distribution of the ARPs of the pages in the log data of the SeniorGezond experiment

To determine the number of stages we fitted models with one up to eight stages and determined the fitness of these models as described in section 4.2. We repeated the experiments with various values for the parameter α to determine the optimal value for α. Figure 7 shows the fitness of the models when various values of α are used. The correct number of stages, three, is found when α lies between 0 and 0.625, with the most clear optimum around 0.25.

When the stage discovery algorithm is applied to a new site, the optimal value for α cannot be determined is this way, as for a new site the correct number of stages is unknown. However, in the following sections we will see that an α of 0.25 also works well in other domains, so that in a new domain this value can be used directly.

Next, the model with three stages was used to assign each page to a stage. Table 3 shows the proportion of the SeniorGezond pages that was classified correctly, the accuracy. Note that a stage assignment is called correct if a *problem* page is assigned to the first stage, a *solution* to the second stage or a *product* page to the third stage. After the stage construction step 86% of the pages were assigned to the correct stage. As visible in the table most *problem* and *product* pages were classified correctly, but the assignment of the *solution* pages was not very accurate. Inspection of the process showed that most of these pages were not really misclassified, but not yet assigned to a stage. In the stage optimization step these unclassified pages were assigned to a stage and all misclassifications were repaired. In the end all pages were assigned to the correct stage. These results lead to the conclusion that the stage discovery algorithm can accurately discover the navigation stages of the SeniorGezond site from log files.

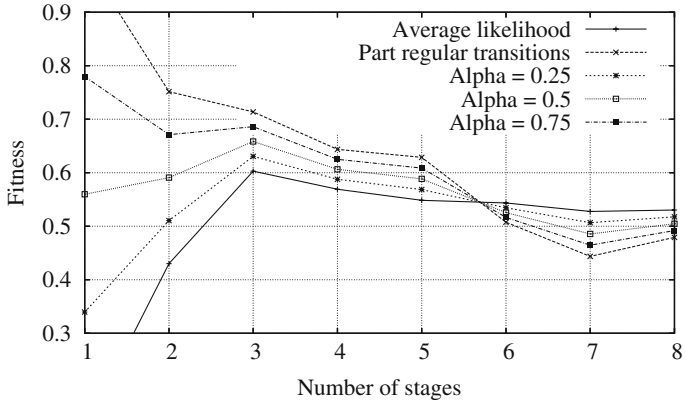

Fig. 7 The fitness of models with various numbers of stages and various values of α for the SeniorGezond data

Table 3 The accuracy of the stage discovery algorithm on the pages of the SeniorGezond site

Step	Accuracy			
	Problems	Solutions	Products	Total
Stage construction	0.90	0.45	0.98	0.86
Stage optimization	1.00	1.00	1.00	1.00

6 Discovering stages for a hardware comparison site

We replicated the Senior Gezond experiment with pages and tasks in a second domain. The site we used in this experiment contains information about computers and products related to computers such as printers and digital cameras. The site not only provides information about specific products and terminology, but also about the importance of the various features of the products.

The site consists mainly of four page types. The so called *howto* pages tell the users how to buy a product from some category. They discuss the different types of products, the importance of the features for various purposes and explain the terminology used to describe the features. For instance, the 'How to buy a printer' pages explain the difference between laser printers and inkjet printers and advice the users on which type of printer to buy in which situation. In addition, they explain the importance of features such as resolution and cartridge capacity. The *overview* pages provide a site-by-site comparison between a number of top-rated products. For example, the laser printer *overview* pages show small photos of ten laser printers and list briefly the most important features of each printer. The most specific pages are the *product* pages. These pages contain detailed information about single products. The full specifications of the products are given and for some products a series of photos is provided. Besides these three types of structured pages, the site also contains a number of news articles. These address a wide variety of topics, including new developments, trends and opinions.

Users who want to buy a product without being an expert in the area of the product can first explore the domain by reading the *howto* pages. Once they have an idea of their needs in terms of product features, they can use the *overview* pages to select some promising products. Finally, they can make a decision based on the specific product features. This scenario suggests that the *howto* pages form the first navigation stage, the *overview* pages the second and the *product* pages the third stage. The role of the *news* pages is less clear.

Despite the natural order of the page types, the hardware site does not provide a stage-oriented menu. All four page types are represented as top level items in the site's menu. The links to the *howto* pages are not emphasized, so that the user is given no clue about what pages are good starting points. In contrast, a stage-oriented menu would guide the users from the *howto* pages via the *overview* pages to the *product* pages. Such a menu could potentially reduce the users' efforts needed to find the pages that are relevant in each stage of the search process.

To see whether users are indeed inclined to visit the hardware pages in some order we conducted an experiment parallel to SeniorGezond experiment. For this experiment we selected *howto*, *overview*, *product* and *news* pages from eleven product categories. Again we removed the menu and link structure from the pages and replaced it by a flat menu that did not impose or suggest any visiting order. The number of pages in the hardware comparison experiment was much larger than in the SeniorGezond experiment (303 vs. 120), which made it much harder for the participants to locate the useful pages. With this number of pages selecting a page from an alphabetic list of links to all pages of the site would take to long. Therefore, we added a selection facility, which allowed the participants to enter keywords and only view links to pages that contained the keywords. The selected links were still ordered alphabetically and not by relevance, so that the link order did not bias the participants' choices.

Thirty-one participants performed ten search tasks. In each task the participants played the role of a person who wanted to buy a product for some purpose, but who was not knowledgeable in the domain. An example of a task description can be found in Appendix B. The experiment resulted in 288 sessions with an average of 6.4 page views per session (after revisit removal). These figures are comparable to the figures of the Senior Gezond experiment. However, the hardware comparison site contained more pages than the SeniorGezond site, so that the individual pages were visited less frequently as shown in Table 4.

Figure 8 shows the distribution of the ARPs of the four page types of the hardware site. The figure confirms our hypothesis that the *howto* pages are visited mostly in the beginning of the sessions, the *overview* pages in the middle and the *product* pages in the end. Another interesting finding is that the *news* pages did not seem to belong to a navigation stage, but were visited throughout the sessions. We did not include the transition matrix here, but it shows the same patterns.

