This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

Personal Web Revisitation by Context and Content Keywords with Relevance Feedback

Li Jin, Gangli Liu, Chaokun Wang and Ling Feng, *Senior Member, IEEE*

Abstract—Getting back to previously viewed web pages is a common yet uneasy task for users due to the large volume of personally accessed information on the web. This paper leverages human's natural recall process of using episodic and semantic memory cues to facilitate recall, and presents a personal web revisitation technique called *WebPagePrev* through context and content keywords. Underlying techniques for context and content memories' acquisition, storage, decay, and utilization for page re-finding are discussed. A relevance feedback mechanism is also involved to tailor to individual's memory strength and revisitation habits. Our 6-month user study shows that: (1) Compared with the existing web revisitation tool *Memento*, *History List Searching* method, and *Search Engine* method, the proposed *WebPagePrev* delivers the best re-finding quality in finding rate (92.10%), average F1-measure (0.4318) and average rank error (0.3145). (2) Our dynamic management of context and content memories including decay and reinforcement strategy can mimic users' retrieval and recall mechanism. With relevance feedback, the finding rate of *WebPagePrev* increases by 9.82%, average F1-measure increases by 47.09%, and average rank error decreases by 19.44% compared to stable memory management strategy. Among time, location, and activity context factors in *WebPagePrev*, activity is the best recall cue, and context+content based re-finding delivers the best performance, compared to context based re-finding and content based re-finding.

Index Terms—Web revisitation, access context, page content, relevance feedback

1 Introduction

1.1 Motivation

Nowadays, the web is playing a significant role in delivering information to users' fingertips. A web page can be localized by a fixed url, and displays the page content as time-varying snapshot. Among the common web behaviors, web revisitation is to re-find the previously viewed web pages, not only the page url, but also the page snapshot at that access timestamp [1]. A 6-week user study with 23 participants showed nearly 58% of web access belonged to web revisitation [2]. Another 1-year user study involving 114 participants revealed around 40% of queries were re-finding requests [3]. According to [4], on average, every second page loaded was already visited before by the same user, and the ratio of revisited pages among all visits ranges between 20% and 72%.

Psychological studies show that humans rely on both episodic memory and semantic memory to recall information or events from the past. Human's episodic memory receives and stores temporally dated episodes or events, together with their spatial-temporal relations, while human's semantic memory, on the other hand, is a structured record of facts, meanings, concepts and skills that one has acquired from the external world. Semantic information is derived from accumulated episodic memory. Episodic memory can be thought of as a "*map*" that ties together items in semantic memory. The two memories make up the category of human user's declarative memory, and work together in user's information recollecting activities [5]. Thus, when a user's web revisitation behavior happens, s/he tends to utilize episodic memory, interweaved with semantic memory, to recall the previously focused pages. Here, semantic memory accommodates content information of previously focused pages, and episodic memory keeps these pages' access context (e.g., time, location, concurrent activities, etc.) [6], [7].

Inspired by the psychological findings, this paper explores how to leverage our natural recall process of using episodic and semantic memory cues to facilitate personal web revisitation. Considering the differences of users in memorizing previous access context and page content cues, a relevance feedback mechanism is involved to enhance personal web revisitation performance.

1.2 Existing Solutions

In the literature, a number of techniques and tools like bookmarks, history tools, search engines, metadata annotation and exploitation, and contextual recall systems have been developed to support personal web revisitation. The most closely related work of this study is *Memento* system [8], which unifies context and content to aid web revisitation. It defined the context of a web page as other pages in the browsing session that immediately precede or follow the current page, and then extracted topic-phrases from these browsed pages based on the Wikipedia topic list. In comparison, the context information considered in this work includes access time, location and concurrent activities automatically inferred

- *L. Jin, G. Liu and L. Feng are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China. E-mail: l-jin12@mails.tsinghua.edu.cn; gl-liu13@mails.thu.edu.cn; fengling@tsinghua.edu.cn*
- *C. Wang is with the School of Software, Tsinghua University, Beijing 100084, China. Email: chaokun@tsinghua.edu.cn*

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

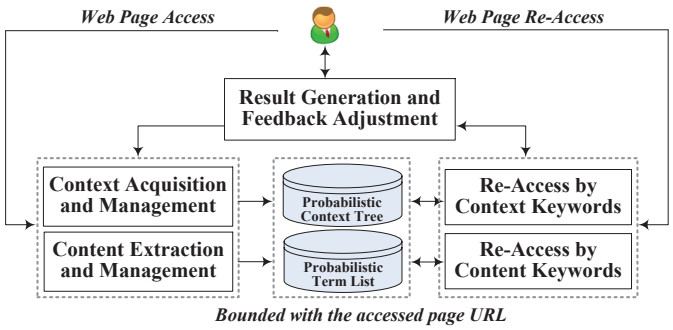


Fig. 1. The personal web revisitation framework

from user's computer programs. Instead of extracting content items from the full web page as done in [8], we extract them from page segments displayed on the screen in the user's view, and assign a probabilistic value to each extracted term based on user's page browsing behaviors (i.e., dwell time and highlighting), as well as page's subject headings and term frequency-inverse document frequency (tf-idf), reflecting user's impression and likeliness of using the keyword as recall content cues. Other closely related work such as [9], [10], [11] enabled users to search for contextually related activities (e.g., time, location, concurrent activities, meetings, music playing, interrupting phone call, or even other files or web sites that were open at the same time), and find a target piece of information (often not semantically related) when that context was on. This body of research emphasizes episodic context cues in page recall. How to grasp possibly impressive semantic content cues from user's page access behaviors, and utilize them to facilitate recall are not discussed. To tailor to individual's web revisitation characteristics, as well as human user's context and content memory degradation nature, this study presents methods to dynamically tune influential parameters in building and maintaining probabilistic context and content memories for recall.

1.3 Our Work

Fig. 1 plots our personal web revisitation framework with relevance feedback. It consists of two main phases.

(1) *Preparation for web revisitation.* When a user accesses a web page, which is of potential to be revisited later by the user (i.e., page access time is over a threshold), the *context acquisition and management* module captures the current access context (i.e., time, location, activities inferred from the currently running computer programs) into a probabilistic context tree. Meanwhile, the *content extraction and management* module performs the unigram-based extraction from the displayed page segments and obtains a list of probabilistic content terms. The probabilities of acquired context instances and extracted content terms reflect how likely the user will refer to them as memory cues to get back to the previously focused page.

(2) *Web revisitation.* Later, when a user requests to get back to a previously focused page through context and/or content keywords, the *re-access by context keywords* module and *re-access by content keywords* module search the probabilistic context tree repository and probabilistic term list repository, respectively. The *result generation and feedback adjustment* module combines the two search results and returns to the user a ranked list of visited page URLs. The relevance feedback mechanism dynamically tunes influential parameters (including memories' decay rates, page reading time threshold, interleaved window size threshold, weight vectors in computing the association and impression scores), which are critical to the construction and management of context and content memories for personal web revisitation.

The main contributions of our paper thus lie in the following three aspects:

- We present a personal web revisitation technique, called *WebPagePrev*, that allows users to get back to their previously focused pages through access context and page content keywords. Underlying techniques for context and content memories' acquisition, storage, and utilization for web page recall are discussed.
- Dynamic tuning strategies to tailor to individual's memorization strength and recall habits based on relevance feedback (e.g., weight preference calculation, decay rate adjustment, etc.) are developed for performance improvement.
- We evaluate the effectiveness of the proposed technique *WebPagePrev*, and report the findings (e.g., the importance of context and content factors) in web revisitation through a 6-month user study with 21 participants.

