# Efficient Representation of the Lifelong Web Browsing User Characteristics

Márius Šajgalík, Michal Barla, Mária Bieliková

Institute of Informatics and Software Engineering Faculty of Informatics and Information Technology Slovak University of Technology, Bratislava, Slovakia {sajgalik, barla, bielik}@fiit.stuba.sk

**Abstract.** Client-based user modelling has already been studied and clearly has its place among generic approaches to the user modelling. It is especially advantageous for lifelong user modelling as it can support the modelling in any time and any place including consideration of user privacy. Emergence of web browser extensions opens up possibilities of pure browser-based realisation of client-based user modelling. In this paper, we focus on the efficient representation of a generic user modelling framework in form of a browser extension. Efficiency is crucial also from the lifelong perspective. We propose an efficient method of lifelog indexing and modelling various user characteristics inside the web browser. We evaluated properties of proposed representation and describe its applicability in some common use cases.

# 1 Introduction

In this modern era of ubiquitous computers in our everyday life, the need of ubiquitous lifelong user modelling approaches has emerged. With regard to the shift in humancomputer interaction from desktop computing to mobile, the decentralised client-based approaches have been encouraged to support variety of adaptation goals. From this ubiquitous perspective, the web browser seems to be an ideal choice for lifelong user modelling, since we use it everywhere, every day, on each device – be it a desktop computer, laptop, tablet or mobile phone. The emergence of new web technologies like HTML5 and support for powerful extensions makes the web browser a capable platform suitable to perform user modelling and personalisation across the Web, while still keeping our user profile under our complete control, without the need to disclose our identity, browsing history or our user model to any third-party service.

From the user modelling perspective, we have a tremendous choice in regards of user actions within a web browser. Not only we get a complete surfing history, but we get also the context of all actions such as mouse movements or other tabs opened at the same time. On the other hand, being on a client side, we need to focus much more on the efficiency of all user modelling processes and of all data structures used to capture our user model, since we are rather limited in resources compared to server. We developed BrUMo<sup>1</sup> – a specialised web browser extension, which represents a browser-based instance of our user modelling and personalisation framework. According to statistics recorded within BrUMo, the average number of visited webpages can reach over 200 per day for some users. It is important to note that nowadays web pages often do not reload the entire content, but only its part is updated via AJAX. If we count all requests made by a single user, the average number of AJAX requests can grow up to over 600 per day. If we are to extract and store keywords or other metadata of each visited webpage and update it according to all changes made within a single visit to index user knowledge or interests, the efficiency is really crucial. We need not only an efficient storage of the model in terms of required allocated space and memory, but we also need to consider efficiency of perpetual retrieval of information from the model.

In this paper, we propose mechanisms to index various user characteristics captured in the user model in an efficient manner. We have extended the concepts of some fundamental data structures like Patricia trie to design a user and a domain interest tree, which provide two perspectives of the user model. Although the usage of such basic data structures as well as their extensions is much broader than its application in user modelling, we consider it very important to deal with the efficient representation of user characteristics in the web browser as its resources and possibilities are still limited there in comparison with server-based or cloud-based solutions.

# 2 Related Work

In [4] an analysis of multiple challenges of cloud-based user modelling in favour of clients is presented. The authors propose to decentralise and push user models down to the client side and describe a sample PaaS (Personalisation as a Service) architecture. In [1] authors discuss the possibility of scenario where server is used only to update the profile and to perform personalised response computation, while all information is stored locally. The authors of [7] present an architecture of a client-side framework to provide adapted content. However, they use additional storage and management servers. There is also other user modelling framework called PersonisJ [3], which is specially aimed at Android phone platform. There is also an evaluation of its efficiency, which shows how it is important and thus also supports the importance of our work. An approach to web search personalisation described in [11] is based on a model of user interests coming from the user's web browsing history. It is realized as Mozilla Firefox extension, but only to collect the information about user. All computational work is done on the server. However, the author in [11] states the possibility of implementing it solely client-side, which would avoid the privacy concerns arising with server-based approaches.