The stage discovery algorithm was applied to the log data. Figure 9 shows the fitness of models with various numbers of stages. The models with 6 and 8 stages have a fitness of 0. In these cases the solutions of the EM algorithm were not valid stage models, because one of the model components had zero probability (see Sect. 4.2.2). Unfortunately, the model with three stages did not have the highest fitness with any value of α. The algorithm comes closest to the correct solution when an α of 0.25 is used, which coincides with the optimal value found in the SeniorGezond experiment.

Table 4 The number of pages and page visits per page type in the hardware comparison experiment after removal of revisits

Page type	No. pages	No. visited pages	No. visits	Avg. no. visits per visited page
Howtos	44	40	543	13.6
Overviews	27	26	359	13.8
Products	136	122	804	6.6
News	96	59	136	2.3
Total	303	247	1842	7.5

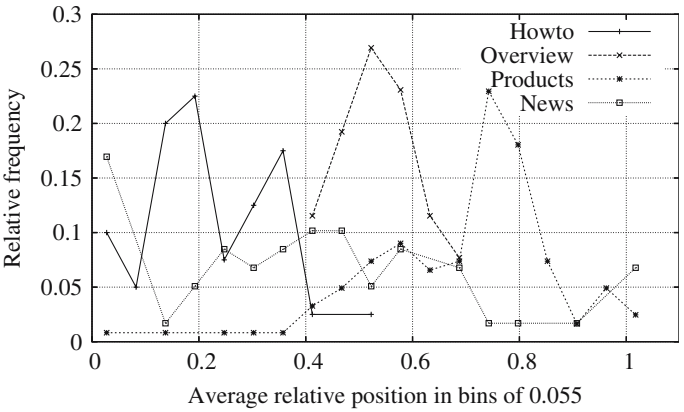

Fig. 8 The distribution of the ARPs of the pages in the log data of the hardware comparison experiment

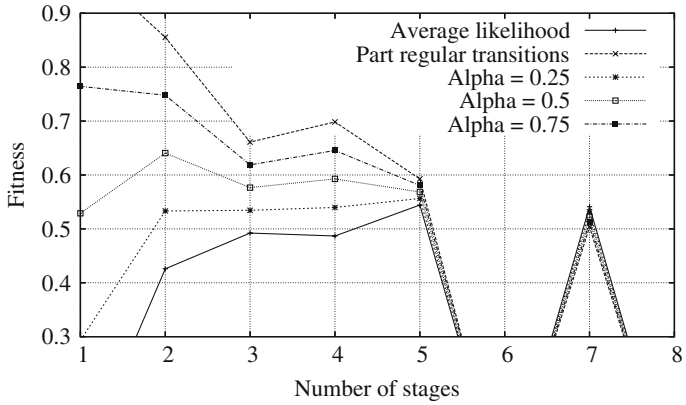

Fig. 9 The fitness of models with various numbers of stages and various values of α for the hardware comparison data

Table 5 The accuracy of the stage discovery algorithm on the pages of the hardware comparison site

Step	Accuracy			
	Howto	Overview	Products	Total
Stage construction	0.78	0.69	0.61	0.66
Stage optimization	0.90	0.88	0.69	0.76

The average likelihood and proportion regular transitions in Fig. 9 follow less smooth courses than the ones of the SeniorGezond data. This is most likely due to the smaller numbers of visits per page, which makes the ARP values less accurate and the boundaries of the stages less sharp. In the next section, the negative effect of small amounts of data is shown in simulation experiments.

Subsequently, we evaluated how accurate the stage discovery algorithm could find the stages, when the optimal number of stages was known. The algorithm was applied to the experimental data and assigned all visited pages to a stage. We limit the computation of the classification accuracy to the *howto*, *overview* and *product* pages, because the *news* pages do not have a correct stage. Ideally the algorithm would recognize automatically which pages belong to a particular part of the sessions and which pages are visited throughout the sessions, but the current version does not yet include this feature.

The accuracy of the classification after stage construction and stage optimization is shown in Table 5. The algorithm assigned 76% of the pages to the correct stage. The classification of the *howto* and *overview* pages was very accurate, but the classification of the *product* pages proved more difficult. This difference can be explained by the fact that the *howto* and *overview* pages are visited twice as much as the *product* pages (see Table 4). More visits per page make the ARP values more accurate and provide more evidence during the optimization phase. This fact also explains the difference with the accuracies found in the SeniorGezond experiment. In the next section, we evaluate the effect of the number of pages on the accuracy in detail.

The effect of the size of the log files can be seen in Fig. 10. This plot was created by running the stage discovery algorithm on randomly selected parts of the hardware comparison log. Adding more data dramatically improves the accuracy of stage

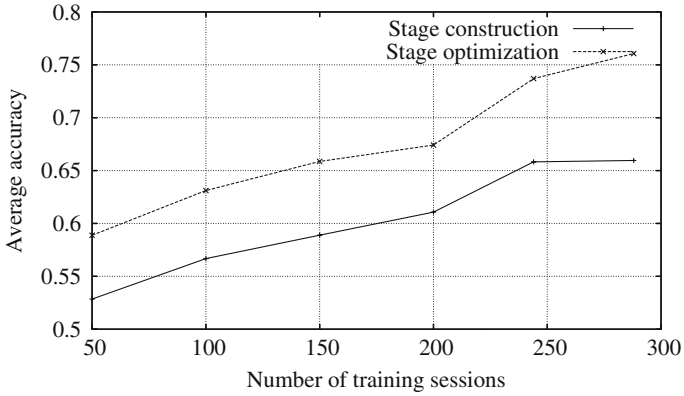

Fig. 10 The average accuracy with various numbers of sessions taken from thelog data of the hardware comparison experiment

construction and stage optimization. The figure suggests that at 288 sessions the accuracy of the classification has not yet reached a maximum and can be improved by adding more data.