The rest of the paper is organized as follows. In Section 2, we review closely related work. Then we address the acquisition and management of user's previous access context and content-related information in Section 3, and describe our personal web revisitation approach in Section 4. A relevance feedback mechanism is detailed in Section 5. We evaluate the performance from a 6-month user study in Section 6, and discuss further issues in Section 7. Finally, Section 8 concludes the paper.

2 Related Work

To support personal web revisitation, a number of techniques and tools are developed, including bookmarks, history tools, search engines, metadata annotation and exploitation, and contextual recall systems.

Bookmarks. Apart from $back/forward$ buttons, manually/automatically bookmarking favorite web pages in web browsers enables users to get back to the previously accessed pages. According to user's every visited web page and browsing preferences, [12], [13] built bookmarks automatically and organized them into a recency list [12] or layered structure [13], respectively. Gamez *et al*. [14] further used classifiers to forecast a few of the bookmarks that are more probably to be visited later and showed them in the browser bookmarks personal toolbar, so that the user can access the desired web page through a single mouse click. Bearing similarities

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

to [12] and [14], Kawase *et al*. [15] recommended visited pages relevant to the currently viewed pages, and presented them in a dynamic browser toolbar. Besides, the *SearchBar* tool [16] allowed users to organize their historic search keywords and click pages under different topics. Users can make notes on the topics for easy navigation. With the *Landmark* tool [17], users can also mark a specific part of the page.

History Tools. History tools of web browsers maintain user's accessed URLs chronologically according to visit time (e.g., today, yesterday, last week, etc.), and accessed page titles and contents. Tauscher and Greenberg [2], [18] analyzed 6 weeks of usage data collected from 23 participants when using a commercial browser Mosaic, and discovered that people tend to revisit pages just visited, access only a few pages frequently, browse in very small clusters of related pages, and generate only short sequences of repeated URL paths, which can be used to develop guidelines for the design of history mechanism. *Google Web History*[1] keeps user's search keywords and clicked pages, and categorizes them into image, news, ordinary page, etc. Users can navigate or search the history by page title/content keywords. *Contextual Web History* [19] improved the visual appearance of the web browser history by combining web site thumbnails and content snippets to assist users to easily browse or search their histories by time. *Visual History Tool* [20] encoded four features of a visited web page, which consists of user's page interests measured by dwell time, the frequency and recency of the visit, and navigational associations between pages. List- and graph-based forms are then adopted to provide navigation histories. *xMem* [21] improved history mechanisms by intermixing semantic aspects with the temporal dimension of the accessed pages. It organized the pages into groups and presented a navigational history instead of simply exploiting time-sort history. *SearchPanel* [22] combined web page and process metadata into an interactive representation of the retrieved documents that can be used for sense-making, navigation, and re-finding documents.

Search Engines. Tyler and Teevan [23] studied how search engines are used for re-finding previously found search results. It explored the differences between queries that had substantial/minimal changes between the previous query and the revisit query. Through observing the differences between re-finding behavior occurring within the same session and across multiple sessions, the results showed that cross-session re-finding may be a way to bridge a task between two different sessions. *Re:Search* [24] supported simultaneous finding and re-finding on the web. Past queries were indexed to identify repeated searches, and the most recently viewed results were stored in a result cache. When a user's query was similar to a previous query, *Re:Search* obtained the current results from an existing search engine, and fetched relevant previously viewed results from its cache. The newly available results were then merged with the previously viewed results to create a list that supported intuitive re-finding and contained new information. Adar *et al*. [25] analyzed 5-week web interaction logs from over 612,000 users, and interview studies from 20 participants who installed software to log web page visits for one to two months. They identified twelve different types of revisitation curves corresponding to four groups (i.e., fast, medium, slow, and hybrid revisits), and regarded each of them as a signature of user behavior in accessing a given web page. The analysis of revisitation behaviors for web users in various contexts could empower search engines to better support fast, fresh, and effective finding and re-finding.

Metadata Annotation and Exploitation. *Haystack* [26] stored arbitrary objects of interest to a user, and recorded arbitrary (predefined or user-defined) properties of and relationships between the stored information. It coined a uniform resource identifier (URI) to name anything of interest, including a document, a person, a task, a command/menu operation, or an idea. Once named, the object can be annotated, related to other objects, viewed, and retrieved through arbitrary properties, which served as useful query arguments, as facets for metadata-based browsing, or as relational links to support the associative web browsing. Bearing the similarity to *Haystack*, a SQL-based *MyLifeBits* platform [27] was built for recording, storing, and accessing a personal lifetime archive. It stored content and metadata for a variety of item types, including contacts, documents, email, events, photos, music and video, which were linked together implicitly using "*time*", or explicitly linked with typed links such as a "*person in photo*" link between a contact and a photo, or a "*comment*" link between a voice comment and a document. With linking, the traditional folder (directory) tree was replaced by a more general "*collections*" function using a directed acyclic graph (DAG).

Leveraging Access Context and Page Content. *Stuff I've Seen* [28] built a unified index of information that a person has seen on the computer, including emails, web pages, documents, media files, calendar appointments, etc., and allowed the use of such contextual cues as time, author, thumbnails and previews to search for information. Deng *et al*. [10] allowed users to re-find web pages and local files through previous access context, including time, location, and concurrent activities. It clustered and organized context instances in a context memory, and dynamically degraded the context instances to mimic user's memory decay feature. A query-by-context model for information recall was presented upon the context memory. *YouPivot* [9] leveraged human user's natural method of recall by allowing a user to search through their digital history (e.g., files, URLs, physical location, meetings and events) for the context they do remember. The user can then Pivot, or see everything that was going on while that context was active. Further, *YouPivot* displayed a visualization of the user's activity, providing another method for finding context. *Memento* [8] provid-

1. http://www.google.com/history

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

ed users with descriptive topic-phrases extracted from access context and page content to aid web revisitation. Browsed pages which followed and preceded the accessed web page constitute the page's access context.

3 Preparation for Web Revisitation

This section describes the acquisition and management of user's previous access context and content-related information to prepare for user's web revisitation.

3.1 Context Acquisition and Management Module

3.1.1 Context Acquisition

Three kinds of user's access context, i.e., access time, access location, and concurrent activities, are captured. While access time is determinate, access location can be derived from the IP address of user's computing device. By calling the public IP localization API, we can map the IP address (e.g., "166.111.71.131") to a region (e.g., "Beijing, Tsinghua University"). In order to get a high-precision location, we further build an IP region geocoding database, which could translate a static IP address to a concrete place like "Lab Building, Room 216". If the user's GPS information is available, a public GPS localization application could also help localize the user to a Point of Interest (POI) in the region. User's concurrent activities are inferred from his/her computer programs, running before and after the page access. We continuously monitor the change of user's focused program windows, which can be either a web page, a word file, or a chatting program window, etc., during user's interaction with the computer. Once a user visits a web page longer than a threshold τ_c, computer programs that run interleaving with the current web access program for over τ_c time are taken as the associated computer programs (i.e., context activities).

Let $c[t_s, t_e] = (c.title, c.dur, c.freq)$ denote a computer program within the time window $[t_s, t_e]$, where $c.title$ is a set of words after removing stop words and non-*WordNet* words from the title of the computer program, $c.dur$ is the total running time of the program within the time window $[t_s, t_e]$, and $c.freq$ is the total focus frequency within $[t_s, t_e]$. There are two ways to have a focus program. One is done by the user to manually switch to the program window, and others are the automatically running programs like audio/video players.