An exhaustive overview of what data can be captured directly in the browser is analysed in [14]. Another study in [5] shows what additional information about the user can be captured on client side by means of emerging Web 2.0 technologies. GINIS

<sup>&</sup>lt;sup>1</sup> Browser-Based User Modelling and Personalisation Framework - http://brumo.fiit.stuba.sk/

framework [16] is one representative of browser-based systems. It is a customised tabbed Internet Explorer browser using .NET framework. Although not explicitly stated, we assume they used MS SQL database to log user actions, which is supported by their claim that "it is easy to log high-granularity data using the provided .NET framework". From our perspective, GINIS framework has several disadvantages. It requires user to install and use some non-standard specialised browser, which for example does not guarantee to receive important updates unlike the standard official releases. Moreover, they focus just on their single goal (to classify content as interesting or not) and build a decision tree based on user behaviour data. They do not present any general user model or other mechanisms to index their raw logs. Somewhat more relevant to our work is an example of browser-based user modelling system in form of an extension presented in [8]. There, authors mention that they use HTML5-based SQL database to store two tables – one for keywords with their corresponding frequencies and second for the visited webpages with their visit frequencies. Although this approach has more general user model and all the advantages of browser extension, it fails to provide some additional information about user, which can be required in context of lifelog creation as well as in some common personalisation scenarios.

Yet another approach is to place the user modelling platform to the middle between the Web and a client in a form of specialised proxy. For instance, PeWeProxy is a proxy server, which builds a term-based user model representing the user's interests from keywords and terms automatically extracted from web pages passing through the proxy. It shifts the personalisation part to the client using personalisation scripts embedded into the browsed web pages [10]. Various research extensions like estimation of user interests [6] and web search disambiguation [9] have already been developed for it, though all of them are also run within the proxy server.

To summarise, it seems that there is a demand for pure client-side solutions to user modelling and personalisation, which address the problem of efficient representation of generic lifelong user model. Servers satisfy most of computational power and storage force requirements. Shifting to client however, requires more focus on this topic, since the possibilities and resources are generally much more limited.

# **3** Representing the User

Web (search) history analysis and keyword-based user models are becoming more and more popular solutions for user modelling [2]. Keywords representing users' interests are relatively easy to acquire and they could be easily presented to a user (to justify or explain personalisation, to enable a user to scrutinise her model and provide an explicit feedback upon it). At the same time, their lightweight semantics provides a solid basis for personalisation. A nice example can be found in [13], where author uses a lightweight folksonomy, which can be considered as a form of collaborative web surfing history, to infer similarity among users or visited documents.

We use keyword-based representation to express various user characteristics such as interests, knowledge, goals, context of work, etc. For example, user interests are represented as weighted vector of terms, where each term is linked to some URL address recorded in the user browsing history. There are multiple terms for each URL and similarly each term can be linked to multiple URLs. These links connecting terms with URLs are also weighted according to their mutual relevance. They denote the relevance of a web page at some URL to given term. Since the term represents a user interest, we can find out how interesting particular web page is by following the corresponding link. It is important to note the variability of terms that can stand for not just words extracted from read articles, but possibly other units like stems, lemmas or concepts. Similar logical representation can be applied to other user characteristics, e.g. terms represent knowledge concepts or particular goals in educational domain.

Currently, in our BrUMo platform, we use keyword extraction to infer user interests. To extract keywords from webpage, we combine multiple methods. First, we consider the content of webpage. We extract the article using Readability<sup>2</sup> and utilise browser's built-in functionality<sup>3</sup> to obtain raw text. In further pre-processing we tokenise the text into words using jspos<sup>4</sup> lexer, filter out stop-words and all words shorter than 3 characters and consider further only nouns as tagged by jspos POS tagger. With these feasible words extracted, we compute relevance of each word as an average of normalised TF-IDF [15] and TextRank [12]. We normalise the TF-IDF value by text length. Afterwards, we look at keywords meta-tag in HTML structure and propagate these keywords by doubling their relevance value. The IDF values were obtained from Google N-gram corpus<sup>5</sup>.

In this paper, we present two basic data structures to index user characteristics – user interest tree and domain interest tree. The user interest tree serves perfectly for indexing global user interest. However, a user can have different preferences in different domains. These are represented by so called local interests, since they are significant only within a particular domain. Domain interest tree is designed to index these local interests.

#### 3.1 User Interest Tree

User interest tree captures global user interests. By storing the terms representing those interests using a Patricia trie, we can easily execute all queries based on the term index, e.g., fast insertion/deletion or to iterate over all the terms stored in the tree in alphabetical order. To enable retrieval of the topmost relevant terms to get the most relevant global user interests we extended the basic structure using a labelling technique which enables us to speed up the tree node traversal in an order of term relevance.