In conclusion, we found strong indications that the various page types of the hardware comparison site are used during different navigation stages. The stage discovery algorithm is capable of finding the foundations of the stage structure, although more data is necessary to automatically determine the optimal number of stages. The discovered stage structure can be used to build a stage-oriented menu, which matches the users' search patterns (see Sect. 8). Such a menu might provide better guidance to the users than the currently available topic-based menu.

7 Analysis of the sensitivity of the method

The previous sections discussed applications of our method to pages taken from existing sites. In this section, we analyze the sensitivity of the method to several characteristics of the data using artificial data.

The behavior of users is simulated with a finite state automaton. The automaton consists of an ordered set of states and a transition function. The states in the automaton correspond to navigation stages. The transition probabilities between the states stand for the probabilities of going from a page in one stage to a page in another stage. The transition probabilities are determined by three parameters: the probability of staying in the same stage, p_{stay}, the probability of going to the next stage, p_{proceed}, and the probability of going to any other stage, p_{jump}. Each state consists of a set of pages. All pages in a state have equal probability of being visited. Thus, the probability of visiting a page p in stage s is the probability of going to stage s divided by the number of pages in s. Figure 11 shows an example of an automaton with three states.

The automata are used to generate sets of user sessions (log files). The generation process starts with an empty session in the start state. A state transition is performed with the probabilities determined by the transition function. If the new state is a content state a page is randomly selected from the new state and added to the session. Then a second state transition is performed, etc. A session is complete when the stop state is reached.

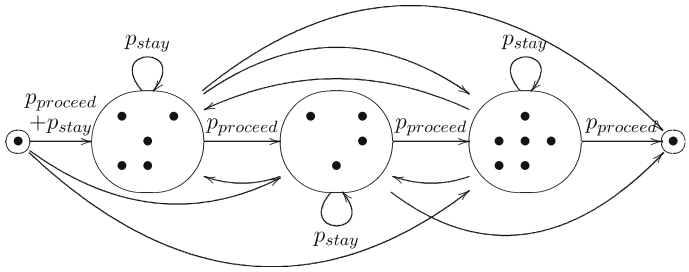

Fig. 11 A simulation model with three content states. States are represented by circles, pages in the states by dots. All unlabeled arrows have probability $p_{\text{jump}}/2$

Simulation models with various characteristics are used to generate data sets for the sensitivity tests. In each case the reference data set is the one that is most similar to the SeniorGezond data. Like the SeniorGezond site, the model for the reference set has 3 stages with respectively 10, 20 and 60 pages. It is used to generate 244 sessions. Furthermore, the probability of going to a random stage (p_{jump}) is set at the value found in the Seniorgezond data, 0.136. The values of p_{stay} and p_{proceed} are adjusted, so that the average length of the sessions in the reference data set becomes equal to the average length of the SeniorGezond sessions.

7.1 Finding the number of stages

To determine under which conditions the right number of stages is found we used the simulation model to generate log files with various numbers of stages and various numbers of sessions. We had the stage discovery algorithm choose between models with one to six stages. We repeated each experiment 50 times and evaluated in how many cases the algorithm was able to find the correct number of stages.

Figure 12 shows the part of the log files for which the correct number of stages was found when the real number of stages was 3. Best results are achieved with an α between 0 and 0.5, in other words when the likelihood of the Gaussian mixture was weighted more heavily than the proportion regular transitions. Within this range the algorithm is robust against small changes in the value of α. This confirms our claim that the α of 0.25 can be used safely in new domains. The estimation of the number of stages becomes much more accurate if more training sessions are available. From Fig. 13 we can see why: when more data is available each page is visited more often so that the deviations of the ARPs of the pages in the various stages are smaller. This results in larger 'gaps' between the ARPs of the pages from different stages which makes the stages more easily separable.

In the next experiment, we varied the number of stages by adding more stages with 20 pages between the first and the last stage. As visible in Fig. 14, the higher the number of stages, the more difficult it is to find the correct number of stages. When there are more stages, the means of the ARPs of the pages of the stages lie closer together, while the variance does not change. This increases the overlap between the stages which makes the individual stages harder to distinguish. To be able to discover models with more stages, better estimations of the ARP values are necessary. As Fig. 14 shows, this can be accomplished by acquiring larger log files.

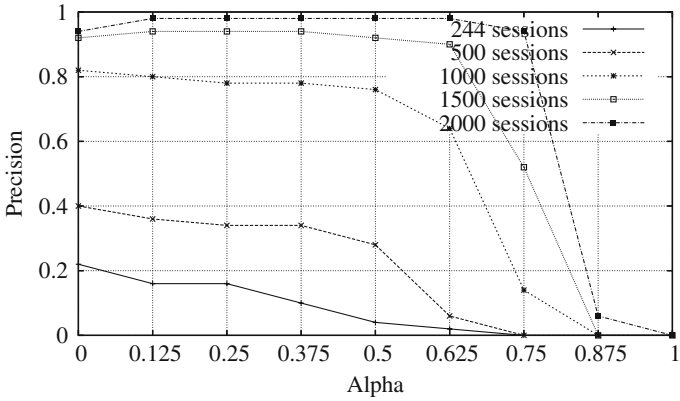

Fig. 12 Part of the experiments in which the correct number of stages is found with various numbers of sessions and various values of α

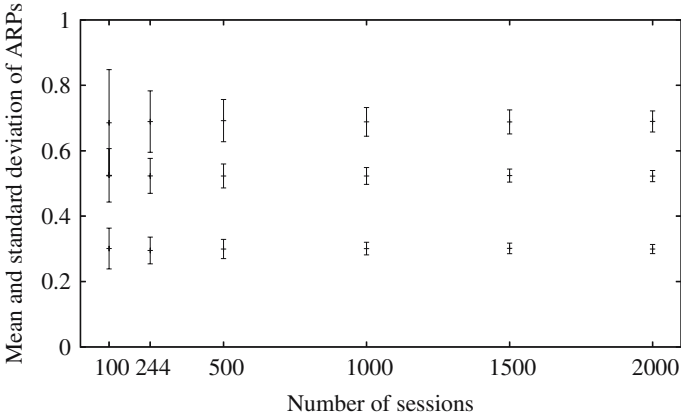

Fig. 13 The average mean and standard deviation of the ARPs of the pages from three stages and various amounts of training sessions