Definition 1: Assume a web page access program $w[w_s, w_e] = (w.title, w.dur, w.freq)$ accesses a web page at time w_s, leaves the page at time w_e, and the total visit time of the page (i.e., the total focus duration time of program w) is longer than τ_c. Computer program c is called an **associated computer program (context activity)** within the w's interleaving window $[w_s\text{-}\Delta, w_e\text{+}\Delta]$, denoted as $c[w_s\text{-}\Delta, w_e\text{+}\Delta] = (c.title, c.dur, c.freq)$, if and only if $(c.dur > \tau_c)$.

Parameters τ_c and Δ are subject to individual's memorization and recall characteristics, and will be dynamically tuned based on user's relevance feedback. Initially, τ_c = 90 seconds, Δ = 600 seconds. □

For each associated computer program (context activity) c of a web page w, we bind an association score $cAs(w, c)$ to express how likely the user will use it as a memory cue to re-access the web page later. Intuitively, a program with a longer focus duration and more focus frequency leaves the user a deeper impression than the one with a shorter focus duration and less focus frequency. Similarity/contrast and temporal contiguity also strengthen the association of the program with the web access according to the laws of association during human memory's recollection [29]. Hence, we compute the association score of access context based on the following four features:

1) c's total focus duration $c.dur$.

2) c's total focus frequency $c.freq$.

3) c's temporal distance from a web page w, $D(c, w)$, which is defined as the minimal distance between c's focus time period and w's start/end period $[w_s, w_e]$. Assume a computer program c is focused within $[s_1, e_1], \ldots, [s_k, e_k] \subseteq [w_s\text{-}\Delta, w_e\text{+}\Delta]$, respectively. $D(c, w) = arg\ min_{1 \le i \le k}\ dist([s_i, e_i], [w_s, w_e])$, where

$$dist([s_i, e_i], [w_s, w_e]) = \begin{cases} 0 & \text{if } [s_i, e_i] \text{ overlaps } [w_s, w_e] \\ w_s - e_i & \text{if } [s_i, e_i] \text{ precedes } [w_s, w_e] \\ s_i - w_e & \text{if } [s_i, e_i] \text{ succeeds } [w_s, w_e] \end{cases}$$

4) c and w's title similarity, $Sim(c, w) = \frac{|w.title \cap c.title|}{|w.title|} \in [0, 1]$, where $w.title$ and $c.title$ are a set of title words after removing stop words and words not in *WordNet*.

Definition 2: Let $\mathcal{C}$ be the set of associated computer programs for the web page access program $w[w_s, w_e] = (w.title, w.dur, w.freq)$. Given an associated computer program $c \in \mathcal{C}$ of w, where $c[w_s\text{-}\Delta, w_e\text{+}\Delta] = (c.title, c.dur, c.freq)$, the **association score** of c with w is defined as: $cAs(w, c) = \alpha_1 D_{ur}(w, c) + \alpha_2 F_{req}(w, c) + \alpha_3(1\text{-}D_{ist}(w, c)) + \alpha_4 S_{im}(w, c)$, where 1) $D_{ur}(w, c) = \frac{c.dur}{w.dur + 2\Delta}$; 2) $F_{req}(w, c) = \frac{c.freq}{\sum_{c \in \mathcal{C}}(c.freq)}$; 3) $D_{ist}(w, c) = \frac{D(c,w)}{\Delta}$; 4) $S_{im}(w, c) = \frac{|w.title \cap c.title|}{|w.title|}$; and 5) $\sum_{i=1}^{4} \alpha_i = 1$. Initially, α_i (for $i = 1, 2, 3, 4$) is set to $\frac{1}{4}$, and will be dynamically tuned based on user's relevance feedback. □

Example 3.1: Fig. 2 illustrates three associated computer programs c_1, c_2, c_3 of the web page w, which are *visual studio* program, *adobe reader* program, and *music player* program, respectively, where $w.title$ = "*How to: Retarget a project using DTE*", and $w.dur$ = 265s. $c_1.title$ = "*(visual studio) DTE Command*", $c_1.dur$ = 417+146 = 563s, and $c_1.freq$ = 2. Thus, $D_{ur}(w, c_1) = 563/(265 + 2 * 600) \approx 0.38$, $F_{req}(w, c_1) = 2/4 = 0.5$, $D_{ist}(w, c_1) = 114/600 = 0.19$, $S_{im}(w, c_1) = 1/3 \approx 0.33$. Thus, $cAs(w, c)$ = (0.38+0.5+(1-0.19)+0.33)/4 ≈ 0.51. □

3.1.2 Construction of Probabilistic Context Trees

Access context (i.e., time, location, and concurrent computer programming activities) is organized in a **probabilistic context tree** to support generalized revisit queries due to human user's cognitive understanding and progressive decay during learning and recalling

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

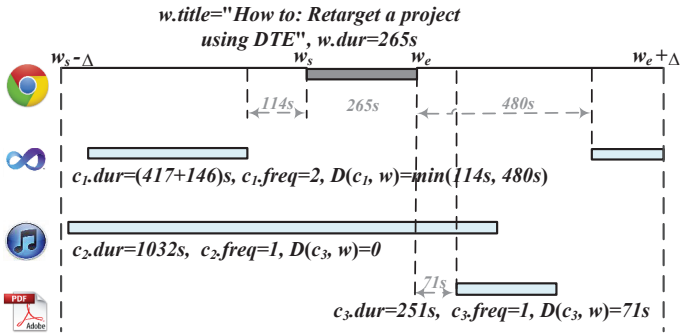


Fig. 2. Three associated computer programs of the web page access program w

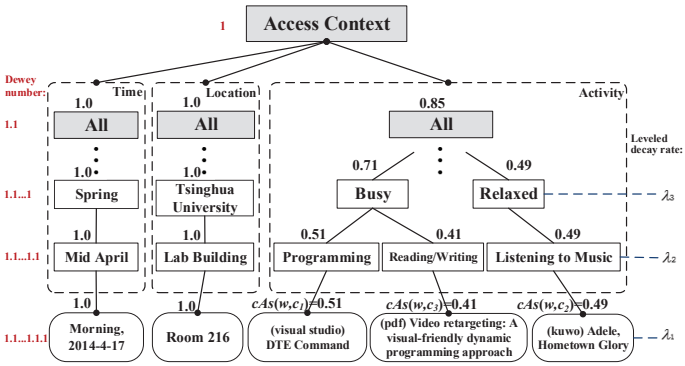


Fig. 3. Probabilistic context tree of the web page access program w

processes [30]. Each leaf node is bounded with a score in [0,1], stating the likelihood that this context node is used as a contextual cue. In the activity subtree, leaf nodes' scores are the association scores defined in Def. 2. As time and location are deterministic, leaf nodes in the time and location subtrees are set to 1.0. With the scores of all the independent leaf nodes available, we can compute the scores of their parent nodes through Jordan formula [31], which is defined as the union of n random events based on the inclusion-exclusion principle. Let $\{child_1, child_2, \ldots, child_n\}$ be a set of child nodes of the same parent node $parent$, the score after union operation is calculated as follows:

$$cAs(w, parent) = (-1)^2 \cdot \sum_{i=1}^{n} cAs(w, child_i) +$$
$$(-1)^3 \cdot \sum_{1 \le i < j \le n} cAs(w, child_i) \cdot cAs(w, child_j) +$$
$$(-1)^4 \cdot \sum_{1 \le i < j < k \le n} cAs(w, child_i) \cdot cAs(w, child_j) \cdot cAs(w, child_k) +$$
$$\cdots + (-1)^{n+1} \cdot cAs(w, child_1) \cdot cAs(w, child_2) \cdot \ldots \cdot cAs(w, child_n)$$

In this way, we can assign scores to all the nodes in the context tree. Fig. 3 gives a leveled probabilistic context tree example for w, whose activity leaf nodes correspond to context activities c_1, c_3, c_2 in Fig. 2. *Busy* as a context node is a general activity status to describe whether the associated computer programs are concerning about working or learning. On the contrary, *relaxed* describes the status of entertainment and leisure. We apply the Dewey encoding scheme to probabilistic context trees based on [32], [33], [34]. Dewey code is a widely used coding scheme for tree structure, where each node is assigned a Dewey number to represent the path from the root to the node. Each component of the path represents the local order of an ancestor node. For example, a tree node n encoded as $n_1.n_2 \ldots n_k$ is a descendant of tree node m encoded as $m_1.m_2 \ldots m_f$ iff $k > f$ and $n_1.n_2 \ldots n_f = m_1.m_2 \ldots m_f$. In our probabilistic context trees, the Dewey number of the root is actually the tree id. For each node in a probabilistic context tree, we build a Trie-based index according to its keywords.