Figure 1 depicts an example of such a labelled tree. It stores words and their relevance (in brackets) PEACE (2), PENCIL (3), PEWE (5), SEBE (4) and SET (0). We label each sub-tree by the relevance of the most relevant term in it. Thus, we can easily

<sup>&</sup>lt;sup>2</sup> Readability – https://code.google.com/p/arc90labs-readability/

<sup>&</sup>lt;sup>3</sup> textContent property – http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/core.html#Node3-textContent

<sup>&</sup>lt;sup>4</sup> JavaScript part-of-speech tagger - http://code.google.com/p/jspos/

<sup>&</sup>lt;sup>5</sup> Google N-gram corpus - http://storage.googleapis.com/books/ngrams/books/datasetsv2.html

retrieve the most relevant term in each sub-tree by following the path labelled by maximal value. The creation of such a tree is simple. The words are inserted in a common manner like into an ordinary Patricia trie. In addition to that, we update all vertices on the path from root to the inserted leaf node so that the above stated labelling rule holds true.

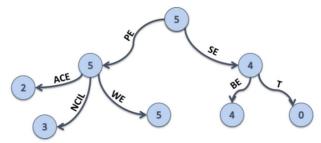


Fig. 1. Labelled Patricia trie

With such a labelled tree we propose following steps to iterate over all of the terms of a valid user interest tree in order of their relevance:

- 1. Initialise empty array of results and two heaps.
- 2. Initialise first heap by inserting the user interest tree root node in it. This heap always pops out the node labelled by maximal value.
- 3. Initialise empty second heap. This one always pops out the most relevant node.
- 4. Pop out node v from first heap. If it is empty, we have already traversed all nodes.
- 5. If v represents a term (not just prefix), push it into the second heap.
- 6. While there are nodes in the second heap with relevance not lower than label of v, pop them into the results array. If the desired count of results is reached, stop here.
- 7. Push all children of node v into the first heap and continue with step 4.

Every node is at worst once inserted and removed from each heap. Time complexity is therefore  $O(n \times m \times \log_2(n \times m))$  where *n* is the number of terms in the tree and *m* is the length of the longest term, which is asymptotically optimal. However, the advantage of this algorithm over a simple sorting of all terms is the ability to terminate it prematurely once we got the required number of terms. This can greatly reduce the running time of retrieval of just first *k* most relevant terms to  $O(k \times m \times \log_2(k \times m))$ . Note that node relabeling is done with insertion/deletion in O(m). Another advantage is that we can use multiple different labels, so that we can retrieve also the most recently added term to get the context of user's work. Additional labels are also important for managing the tree over longer time period and limit the overall tree size. Labels for the least relevant or the oldest terms can be used to remove such surplus terms from the tree since that could mean that user is interested in them no more.

### 3.2 Domain Interest Tree

Domain interest tree is similar to a user interest tree in its structure, but it is extended with some concepts of the generalised suffix tree. Unlike the generalised suffix tree, the domain interest tree does not need to store all the suffixes of strings, but only those suffixes that represent some subdomain. Thus, primary key in this tree is a URL address and its suffixes corresponding to different subdomains. Domain interest tree can also be labelled like in the case of the user interest tree, e.g., for each sub-tree, its root is labelled with a frequency value of the most frequent URL address within this sub-tree. Thus, we can easily determine the most frequent URLs.

In addition, we store first k most relevant interests (terms) in each node within the sub-tree rooted at this node. This allows us to retrieve efficiently local interests within various subdomains. Constant k represents a trade-off between performance and precision. Higher k means that more terms are propagated to the parent node for the price of lower performance. If we need more than first k most relevant terms in a particular subdomain, we can recursively nest further to search the child nodes' k most relevant terms until we reach the desired number of a user's local interests.

#### 3.3 Lifelong perspective

Despite the powerful built-in labelling technique, which is used to manage the tree and limit its size over longer time period, it is just not enough from the lifelong point of view. A single tree as it is, can be managed only within given time sliding window. The need of limiting the tree size arises from the fact that we are limited in memory and the naïve idea of simply building one huge tree out of the whole browsing history is indispensable.

Therefore, we propose to factorise the whole browsing history into multiple trees. In other words, we maintain one tree sliding over time and in regular time intervals, take a snapshot of it and store it into database. Moreover, we build a tree hierarchy out of these trees (user/domain interest trees) in order to get the most relevant terms for different time intervals at different abstraction level. We show a sample of such hierarchy in Figure 2. To create a tree on higher level of abstraction, we simply combine content of all underlying trees, but limit the size of the resulting tree to contain just the most relevant features. Using this tree hierarchy, we can easily get the overall lifelong-spanning characteristics, which are stored in the root.