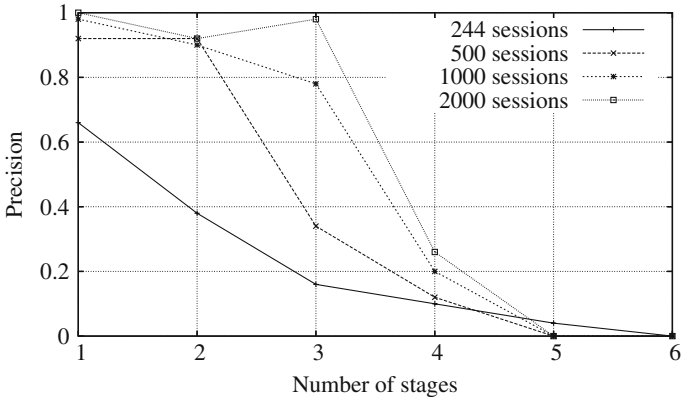

Fig. 14 Part of the experiments in which the correct number of stages is found with various numbers of stages, various numbers of sessions and an α of 0.25

7.2 Page classification

In this section we evaluate the accuracy of the stage assignment when the number of stages is known. All presented accuracies are averages over 50 generated log files.

First, we look at the effects of stage construction and stage optimization on the logs generated with the reference model. Figure 15 plots the distribution of the accuracy over 50 runs. After stage constructions most runs have an accuracy of around 0.7.The optimization step makes some runs much better and others much worse. This is a direct consequence of the bootstrapping method. When enough pages are classified correctly, misclassified pages have a large probability of occurring between two correctly classified pages and being fixed. On the other hand, when too many pages become misclassified, the stage of correctly classified pages can be changed, so that even more pages become misclassified. This 'snowball' effect results in large numbers of very good and very bad runs and relatively small numbers of mediocre runs.

In the second experiment, we varied the number of sessions per log file. Figure 16 shows the accuracy after stage construction and stage optimization. Both accuracies are higher when more training data is available. The effect on the optimized accuracy is stronger, because bootstrapping benefits from more data as well as a better initialization. These effects are similar to the ones found for the hardware comparison data in Sect. 6.

We varied the total number of pages while keeping the ratio between the numbers of pages in the three stages fixed. The results (after optimization) are presented in Fig. 17. If there are more pages, the available data per page is less, which results in a decrease in construction accuracy. When there are many pages the optimized accuracy suffers from the lower construction accuracy. At the same time more pages also mean that there is more data available for the bootstrapping phase. As can be seen in Fig. 17, these two opposite effects make that the algorithm performs optimal when the number of pages is about 50.

In the next experiment again the number of pages was varied, but in this case pages were only added to the second stage. Figure 18 shows that the classification accuracy decreases when the number of pages in the stages become unbalanced. There are two reasons for this effect. First, if one stage has much more pages than the others, there is

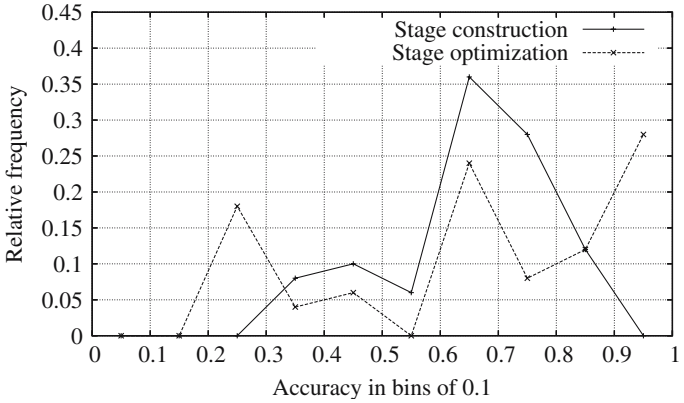

Fig. 15 The distribution of the accuracy over 50 simulation runs

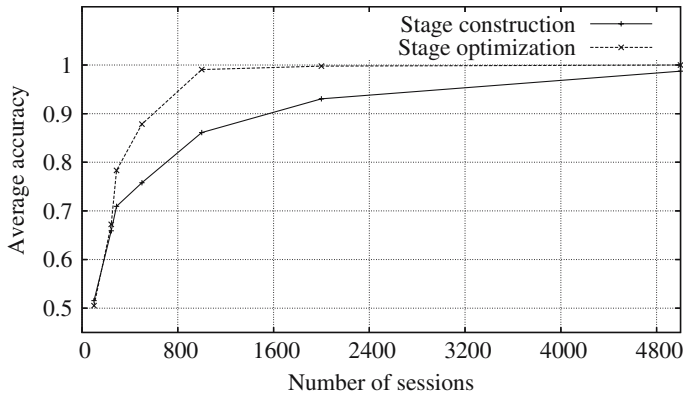

Fig. 16 The average accuracy with various numbers of sessions

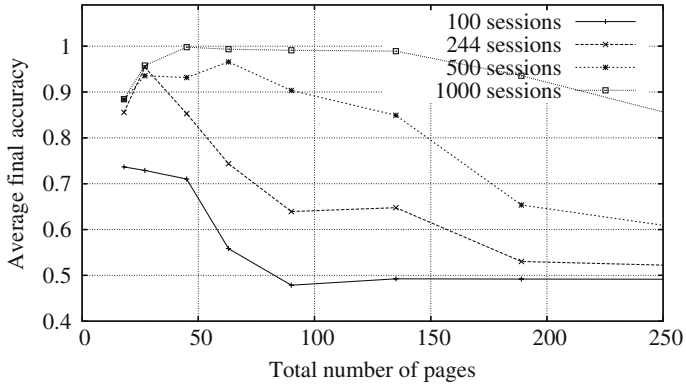

Fig. 17 The average accuracy with various numbers of pages (varied in all stages)