The time complexity of building context trees is $\mathcal{O}(n_c \cdot h + n_c \cdot h \cdot |c|)$, where n_c is the number of captured context instances, $|c|$ is the average instance length, and h is the height of context tree.

3.1.3 Decay and Reinforcement of Probabilistic Context Trees

The obtained probabilistic context trees will evolve dynamically in life cycles to reflect the gradual degradation of human's episodic memorization as well as the context keywords that users will use for recall. That is, for each node in the probabilistic context tree, its association score will progressively decay with time. Psychological study [35] showed that the memorization status of a value v can be expressed as a function of the exponential in the square root of elapsing time (also called age), that is, $v(t) = v(t_0) \cdot e^{-\lambda\sqrt{t-t_0}}$, where $v(t_0)$ is the original value of v at the start time t_0, $v(t)$ is the degraded value of v at time $t \ge t_0$, and λ is the value's decay rate.

For different hierarchical values in the probabilistic context tree, as specific values at lower levels usually degrade faster than general ones at upper levels in human's memory, different decay rates λ_{level_i} $(i = 1, 2, 3, \cdots)$ are assigned in line with the Ebbinghaus Forgetting Curve[2], a graph illustrating how we forget information over time. It was formulated in 1885 by psychologist Hermann Ebbinghaus, who conducted experiments on himself to understand how long the human mind retains information over time. Ebbinghaus took himself as a test subject to examine his own capacity to recollect information by creating a set of 2,300 three-letter, meaningless words to memorize. He studied multiples lists of these words and tested his recall of them at different time intervals over a period of one year. Ebbinghaus discovered that 58.2% was remembered after 20 minutes, 44.2% after 1 hour, 35.8% after 8-9 hours, 33.7% after 1 day, 27.8% after 2 days, and 25.4% after 6 days. Fitting formula $v(t) = v(t_0) \cdot e^{-\lambda\sqrt{t-t_0}}$ with these experimental values, we can calculate and obtain seven different decay rates, and the average decay rate approximates to 0.05. Further similar memorization experiments on meaningful essays and poems demonstrate similar exponential decay patterns in the square root of elapsing time, whose corresponding decay rates exhibit linear relationships with that on words, i.e., $\lambda_{word} : \lambda_{essay} : \lambda_{poem} \approx 12 : 6 : 1$. Based on these findings, we initialize the decay rates at

2. http://www.wisegeek.com/what-is-the-forgetting-curve.htm

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

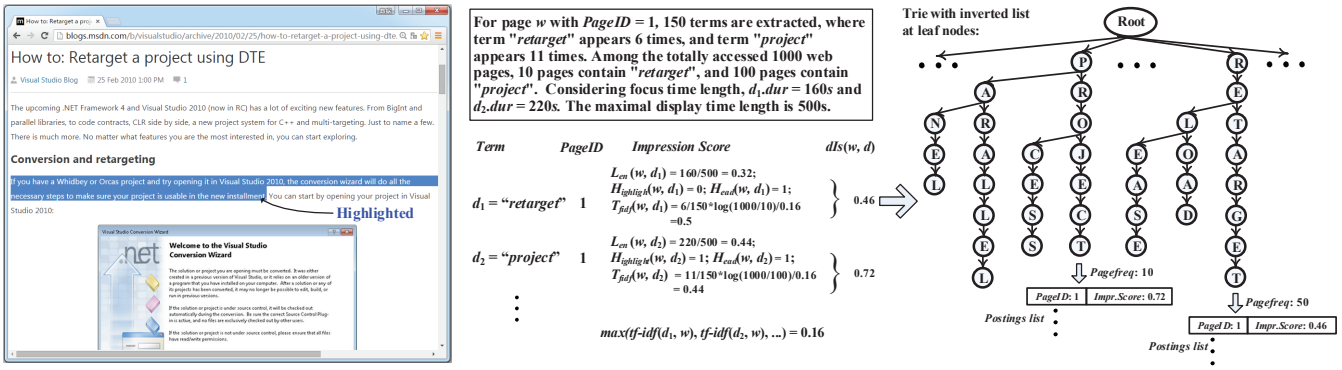


Fig. 4. Probabilistic term list extraction and management of the focused web page w

different hierarchical levels by $\lambda_{level_1} = 0.05$, $\lambda_{level_2} = \frac{1}{2\times1}\lambda_{level_1}$, $\lambda_{level_3} = \frac{1}{3\times2}\lambda_{level_2} = \frac{1}{12}\lambda_{level_1}$. Overall, $\lambda_{level_{i+1}} = \frac{1}{(i+1)\times i}\lambda_{level_i}$ $(i = 1, 2, 3, \cdots)$, whose values will be dynamically adjusted according to user's revisit queries and relevance feedback. The association score of access context value thus degrades as $cAs(w, c, t) = cAs(w, c, t_0) \cdot e^{-\lambda_{level_i}\sqrt{t-t_0}}$.

Apart from memory degradation, the probabilistic context tree may also experience reinforcement due to user's revisit queries. That is, if user types in a context value in the context tree, its possibly degraded association score is reset to the original one, and all its ancestors' scores (if degraded) are also re-computed based on this original value. The decay starting time for its located level is meanwhile reset to the current time.

3.2 Content Extraction and Management Module

Apart from access context, users may also get back to the previous viewed pages through some content keywords. Instead of extracting content terms from the full web page, we only consider the page segments shown on the screen. There are many term weighting schemes in the information retrieval field. The most generic one is to calculate term frequency-inverse document frequency (tf-idf) [36]. For personalized web revisitation, merely counting the occurrence of a term in the presented page segment is not enough. Also, user's web page browsing behaviors (e.g. visitation time length and highlighting or not), as well as page's subject headings, are counted as user's impression and potential interest indicators for later recall. In a similar manner as access context, we bind an impression score to each extracted content term d, showing how likely the user will refer to it for recall based on the four normalized features.