# 4 Application and Evaluation of User Model

We designed our indexer with its underlying indexing data structures to achieve the best possible time and memory complexity for all needed operations to be ready for real-world lifelong user modelling and personalisation in a web browser environment. We evaluated main characteristics of the proposed data structures. User interest tree enables us efficiently:

- to retrieve the most relevant terms (e.g., user interests, concepts, context)
- to retrieve the latest updated terms (e.g., temporal interests, fresh knowledge)
- to retrieve the relevance of given term (e.g., how much is it interesting for user, how well she grasps given knowledge concept)

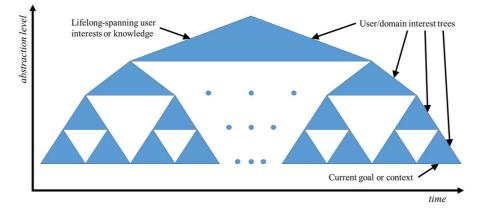


Fig. 2. Hierarchy of trees for different time and abstraction level.

- to retrieve the most relevant web pages from user web browsing history for a given term (e.g., which pages are interesting, contain particular concepts)
- to retrieve the least relevant or the oldest accessed term, which can be considered for removal (e.g., user has no more this kind of interest, has forgotten learned knowledge, changed context)

Domain interest tree broadens these possibilities by enabling us efficiently:

- to retrieve the most relevant terms within given domain (e.g., local interests)
- to retrieve the most relevant web pages from user web browsing history within a given domain (which pages are interesting within a given domain, are representative for a given concept)
- to retrieve the most relevant domains/subdomains (e.g., which domain is the most interesting one, contains the best grasped concepts

All these queries are just a top of the hill of possibilities. They represent only the most generic ones sufficient to cover most of common personalisation scenarios. There is much more of them depending on the chosen labels. All these queries (except those based on a simple term retrieval, which is linear to its length) can be done in time  $O(k \times m \times \log_2(k \times m))$  as we already analysed above (*k* being number of topmost terms and *m* the length of the longest of them) using presented algorithm.

To wrap up this analysis and solve yet unanswered common question on magnitude of time complexity constant, we performed several experiments on a sample scenarios executed directly in the browser (see Table 1). We compare our indexer to various browser implementations of JavaScript object, which can be considered as the state-ofart implementations of commonly used associative array data structure. Note that operations like retrieval of the most relevant terms is not supported by default in JavaScript object. JavaScript object allows only one type of index to be used, which is the term itself in this case. To simulate retrieval of the most relevant terms, we need to copy all items into an additional array and sort it by relevance. Therefore, the time efficiency is the same regardless of how many items we retrieve. Such operations are rather demanding in various personalisation scenarios, since we want to recommend only the very relevant things (like movies, articles) and we do not need to bother with considering some less relevant interests (excluding local user interests, i.e. the most relevant terms within some particular domain, which are discussed in section about domain interest tree).

Browser	Data structure	Term in-	Retrieval	Retrieval of 10	Retrieval of 50
		sertion	by term	most relevant	most relevant
				terms	terms
Chrome 17	Our indexer	1.77 μs	0.53 μs	200 µs	560 μs
Chrome 17	JS object	0.57 μs	0.04 µs	8650 μs	
IE 9	Our indexer	4.42 μs	3.54 μs	750 μs	1540 μs
IE 9	JS object	2.37 μs	1.4 μs	34610 μs	
Firefox 11	Our indexer	9.33 μs	8.06 µs	1090 µs	3410 µs
Firefox 11	JS object	1.7 μs	0.56 µs	13840 μs	

Table 1. Run time comparison of selected data structures in different browsers

In our tests we used collection of 4 204 weighted terms (unique keywords potentially representing some reasonable interest) which were extracted by our BrUMo framework from web pages of a real user browsing history. All test results are given as an average of 100 test runs performed on Dell laptop with 2 GHz Intel Core 2 Duo under Windows 7 64-bit. In retrieval of the most relevant terms, our indexer clearly outperforms all browsers' implementations of JS object. Nonetheless, term insertion and term retrieval is still reasonably fast (in order of microseconds), which is sufficient for real-time usage in collaborative distributed lifelong personalisation. Interestingly, the ordering of browsers' performance differs between our indexer and JavaScript object.

# 5 Conclusions

In this paper, we presented an efficient representation of user characteristics suitable for limited web browser environment. We described a powerful labelling technique to index various aspects of user model. We proposed a method of lifelog indexing as well, which makes it a complete lifelong user modelling component. We demonstrated the real-world performance of the proposed indexer within BrUMo framework.