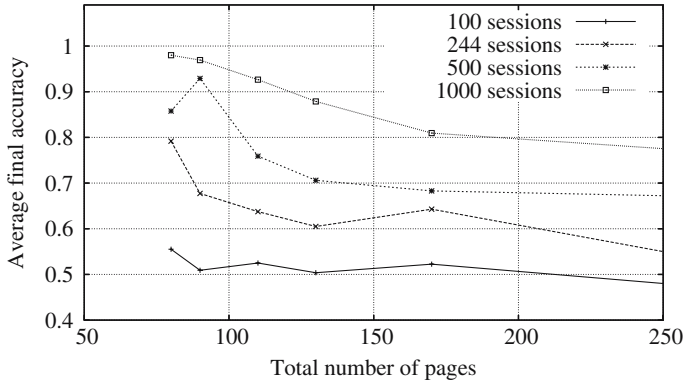

Fig. 18 The average accuracy with various numbers of pages (varied in the second stage)

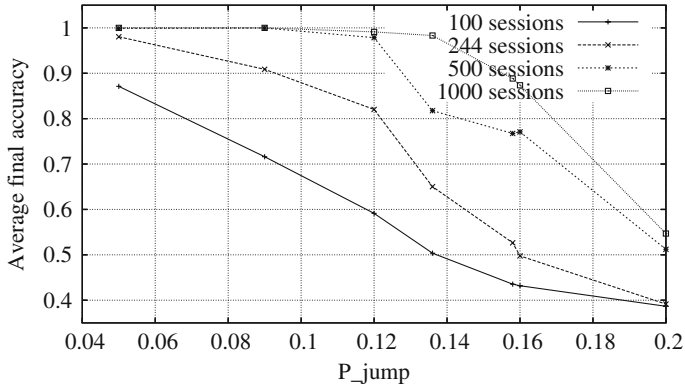

Fig. 19 The average accuracy with various values for P_{jump}

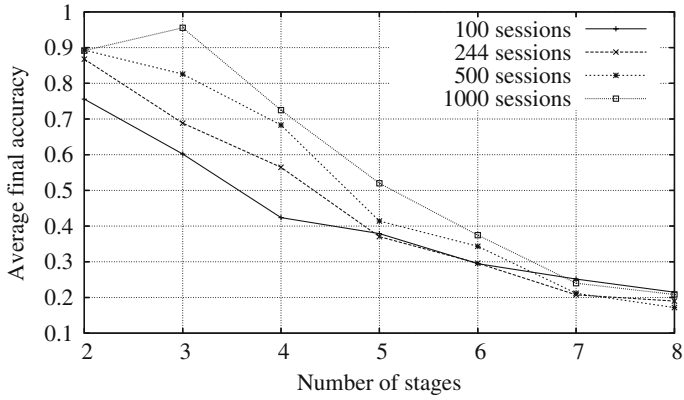

Fig. 20 The average accuracy with various numbers of stages

relatively little data about the pages in the large stage. The large number of imprecise ARP values hinders the stage construction. Second, EM assigns a large probability to the large stage. Because of this more pages from the smaller stages are classified incorrectly as pages from the large stage than vice versa. During bootstrapping this effect is magnified, so that the large stage 'swallows' the smaller stages. Fortunately, the figure also shows that these effects are less likely to occur when more data is available.

We made the behavior of the simulated users less predictable by increasing the probability of making irregular stage transitions (p_{jump}). Figure 19 shows the results of varying the value of p_{jump}, while keeping the ratio between p_{stay} and p_{proceed} constant. Imprecise ARP values resulting from large numbers of irregular transitions reduce the construction accuracy. In the optimization step the algorithm suffers both from the reduced construction accuracy and from the many irregular transitions in the data. All together, the accuracy drops when the percentage irregular transitions exceeds a certain maximum, but irregularity can be compensated for by adding more log data.

In the last experiment, the number of stages was varied by adding more stages with 20 pages between the first and the last stage. From Fig. 20 we can see that there is

a maximum number of stages that can be learned with a certain amount of training data. With high numbers of stages the stage boundaries become very tight compared to the deviation of the ARPs. Since more data makes the deviation smaller, more stages can be learned if more training sessions are available.

In summary, the algorithm appears to be sensitive to irregularities in the data and the complexity of the site and the navigation. However, these problems can be overcome by providing more training data. This is a promising result, as log files of web sites are typically very noisy but also extremely large.

8 Building problem-oriented navigation structures

The previous sections discussed how pages of a web site can be divided into navigation stages on the basis of the pages' usage. In this section we demonstrate how a discovered stage structure can be used to create a menu that guides a site's visitors through the navigation stages.

In Fig. 1, we showed the structure of the problem-oriented menu of the Senior Gezond site. The tree like structure can be decomposed into two orthogonal structures. The vertical layers represent the navigation stages: the left most layer contains the *problem* pages, the middle layer the solutions and the right most layer the products. The horizontal structure represents the pages' topics. Below each *problem* page we find the *solution* pages that treat the same subject as the *problem* page and below each solution we find the products that implement the solution. The topic structure and the stage structure are orthogonal: topics stretch over multiple stages and stages contain pages from all topics.

To construct a problem-oriented menu for a site one needs to divide the site's pages along both dimensions. The stage discovery algorithm can be used to form the stages. The topics can be taken from the site's topic-based menu, if such a menu is available. Otherwise, a topic structure can be found with traditional content- or usage-based clustering methods, for example with the ones used in Mobasher et al. (2002), Perkowitz and Etzioni (2000)and Pierrakos and Paliouras (2005).

The experiments in section 5 showed that the stage discovery algorithm was able to find the stage structure of the SeniorGezond menu. By combining the stages with the site's topic structure the problem-oriented menu of the SeniorGezond site can be reconstructed. Ezendam et al. (Ezendam et al. 2005; Alpay et al. 2005) showed that the addition of this menu to the SeniorGezond site made it significantly easier for the users to find their way through the site. This shows that at least in this case the stage discovery algorithm was able to find a valuable and effective stage structure.