Definition 3: Let d be a content term extracted from the web page segment, shown on the screen of the access program $w[w_s, w_e]$. The **impression score** of d with w is defined as: $dIs(w, d) = \beta_1 L_{en}(w, d) + \beta_2 H_{ighlight}(w, d) + \beta_3 H_{ead}(w, d) + \beta_4 T_{fidf}(w, d)$, where 1) $L_{en}(w, d)$ is the ratio of the time length when the page segment containing d was displayed on the screen versus the maximal display time length of all the viewed page segments; 2) $H_{ightlight}(w, d) = 1$ if user highlights d, and 0 otherwise; 3) $H_{ead}(w, d) = 1$ if d occurs in the page title, and 0 otherwise; 4) $T_{fidf}(w, d)$ is the ratio of term d's tf-idf value $tf-idf(d, w)$ versus the maximal tf-idf value of all the terms extracted from page w; 5) $\sum_{i=1}^{4} \beta_i = 1$. Initially, β_i (for $i = 1, 2, 3, 4$) is set to $\frac{1}{4}$, and will be dynamically tuned based on user's relevance feedback. □

Fig. 4 shows a few content terms extracted from the accessed web page w, where extracted term d's total focus time duration $d.dur$ is more than threshold $\tau_d = 30$ seconds. We organize all the extracted content terms, together with their initial impression scores into a Trie tree based on the longest common prefix. For each term at the leaf node of the Trie tree, an inverted index recording the IDs of all the accessed web pages containing the term is built to facilitate content-based re-search. Like probabilistic context trees in the episodic memory, terms' impression scores in the semantic memory will also progressively decay with time as $dIs(w, d, t) = dIs(w, d, t_0) \cdot e^{-\lambda'\sqrt{t-t_0}}$, where the terms' decay rate λ' is 0.05 in this study, and get reinforced to the original impression scores once the user utilizes them as the content cues for web revisitation.

To gain the speed benefits of indexing at retrieval time, we apply Trie tree to organize the extracted term lists based on the longest common prefix. For each term at Trie tree, inverted index is then built to store a mapping from extracted term lists in advance as shown in Fig. 4. Within a target page collection, we assume that each page has a unique serial number, known as the page identifier ($PageID$). During index construction, the input is term lists for the web pages, we insert the terms into the Trie tree. Meanwhile instances of the same term are grouped together, and the result is split into a dictionary and postings as shown in the right column of Fig. 4. The dictionary records some statistics, such as the number of web pages that contain each term ($Pagefreq$), which also corresponds to the length of each postings list. And postings list stores a list of pairs of impression score $dIs(w, d, t)$ and $PageID$ for a term d.

Here, the time complexity of building content term lists is $\mathcal{O}(n_d \cdot |d|)$, where n_d is the number of extracted terms, and $|d|$ is the average term length.

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

4 Web Revisitation by Context and Content Keywords

Now each user's accessed web page w is bounded with a probabilistic context tree (denoted as $w\#tree$) and a probabilistic term list (denoted as $w\#list$). Let W be the set of user's previously accessed web pages. A revisit query posted by the user at time t is expressed as $W_m = Q(W, Q_c, Q_d, t)$, where Q_c is a set of context keywords, Q_d is a set of content keywords, and answer W_m is a ranked list of matched web pages from W.

The computation of content ranking is straightforward, i.e., $dRank(w\#list|\ Q_d, t) = \prod_{q_d \in Q_d} dIs(w, q_d, t)$, which is the product of matching terms' impression scores against content keywords Q_d. A content ranking example with respect to Q_d = {*"retarget"*, *"project"*} is illustrated in Fig. 5(a). Comparatively, as a context keyword may appear in the titles of multiple tree nodes, the computation of context ranking $cRank(w\#tree|\ Q_c, t)$ is a bit complex. We firstly split a context tree into multiple satisfactory subtrees, so that each subtree contains all the search keywords once and only once. We then compute the ranking of each subtree, and finally merge their ranking results by $cRank(w\#tree|\ Q_c, t) = \sum_{i=1}^{n} cRank(w\#tree_{sub_i}|\ Q_c, t)$. To calculate the ranking score of each subtree, we firstly determine the matched node set $V = \{\nu_1, \nu_2, \cdots\}$. For each context node ν in V, we calculate the matching score by $mAs(Q_c, \nu, t) = \frac{|Q_c \cap \nu.title|}{|\nu.title|} \cdot cAs(w, \nu, t)$. Considering the ancestor node ν_i with a matching child node ν_j ($\nu_j \prec_a \nu_i$), we calculate ν_i and ν_j's matching scores by $mAs(Q_c, \{\nu_i, \nu_j\}, t) = mAs(Q_c, \nu_i \bigcap \nu_j, t) = cAs(Q_c, \nu_j, t)$. The reason is that ancestor node ν_i can be derived from ν_j in context tree, where there exists a dependency relationship. If user remembers the context nodes detailed at lower level, he can directly infer the corresponding nodes at upper level along an upward path. Therefore, the ancestor nodes with matched child nodes can be firstly pruned to keep the rest independent, and $cRank(w\#tree_{sub}|\ Q_c, t)$ is the product of the remaining nodes' scores. Fig. 5(b) gives two smallest subtrees that satisfy Q_c = {*"busy"*, *"programming"*, *"read"*, *"at lab"*}, where "*at*" as a stop word is removed.

In response to a user's web revisitation request, consisting of a set of context keywords Q_c and a set of content keywords Q_d, issued at time t, all the context trees and term lists of user's accessed pages W will be examined, with pages that match Q being extracted as the candidate matched page set W_c. Then the pages with higher matching score will be returned as query result. We call probabilistic context tree $w\#tree$ contains Q_c, if and only if for each context keyword $q_c \in Q_c$, there exists a node c in $w\#tree$ such that $q_c \in c.title$, denoted as $Q_c \subseteq_c w\#tree$. Similarly, we call probabilistic term list $w\#list$ contains Q_d, if and only if for each content keyword $q_d \in Q_d$, there exists a term d in $w\#list$ such that $q_d = d$, denoted as $Q_d \subseteq_d w\#list$.

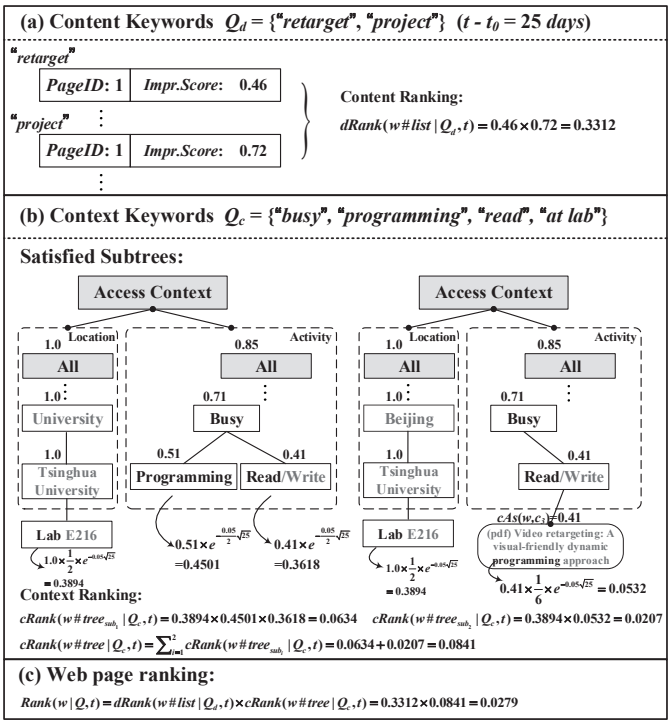


Fig. 5. A web page ranking example for a revisit query containing Q_c and Q_d

Web page w satisfies query request Q, iff $Q_c \subseteq_c w\#tree$ and $Q_d \subseteq_d w\#list$. Its satisfaction rank can be computed based on the context ranking function $cRank(w\#tree|\ Q_c, t)$ and content ranking function $dRank(w\#list|\ Q_d, t)$, that is, $Rank(w|\ Q, t) = cRank(w\#tree|\ Q_c, t) \cdot dRank(w\#list|\ Q_d, t)$. Fig. 5(c) exemplifies the final ranking computation of the matched page w for keyword-based query (Q_c, Q_d).