We focused on a widespread web browser environment, which implies computational limits and explains the importance of designing such low-level mechanisms for user model representation. Although straightforward in their principles, they are powerful enough to accomplish various recommender tasks and supports both collaborative and content-based filtering approaches commonly used in today's recommender systems. Since our experimental framework BrUMo enables communication among its multiple instances, users can be grouped together by comparing weighted vectors of their global interest (see section 3). Subsequently, similar users' models can be queried to retrieve the intended collaborative recommendations. As for content-based recommendation, this is even more straightforward since we already have user feature vectors. These can be retrieved by given webpage or web application to compare it to individual item feature vectors to compute the most relevant items. Since we are client-based, we avoid even the cold-start problem in both cases by sharing our private model (or parts of it) with multiple web systems.

### Acknowledgements

This work was partially supported by the Scientific Grant Agency of Slovak Republic, grant No. VG1/0675/11 and by the Slovak Research and Development Agency under the contract No. APVV-0208-10.

# References

- Bilenko, M., Richardson, M.: Predictive client-side profiles for personalized advertising. In Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '11). ACM, New York, NY, USA, pp. 413–421 (2011).
- Gauch, S., Speretta, M., Chandramouli, A., Micarelli, A.: User profiles for personalized information access. In: Brusilovsky, P., Kobsa, A., Nejdl, W. (Eds.): The Adaptive Web: Methods and Strategies of Web Personalization. LNCS 4321. Springer-Verlag, Berlin Heidelberg New York, pp. 54-89 (2007).
- Gerber, S., Fry, M., Kay, J., Kummerfeld, B., Pink, G., Wasinger, R.: PersonisJ: mobile, client-side user modelling. In Proceedings of the 18th international conference on User Modeling, Adaptation, and Personalization (UMAP'10), de Bra, P., Kobsa, A., and Chin, D. (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 111–122 (2010).
- Guo, H., Chen, J., Wu, W., Wang, W.: Personalization as a service: the architecture and a case study. In Proceedings of the first international workshop on Cloud data management (CloudDB '09). ACM, New York, NY, USA, pp. 1–8 (2009).
- Hauger, D.: Using Asynchronous Client-Side User Monitoring to Enhance User Modeling in Adaptive E-Learning Systems. In: Dattolo, A., Tasso, C., Farzan, R., Kleanthous, S., Vallejo, D.B., Vassileva, J. (Eds.): CEUR Workshop Proceedings, Workshop on Adaptation and Personalization for Web 2.0 (UMAP'09). Vol. 485, pp.50-59 (2009).
- Holub, M., Bieliková, M.: An Inquiry into the Utilization of Behavior of Users in Personalized Web. In: Journal of Universal Computer Science, Vol. 17, No. 13, 1830-1853 (2011).
- Kim, K., Sho, S., Lim, J., Kim, S., Lee, S., Lee, J.: A client profile framework for providing adapted contents in ubiquitous environments. In Proceedings of the 5th international conference on Pervasive services (ICPS '08). ACM, New York, NY, USA, pp. 181–184 (2008).
- Koidl, K., Conlan, O., Wei, L., Saxton, A. M.: Non-invasive Browser Based User Modeling Towards Semantically Enhanced Personlization of the Open Web. 2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications. pp. 35–40 (2011).
- Kramár, T., Barla, M., Bieliková, M.: Disambiguating Search by Leveraging a Social Context Based on the Stream of User's Activity. In: User Modeling, Adaptation and Personalization (UMAP'10). LNCS 6075, Springer, pp. 387–392 (2010).

- Kramár, T., Barla, M., Bieliková, M.: Personalizing Search Using Socially Enhanced Interest Model Built from the Stream of User's Activity. In Journal of Web Engineering, Vol.12, No.1&2, pp. 65–92 (2013).
- Matthijs, N., Radlinski, F.: Personalizing web search using long term browsing history. In WSDM '11: Proc. of the fourth ACM international conference on Web search and data mining, ACM Press, pp. 25–34 (2011).
- Mihalcea, R., Tarau, P.: TextRank: Bringing order into texts. In Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing, pp. 404–411 (2004).
- 13. Mika, P.: Ontologies are us: A unified model of social networks and semantics. Journal of Web Semantics Vol. 5, No. 1, pp. 5–15 (2007).
- Ohmura, H., Kitasuka, T., Aritsugi, M.: A Web Browsing Behavior Recording System. In Lecture Notes in Computer Science: Knowledge-Based and Intelligent Information and Engineering Systems, Springer, Berlin, Vol. 6884, pp. 53–62 (2011).
- 15. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. Information Processing and Management, Vol. 24, No. 5, pp. 513–523 (1988).
- 16. Velayathan, G., Yamada, S.: Behavior based web page evaluation. Journal of Web Engineering Vol. 6, No. 3, pp. 222–243 (2007).