To demonstrate that the stage discovery algorithm also produces meaningful menus in other domains we built a problem-oriented menu for the hardware comparison site. We used the stage structure with three stages as it was discovered in Sect. 6. The topic structure was created with the PACT approach of Mobasher et al. (2002). We did not evaluate other clustering techniques, since our goal was not optimize the topic structure but to demonstrate how stages and topic clusters are combined into a stage-oriented navigation structure. According to the PACT methodology, the sessions were first clustered on the basis of the cosine similarities between the vectors of visited pages. Subsequently, each session cluster was characterized by the pages that were visited frequently in the sessions in the cluster. This resulted in 18 partially overlapping page clusters.

Table 6 Part of the stage and topic structures for the hardware comparison site with a few classified pages

	Topic cluster 1	Topic cluster 2	...
Stage 1	How_to_Buy_a_Printer_–_Shopping_Tips.html How_to_Buy_a_Printer_–_The_ Big_Picture.html How_to_Buy_a_Printer_–_The_ Specs_Explained.html	How_to_Buy_a_Digital_Camcorder_–_Introduction.html Affordable_Camcorders.html How_to_Buy_a_Digital_Camcorder_–_The_Specs_Explained .html	
Stage 2	Top_10_Ink_Jet_Printers_–_Chart.html Top_5_Affordable_All–purpose_Printers_–_Chart.html Top_10_Ink_Jet_Printers_–_List.html	Top_9_Digital_Camcorders_–_List.html JVC_GR–D72US.html Top_9_Digital_Camcorders_–_Chart.html	
Stage 3	Canon_i455_Desktop_Photo_ Printer.html Canon_Pixma_iP1500.html Canon_i860_Desktop_Photo_ Printer.html	Sony_DCR–HC20_MiniDV_Handycam.html Sharp_VL–Z800U.html Panasonic_PV–DV953.html	

Table 6 shows 2 topics of the combined stage and topic structures for the hardware comparison site. Due to space limitations not all pages in the visible cells are shown. The stage classification in Table 6 is not perfect. The first stage of topic 2 contains the *news* page, 'Affordable_Camcorders.html' and the *overview* stage contains the *product* page 'JVC_GR–D72US.html'. In spite of these misclassifications, overall the intended stage structure is clearly visible. The topic clusters are also easy to interpret: cluster 1 contains pages about printers and cluster 2 contains pages about digital camcorders.

Figure 21 shows a part of the menu that was created from the matrix. The stage headings are added by hand. In the situation of Fig. 21 the user has first clicked on 'How to Buy a Printer–The Big Picture'. When he clicked the link the page was shown and the menu below the link opened to reveal three links to *overview* pages. From these the user selected 'Top 5 Affordable All-purpose Printers–Chart' and got access to three *product* links. Finally, he selected the product 'Canon i860 Desktop Photo Printer'. This example demonstrates that the menu indeed behaves as intended: at each step it shows the pages that are both relevant for the user's task and match the user's navigation stage.

The results of the user experiment presented in section 6 indicated that users of the hardware site tend to go through a number of stages when searching for information. The problem-oriented menu supports this pattern by showing the pages in their natural order. The results in Ezendam et al. (2005) and Alpay et al. (2005) show that this can considerably facilitate the users navigation processes. Therefore, we believe a problem-oriented menu like the one presented here can be a valuable addition to the hardware comparison site. However, detailed usability studies are necessary to confirm that the stage menu works also in the hardware domain. In specific, the objective navigation efforts and the subjective experiences of visitors using a problem-oriented menu should be compared to the experiences of visitors using a topic-based structure.

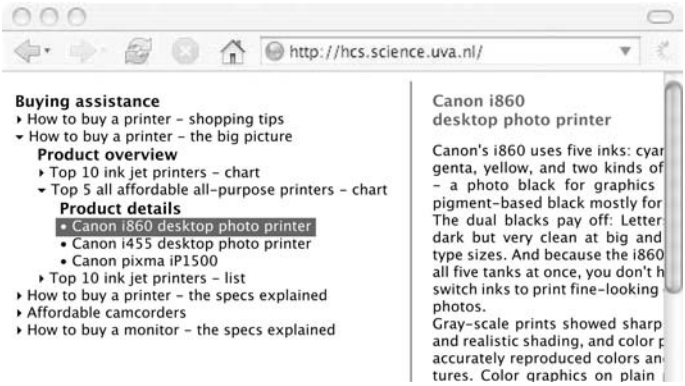

Fig. 21 Part of the automatically constructed stage-oriented menu for the hardware comparison site

Besides the presented navigation menus, stage structures can also provide order information in other types of menus or even other navigation means. The presented menus are difficult to use if too many pages belong to the same stage and topic. In this case the lists of menu items should be subdivided into subtopics. A user now first selects a category before she sees the pages from her current stage and topic. An example of an alternative problem-oriented navigation means is a wizard style interface that helps users step-by-step to formulate their information needs. The stages can also be used to rank the results of a site search engine or recommender system in such a way that links that match the user's current navigation stage appear at the top of the list. In any case the navigation structures allow the users to navigate trough the site and the stages assist this process by presenting the pages in the right order.

9 Conclusions and discussion

Most web sites provide a topic-based menu as the primary means of navigation. They aim to make all content reachable in a small number of clicks. Topic-based structures are efficient when users know what information they want and the menu is only used to reach the relevant links. However, users do not always know exactly what they are looking for and what the site has to offer. For these users a topic-based navigation structure is not ideal, as they often experience great difficulties translating their information needs into the site's topics Alpay et al. (2004).

In Ezendam et al. (2005) an alternative navigation structure is presented that not only aims at making the information reachable but also guides the users through the available information. Ezendam shows that this structure significantly facilitates the navigation process in the domain of falling accidents.

Based on the structure in Ezendam et al. (2005) we created the stage model presented in this paper. According to this model users' navigation processes can be decomposed into a number of navigation stages. Each stage is characterized by a distinct set of pages. We provide an efficient and scalable algorithm to learn the parameters of the model for a given web site. The algorithm divides the set of pages of the site into a number of navigation stages on the basis of the observed usage of the pages in site's server logs. We demonstrate how a filled in model can be used to create a problem-oriented navigation menu.