The detailed procedure is illustrated in Algorithm 1. Through scanning the inverted index, the candidate matched page set W_c can be determined based on matched context trees and matched term lists against a revisit query Q (line 2-4). To compute context ranking, it firstly splits the matched context tree into multiple satisfactory subtrees, then traverses the matched nodes to merge ancestor nodes with child nodes along the same hierarchical path. After calculating the matching score, we can determine each subtree's ranking score $cRank(w\#tree_{sub}|\ Q_c, t)$ and add them up (line 5-15). The content ranking $dRank(w\#list|\ Q_d, t)$ is calculated by multiplying impression score of each content keyword (line 16). Further, the matched web pages' ranking score is the product of context ranking and content ranking (line 17). Finally, the matched pages with lower ranking score are removed, where the parameter δ is initially assigned to 0.2, and dynamically tuned based on relevance feedback. For the rest of pages $W_c{}'$, a quick sorting algorithm is conducted to generate a ranking list W_m (line 18-22). The time complexity of Algorithm 1 is $\mathcal{O}(\sum_{i=1}^{|Q_c|+|Q_d|} |\hbar_i| + |W_c| \cdot \sum_{i=1}^{n} |V_{sub_i}| + |W_c| \cdot |Q_d| + |W_c{}'| \cdot \log(|W_c{}'|))$, where $\hbar$ is the candidate tree or list set by searching the index of context and content keywords.

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

Algorithm 1: Web Page Revisitation Algorithm

Input : a revisit query $Q(W, Q_c, Q_d, t)$
Output: W_m

begin
  $Trees$ = getMatchContextTrees(W, Q_c, t);
  $Lists$ = getMatchTermLists(W, Q_d, t);
  determine candidate matched page set W_c based on $Trees$ and $Lists$;
  foreach $w \in W_c$ **do**
    split $w\#tree$ into n smallest subtrees $w\#tree_{sub_i}$ $(i = 1, \cdots, n)$;
    for $i = 1; i \leq n; i++$ **do**
      determine matched nodes V_{sub_i} of $w\#tree_{sub_i}$;
      foreach $\nu \in V_{sub_i}$ **do**
        if *ν has a matched child node in V_{sub_i}* **then**
          delete ν from V_{sub_i};
        else
          $mAs(Q_c, \nu, t) = \frac{|Q_c \cap \nu.title|}{|\nu.title|} \cdot cAs(w, \nu, t)$;
      $cRank(w\#tree_{sub_i} |\ Q_c, t) = \prod_{\nu \in V_{sub_i}} mAs(Q_c, \nu, t)$;
    $cRank(w\#tree |\ Q_c, t) = \sum_{i=1}^{n} cRank(w\#tree_{sub_i} |\ Q_c, t)$;
    $dRank(w\#list |\ Q_d, t) = \prod_{q_d \in Q_d} dIs(w, q_d, t)$;
    $Rank(w |\ Q, t) = cRank(w\#tree |\ Q_c, t) \cdot dRank(w\#list |\ Q_d, t)$;
  determine the matched page w_τ with highest ranking score;
  foreach $w \in W_c$ **do**
    if $Rank(w |\ Q, t) < \delta \times Rank(w_\tau |\ Q, t)$ **then**
      determine W_c' by deleting w from W_c;
  W_m = Quicksort($W_c', Rank(W_c' |\ Q, t)$);

5 Relevance Feedback

Relevance feedback is an interactive approach that has been shown to work particularly well in classical information retrieval and more recently in web search domain [36]. When a user interacts with *WebPagePrev* during web revisitation phase, s/he can either manually enter some context keywords, or pick up suggested values from contextual hierarchies by clicking the left-side buttons of time, location, and activity bars as shown in Fig. 7. Each contextual hierarchy is dynamically maintained by analyzing the user's clicking behaviors and the statistical frequencies of captured context instances. Frequently accessed context items are top listed in the corresponding contextual hierarchy. User's typos in re-finding requests are automatically corrected by the system based on its indexed content and context keywords.

Fig. 6 shows *top*-4 previously visited web pages under the re-finding context keywords {*"busy"*, *"programming"*, *"at lab"*, *"in April"*}, and content keywords {*"retarget"*, *"project"*}. The user can scroll up and down with the mouse wheel to view all the result pages. If the user double-clicks and dwells on a page by printing, downloading, or reading for a while, we treat the page query relevant. With this feedback information, the web revisitation engine gets to know the system performance, and tune related influential parameters to improve it gradually. Meanwhile, to keep pace with the user's context memorization strength, the engine tunes the leveled-decay rates for probabilistic context memory according to the located levels of typed context keywords.

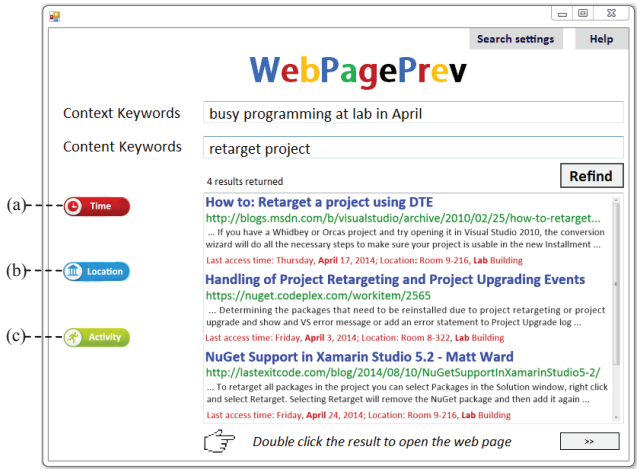


Fig. 6. Web revisitation interface

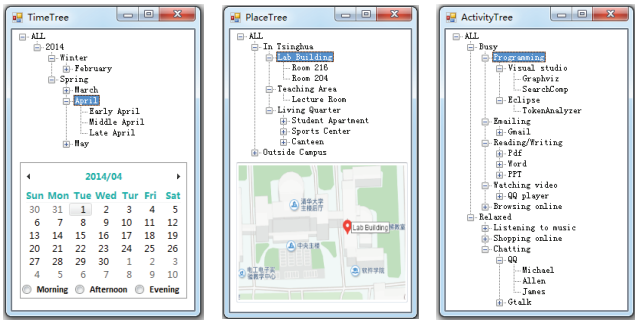


Fig. 7. Suggested values for context keywords input

5.1 Performance Metrics

The web revisitation performance metrics include pages' finding rate, average precision, average recall and average rank error for a set of re-finding requests.

Definition 4: Assume a user's web revisitation request Q returns a ranked list of n result pages, from which the user aims to re-find ψ target pages, and confirms m relevant result pages $\{w_1, \cdots, w_m\}$.
1) The **finding of revisitation** Q is: $Find(Q) = 1$ if the user confirms one or more relevant result pages (i.e., $m > 0$), and 0 otherwise.
2) The **precision of revisitation** Q is: $Precision(Q) = \frac{m}{n}$.
3) The **recall of revisitation** Q is: $Recall(Q) = \frac{m}{\psi}$.
4) The **rank error of revisitation** Q is: $RankError(Q) = \sum_{j=1}^{m} \frac{Pos(Q,w_i)-j}{Pos(Q,w_i)}/m$, where function $Pos(Q, w_i)$ returns the position of the i-th confirmed page w_i in the result page list. □

Definition 5: Let $\mathcal{Q}$ be a set of user's web revisitation requests. The **finding rate**, **average precision**, **average recall** and **average rank error of** $\mathcal{Q}$ are thus defined as follows:
1) $FindRate(\mathcal{Q}) = \frac{\sum_{Q \in \mathcal{Q}} Find(Q)}{|\mathcal{Q}|}$;
2) $AvgPrecision(\mathcal{Q}) = \frac{\sum_{Q \in \mathcal{Q}} Precision(Q)}{|\mathcal{Q}|}$;
3) $AvgRecall(\mathcal{Q}) = \frac{\sum_{Q \in \mathcal{Q}} Recall(Q)}{|\mathcal{Q}|}$;
4) $AvgRankError(\mathcal{Q}) = \frac{\sum_{Q \in \mathcal{Q}} RankError(Q)}{|\mathcal{Q}|}$. □

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

5.2 Influential Parameters to be Tuned

(1) *Parameters used in constructing and managing probabilistic context trees.*

• A weight vector $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ with $\sum_{i=1}^{4} \alpha_i = 1$, used in computing the association score of a context activity c with a web page access program w in Def. 2. It quantifies the importance of c's focus duration, focus frequency, temporal distance from the page access, and title similarity between c and w, respectively. It impacts the ranking position of result pages.

• *Context focus duration threshold* τ_c and *interleaving window size threshold* Δ between c and w in context acquisition. Reducing τ_c and enlarging Δ can capture more associated context activities into the probabilistic context tree, and deliver more result pages.

• Leveled decay rates in probabilistic context trees λ_{level_i} (where $i = 1, 2, 3, 4$ in this study), with which the association score of a context activity c with a web page access w is dynamically computed to match user's memorization strength on context keywords in recall.

(2) *Parameters used in constructing probabilistic content term lists.*

• A weight vector $(\beta_1, \beta_2, \beta_3, \beta_4)$ with $\sum_{i=1}^{4} \beta_i = 1$, used in computing the impression score of a content term d with a web page access program w in Def. 3. It quantifies the importance of d's focus duration, highlighted or not, heading words or not, and tf-idf value, respectively. Similar to $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$, it also impacts the ranking of result pages.

• *Content focus duration threshold* τ_d for content extraction. A lower setting of τ_d enables more content terms to be extracted from the browsed web page and add them to the probabilistic term list.

(3) *Parameter used to adjust the length of result list*

• The parameter δ is used to remove the pages with lower ranking score from result list.

5.3 Tuning Strategies

The parameters tuning is carried out when any of the following three conditions holds:

- Periodically (say, once every two weeks since last tuning);
- When one of the performance metrics drops below a threshold (e.g., $\tau_{FindRate} = 0.8$, $\tau_{Precision} = 0.2$, $\tau_{RankError} = 0.4$) since last tuning;
- When user presses "≫" button at the right-bottom of the screen if s/he is not satisfied with the result and wants more pages.

The tuning consists of 3 steps.

Step 1: Optimize the settings of weight vector $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ and $(\beta_1, \beta_2, \beta_3, \beta_4)$ to ensure user's confirmed relevant result pages are ranked higher than unconfirmed ones, thus decreasing the average rank error. Because the computation of context ranking is independent from that of content ranking. The weight vector α_i and β_i (for $i = 1, 2, 3, 4$) can be respectively optimized in a 4-dimensional weight space to improve context ranking and content ranking for the confirmed relevant result pages. In order to precisely specify the weight coefficients in our linear aggregation function, we propose a weak partial ordering graph model to generate user's preferred ranked list as shown in Fig. 8.

	s_1	s_2	s_3	s_4
w_1	0.8	0.7	0.5	0.2
w_2	0.4	0.6	0.2	0.7
w_3	0.5	0.4	0.7	0.3
w_4	0.2	0.3	0.8	0.4
w_5	0.5	0.3	0.1	0.1

Weak partial ordering graph:

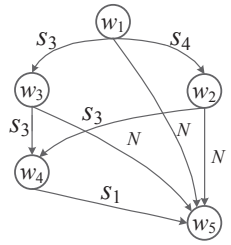


Fig. 8. A weak partial ordering graph example

Definition 6: Given a set of web pages $W = \{w_1, w_2, \cdots, w_n\}$ and corresponding normalized feature values $S = \{s_1, s_2, \cdots, s_m\}$, a weak partial ordering graph is a directly graph $G(V, E)$, where E is a set of edges defining the relative ordering between pages, and V is a set of vertices depicting the $PageID$. There exists an edge from w_i to w_j when one of the following two conditions holds:
1) $\forall s \in S, w_i.s \geq w_j.s$;
2) $\forall s \in S - \{s_k\}, w_i.s \geq w_j.s$, where $s_k \in S$ and $w_i.s_k < w_j.s_k$; □

Example 5.1: Fig. 8 illustrates five web pages with four normalized feature values. For pages w_1 and w_3, the edge e_{13} satisfying condition 2 is labeled by s_3, where $w_1.s_3 < w_3.s_3$. For pages w_2 and w_5, the edge e_{25} satisfying condition 1 is labeled by N.

Taking context ranking as an example illustrated in Fig. 8, we detail the adjustment of weight vector $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ into two cases: 1) When user confirms one result page as relevant, the adjustment is based on the label of edges incident to the confirmed page's node. For example, when user confirms w_5 as a relevant page, we can clearly determine enlarging α_1 can change the relative order between w_4 and w_5 by the label s_1 of e_{45}; 2) When user confirms two or more pages as relevant, the confirmed pages' nodes are firstly clustered as a set. By neglecting the inner edges between graph nodes of confirmed pages, the adjustment is based on the label of edges incident to the cluster set. For example, when user confirms w_3 and w_4 as relevant pages, the inner edge e_{34} is neglected for the clustered pages $\{w_3, w_4\}$. And the labels of incident edges are ranked based on their frequencies. Through enlarging $s3$ and $s4$, the $RankError(Q)$ can be reduced.

During the adjustment of weight vector, the linear constraint $\sum_{i=1}^{4} \alpha_i = 1$ should be satisfied. The objective of weight adjustment is to minimize $RankError(Q)$. For case 1, after increasing α_1 by $\triangle\alpha_1$, other weights should be decreased to satisfy the constraint $\sum_{i=1}^{4} \alpha_i = 1$. We adopt $\Delta\alpha_k = \frac{w_4.s_k - w_5.s_k}{\sum_{i=2}^{4} w_4.s_i - w_5.s_i} \cdot \Delta\alpha_1$ (for $k = 2, 3, 4$) to allocate $\triangle\alpha_1$ to $(\alpha_2, \alpha_3, \alpha_4)$. The larger $w_4.s_k - w_5.s_k$ is, the more corresponding weight decreases. Through traversing the label of other edges $\{e_{15}, e_{35}, e_{25}\}$ con-

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

cerning w_5, the upper bound of w_5's ranking position is 4. For case 2, we firstly enlarge α_3 and α_4 sequentially to reduce $RankError(Q)$. When increasing α_3 by $\triangle\alpha_3$, (α_1, α_2) should be decreased and α_4 remains the same. The decreased amount for (α_1, α_2) is calculated by $\Delta\alpha_k = \frac{(w_1.s_k - w_3.s_k)+(w_2.s_k - w_4.s_k)}{\sum_{i=1}^{2}[(w_1.s_i - w_3.s_i)+(w_2.s_i - w_4.s_i)]} \cdot \Delta\alpha_3$ (for $k = 1, 2$). When context ranking satisfies condition $cRank(w_4\#tree|\ Q_c, t) > cRank(w_2\#tree|\ Q_c, t)$, we begin to increase α_4. For the adjusted weight values that minimize $RankError(Q)$, we add them into the candidate set ς_α. Based on the following user typed queries and confirmed operations, a suitable weight vector value can be learned from ς_α. Considering a set of re-finding requests $\mathcal{Q}$, the adjustment of weight vector is conducted on each graph to minimize $AvgRankError(\mathcal{Q})$.