The stage discovery algorithm was evaluated in a series of experiments. The algorithm proved to be able to find stage structures in log data from user experiments conducted in two very different domains. Simulation experiments showed that the algorithm is able to discover stages in noisy data as long as enough log data is provided. These results indicate that the stage discovery algorithm is an adequate method to automatically create problem-oriented navigation structures for a wide range of domains.

Although the results of the stage discovery algorithm are encouraging, some issues remain unsolved. First, the sites that were used for the data collection in the user experiments were stripped of all navigation structures to exclude the possibility that the discovered patterns were imposed by the sites' structures. However, for real web sites this is a rare situation. Web sites for which a stage-oriented menu is created most likely provide some navigation means that allow the users to browse through the site. The algorithm can be applied without modification to sites which structures do not force the users to follow a specific path through the site, such as topic-based menus and site search engines. More problematic are sites with in-text links and other sequential structures that require that the users click through a series of pages before they can reach the pages they actually want to visit. Such structures bias the navigation behavior of the users and consequently the patterns found in the log files. More research is needed to determine how large the influence of these biases is on the discovered stage patterns and how they can be compensated for.

From Ezendam et al. (2005) and Alpay et al. (2005) we know that in the falling accident domain the stage-oriented menu has a positive effect on the users' navigation experiences. However, we do not know for sure whether stage structures will have the same effects in other domains. We expect that in any domain where users enter a site without knowing exactly what information they need the users will navigate from pages which give an overview of the available options to pages with more specific content. We believe that for these users problem-oriented menus can provide useful guidance, but detailed usability studies are necessary to confirm this. In particular we need to compare the navigation efforts and navigation experiences of visitors using a stage-oriented structure to the efforts and experiences of visitors using a topic-oriented structure.

Other directions of further research involve the extension of the stage discovery algorithm. One point that needs improvement is that in the current version all pages of a site are assigned to a navigation stage. In our experiments we found that sometimes there are pages that do not belong to a particular stage but are used throughout the sessions. The next version of the algorithm will use the standard deviations of the pages' ARP values to determine whether a page should be assigned to a stage or placed somewhere outside the stage-oriented menu.

Another point that needs to be addressed is the creation of menus for sites where some users do not know what they are looking but others have very specific information needs. This situation poses two challenges. For the discovery algorithm the stage patterns become harder to distinguish because they are not visible in the sessions of the users with specific questions. Menu creation becomes more complicated because the menu should now accommodate both user types. We are planning to handle the first challenge by running the algorithm only on sessions which at first sight seem to follow a meaningful pattern. This extra bootstrapping cycle will boost the algorithm's robustness to noise and at the same time make it more efficient. An easy solution to the second challenge is to include two separate menus for the two user types.

However, choosing between the two menus requires extra effort on the users' part. The best solution would be to identify the users' search types very early in the sessions and adapt the interface style to their personal needs. We are currently investigating the possibility to recognize various users types.

The last and maybe most tricky issue is the creation of labels for the stages. Like most clustering methods the stage algorithm creates groups of pages, but a human is needed to interpret the groups and provide labels. Automating the labeling involves identifying the role that the pages play in the users' navigation. Topic clusters can to some extent be characterized by words that occur frequently on the pages or in the pages' annotations, e.g., Perkowitz and Etzioni (2000), Pierrakos and Paliouras (2005). For stage clusters extracting the labels from the pages' contents is more complicated, because the pages' roles are often not mentioned explicitly.

Acknowledgements We would like to thank Stephan ten Hagen for his contributions during the early stages of this research. Furthermore, we thank Laurence Alpay, Nicole Ezendam, Ton Rövekamp, and Marcel Hilgersom for their valuable comments. This research is supported as ToKen2000 project by the Netherlands Organization for Scientific Research (NWO) under project number 634.000.006.

Appendix

A Example assignment from the SeniorGezond experiment

Mister Jansen is 82 years old. He is living with his wife in an apartment for seniors on the ground flour. He has got up early to visit his grandson's birthday. At half past nine his daughter picks him up by car. He had to give up cycling and driving years ago because of his poor sight and rheumatism. He walks towards the car on his daughter's arm and opens the door. When he tries to bend over to enter the car, he suddenly slips. He makes a nasty fall on his elbow and feels a severe pain. Scared to death his daughter calls an ambulance.

Soon after their arrival in the hospital mister Jansen is seen by a doctor. The doctor takes pictures of mister Jansen's arm and concludes that it is broken at two places, but the segments are not displaced. The arm is placed in a cast and mister Jansen receives a recipe for pain killers. From all the events mister Jansen is very tired and his daughter decides to bring him home. On the way home they drive by the drugstore to pick up the pain killers.

The next day mister Jansen feels already much better, but he is still worried about the whole event. He visits the SeniorGezond site to see whether the site can help him to make sure accidents like this won't happen again. Play the visit of mister Jansen to the SeniorGezond site.

B Example assignment from the hardware comparison experiment

You have a small law firm with four employees. You have bought a new computer with a new system to document your cases. To make sure the old paper documentation does not get lost, you decide to buy a scanner and save it all on cd. You want a scanner suitable for this purpose, but you don't know anything about scanners. Find an appropriate scanner.

References

Alpay, L.L., Toussaint, P.J., Ezendam, N.P.M., Rövekamp, A.J.M., Graafmans, W.C., Westendorp, R.G.J.: Easing internet access of health information for elderly users. Health Informatics J. **10**(3), 185–194 (2004)

Alpay, L.L., Ezendam, N.P.M., Zwetsloot-Schonk, J.H.M.: Final report of the Geriwijzer/Senior Gezond project.Technical Report, Leiden University Medical Center, Leiden, The Netherlands (2005)

Anderson, C.R., Domingos, P., Weld, D.S.: Adaptive web navigation for wireless devices. Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, pp. 879–884. Seattle, Washington, USA (2001)

Anderson, C.R., Horvitz, E.: Web montage: a dynamic personalized start page. Proceedings of the 11th International Conference on World Wide Web, pp. 704–712. Honolulu, Hawaii, USA (2000)

De Bra, P., Calvi, L.: AHA! an open adaptive hypermedia architecture. New Rev Hypermedia Multimedia **4**, 115–139 (1998)

Brusilovsky, P.: Adaptive hypermedia. User Model. User-Adapt. **11**(1–2), 87–110 (2001)

Brusilovsky, P., Eklund, J., Schwarz, E.: Web-based education for all: a tool for developing adaptive courseware. Proceedings of the Seventh International World Wide Web Conference, pp. 291–300. Brisbane, Australia (1998)

Cadez, I., Heckerman, D., Meek, C., Smyth, P., White, S.: Model-based clustering and visualization of navigation patterns on a web site. Data Mining Knowledge Discovery **7**(4), 399–424 (2003)

Carroll, J.D.: Individual differences and multidimensional scaling. Multidimensional Scaling: Theory Appl. Behav. Scio. **1**, 105–155 (1972)

Choo, C., Detlor, B., Turnbull, D.: working the web: an empirical model of web use. Proceedings of the 33rd Hawaii International Conference on System Sciences. Maui, Hawaii (2000)

Clancey, W.: Heuristic classification. Artif. Intell. **27**(3), 289–350 (1985)

Cooley, R., Mobasher, B., Srivastava, J.: Data preparation for mining world wide web browsing patterns. J. Knowledge Inform. Syst. **1**(1), 5–32 (1999)

Deshpande, M., Karypis, G.: Selective Markov models for predicting web page accesses. ACM Trans. Internet Technol. **4**(2), 163–184 (2004)

Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. Roy. Stat. Soc. **39**, 1–38 (1977)

Domshlak, C., Joachims, T.: Efficient and non-parametric reasoning over user preferences. This issue.

Ezendam, N.P.M., Alpay, L.L., Rövekamp, A.J.M., Toussaint, PJ.: Enhancing accessibility of the content of a fall prevention website for elderly: a cross sectional study. Technical Report, Leiden University Medical Center, Leiden, The Netherlands (2005)

Herder, E.: Sniffing around for providing navigation assistance. Proceedings of the Workshop on Adaptivity and User Modeling in Interactive Systems, pp. 20–24. Berlin, Germany (2004)

Hollink, V., van Someren, M., ten Hagen, S.: Discovering stages in web navigation. Proceedings of the 10th International Conference on User Modeling, pp. 473–482. Edinburgh, UK (2005a)

Hollink, V., van Someren, M., ten Hagen, S., Wielinga, B.: Recommending informative links. Proceedings of the IJCAI-05 Workshop on Intelligent Techniques for Web Personalization, pp. 65–72. Edinburgh, UK (2005b)

Jin, X., Zhou, Y., Mobasher, B.: Task-oriented web user modeling for recommendation'. Proceedings of the 10th International Conference on User Modeling, pp. 109–118. Edinburgh, UK (2005)

Mobasher, B., Dai, H., Luo, T., Nakagawa, M.: Discovery and evaluation of aggregate usage profiles for web personalization. Data Mining Knowledge Discovery **6**, 61–82 (2002)

Perkowitz, M., Etzioni, O.: Towards adaptive web sites: conceptual framework and case study. Artif. Intell. **118**, 245–275 (2000)

Pierrakos, D., Paliouras, G.: Exploiting probabilistic latent information for the construction of community web directories. Proceedings of the 10th International Conference on User Modeling, pp. 89–98. Edinburgh, UK. (2005)

Pierrakos, D., Paliouras, G., Papatheodorou, C., Spyropoulos, C. D.: Web usage mining as a tool for personalization: A survey. User Model. User-Adapt. **13**(4), 311–372 (2003)

Pirolli, P., Fu, W.-T.: SNIF-ACT: a model of information foraging on the world wide web. Ninth International Conference on User Modeling, pp 45–54. Johnstown, USA (2003)

Pitkow, J.E., Pirolli, P.: Mining longest repeated subsequences to predict world wide web surfing. Proceedings of the Second USENIX Symposium on Internet Technologies and Systems, pp. 139–150. Boulder, USA (1999)

Sarukkai, R.: Link prediction and path analysis using markov chains. Proceedings of the Ninth International World Wide Web Conference, pp. 377–386. Amsterdam, The Netherlands (2000)
Schwab, I., Pohl, W.: Learning user profiles from positive examples. Proceedings of the ACAI'99 Workshop on Machine Learning in User Modeling, pp. 15–20. Chania, Greece (1999)
Ypma, A., Heskes, T.: Automatic categorization of web pages and user clustering with mixtures of hidden markov models. Lecture Notes in Computer Science, **2703**, 35–49 (2003)
Zhu, T., Greiner, R., Häubl, G.: Learning a model of a web user's interests. Proceedings of the Ninth International Conference on User Modeling, pp. 65–75. Johnstown, USA (2003)

Authors' vitae

Vera Hollink is a Ph.D. student at the Human-Computer Studies Laboratory of the University of Amsterdam. In 2002 she received her M.S. degree in Artificial Intelligence from the same university. During her master's studies she has been involved in projects on information retrieval and natural language processing. Her Ph.D. project focuses on the use of machine learning techniques to automatically improve link structures of web sites. The current work reports on the outcomes of her Ph.D. work.

Maarten van Someren is Assistant Professor of Artificial Intelligence at the University of Amsterdam, working mainly in the area of Machine Learning. Since 1985 he has been a senior researcher and project coordinator of the projects in Machine Learning, Knowledge Acquisition and Cognitive Modeling. The research in this volume is part of an effort to apply Machine Learning to self-improving interfaces.

Bob J. Wielinga (1945) studied physics at the university of Amsterdam, where he was awarded a Ph.D. degree cum laude in 1972 for a thesis in nuclear physics. In 1977, Wielinga was appointed senior lecturer at the Department of Psychology of the University of Amsterdam. Since 1983,Wielinga has performed research on the methodology of knowledge-based system design and knowledge acquisition. In 1986, Wielinga was appointed full professor of Social Science Informatics in the Faculty of Psychology. In this capacity Wielinga was and is team leader of several research projects, including KADS, ACKnowledge, REFLECT and KADS-II. He was one of the main contributors to the development of the KADS methodology for knowledge based system development.