The weight adjustment for $(\beta_1, \beta_2, \beta_3, \beta_4)$ can be also conducted in a similar manner.

Step 2: Decrease the context and content focus duration thresholds τ_c and τ_d by half, and doubly increase the interleaving window size threshold Δ in context acquisition, if the finding rate is low and less than $\tau_{FindRate}$. In this way, more pages could be returned in the result list. The other way around, to improve the average precision, context and content focus duration threshold τ_c and τ_d are to be increased, and the interleaving window size Δ is to be shortened, aiming to drop out more irrelevant web pages from the results. To this end, we examine all user's confirmed relevant pages since last tuning, and take the smallest τ_c and τ_d, as well as the largest Δ, of their associated context activities and content terms as the new values of τ_c, τ_d, and Δ. Here, the parameters tuning for the improvement of $FindRate(\mathcal{Q})$ and $AvgPrecision(\mathcal{Q})$ is in two different directions. If the former is less than the later, our tuning is $FindRate$-oriented, and otherwise $AvgPrecision$-oriented.

Step 3: Adjust leveled decay rates of context trees λ_{level_i} (for $i = 1, 2, 3, 4$) according to the hierarchical levels of context keywords that user exploited in his/her revisit queries, since the later reflects the context memorization status. To match the user's context memory degradation, we count the number of typed context keywords at each level as ξ_i, and approximate the user's context memorization amount at $level_i$ by $cm_i = \frac{\xi_i + \xi_{i-1} + \cdots + \xi_1}{\sum_{j=1}^{4} \xi_j}$. It is assumed that if a user remembers a context value at a lower level (e.g., "Lab Building"), there is no doubt that s/he also remembers its upper-leveled value (e.g., "Tsinghua University"). Let $cm_i = e^{\lambda_{level_i} \cdot \sqrt{t_0 - t}}$. The context memory's decay rate at $level_i$ is adjusted to $\lambda_{level_i} = \frac{log\ (cm_i)}{\sqrt{t - t_0}}$, where t_0 is the initial time that the context tree is constructed, and t is the current tuning time.

Step 4: Decrease δ when confirmed pages with lower ranking score are still removed from result list after *Step 1*. Meanwhile, increase δ gradually to improve system's $AvgPrecision(\mathcal{Q})$ when $FindRate(\mathcal{Q})$ keeps stable.

Everytime when the tuning strategy is conducted, the influential parameters are adjusted to gradually approximate the inherent browsing behaviors and recall habits. Repeating such parameter adjustment, the strategy can guarantee that there is a better solution to improve the average re-finding performance after a period of time.

The time complexity of relevance feedback mechanism is $\mathcal{O}(N_q \cdot \lambda \cdot \log n + N_q \cdot \sum_{i=1}^{L} |\mathcal{S}_i|)$, where N_q is the number of revisit queries, λ is the number of candidate paths, n is the number of graph nodes for $G(V, E)$, L is the number of hierarchical levels, and $\mathcal{S}_i$ is the context node set at the ith level.

6 Evaluation

In this section, we firstly examine the effectiveness of the proposed web revisitation technique through a 6-month user study with 21 participants, then evaluate the scalability of our approach on a large synthetic dataset. The experiment on synthetic data is running on a PC with 3.10 GHz Intel i5-3450 CPU, and 10 GB memory.

6.1 User Study

6.1.1 Experimental Setup

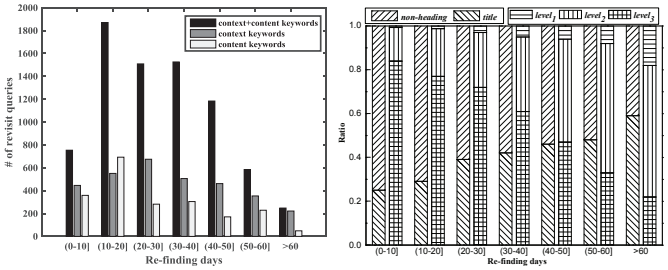
We conducted a 6-month user study from Feb. 16, 2014 to Aug. 16, 2014 to investigate the re-finding performance of four different methods, i.e., *WebPagePrev*, *Memento*, *History List Searching*, and *Search Engine*. 14 students, 2 teachers, 3 office staffs, and 2 engineers (totally 21 users, 13 male and 8 female, aged between 18 and 35) from Tsinghua University, Beijing were invited to participate in the user study. We firstly installed *WebPagePrev* and *Memento* on each user's laptop. A javascript-based cross-platform plug-in was deployed on each user's *Chrome* web browser to obtain his/her page visit behavior, and a context monitor developed by C++ was deployed in the background to capture running computer programs by calling the windows API functions.

We then provided the following instructions to the users before starting the experiment: 1) Each user is suggested to execute at least one re-finding task per day. 2) For each re-finding task, four different methods are to be invoked. The execution sequence is random. 3) When using *WebPagePrev* to do re-finding, context keywords (from time, location, and activity context hierarchies) and content keywords (from page's title and focused body text) could be input. With *Memento*, content keywords (i.e., topic-phrases extracted from the page) and context keywords (i.e., topic-phrases extracted from the preceding and following pages) could be input. The users could leave either context or content keywords empty if they do not want to. With *History List Searching*, content keywords from page's title could be input. With *Search Engine*, content keywords from page's title, body text, and other descriptive information could be input. 4) The users should connect the external GPS module with the laptops when they are outside the campus. 5) At the end of a re-finding task, the users should record their search keywords, the number of target pages they looked for, the ranking position(s) of returned target page(s), and the length of result page list.

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

During our 6-month user study, the total number of focused pages is 111042, and about 5288 per user. Each user raises about 618 revisit queries. On average, with *WebPagePrev*, the users use 4.37 context keywords and 2.26 content keywords to execute a revisit query. With *Memento*, 1.82 context keywords and 2.07 content keywords are used. Besides, 1.41 content keywords and 3.84 content keywords are used with *History List Searching* and *Search Engine*, respectively.



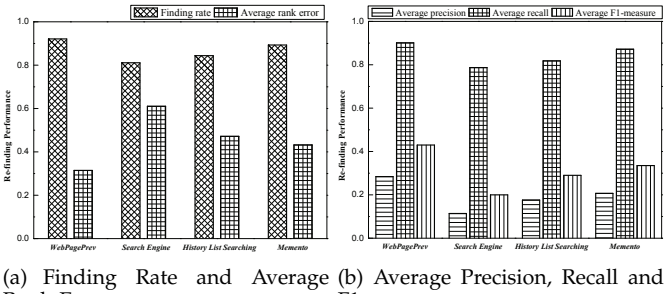
(a) # of different query types (b) Ratio of query keywords

Fig. 9. Statistics of revisit queries in different re-finding time intervals for *WebPagePrev*

From Fig. 9(a), we can see the number of three query types in different re-finding time intervals, where re-finding days day_R is the elapsed days since the page was accessed. The results show users are more dependent on episodic memory cues when searching for the target page with a longer day_R. For example, the number of queries using context keywords is more than that of queries using content keywords when day_R>20, especially 4.53 times when day_R>60. Fig. 9(b) shows the ratio of query keywords belonging to different context hierarchical levels and page segments in different re-finding time intervals. Owning to the decay of episodic memory, as time elapses, more general context keywords from level 2, 3 are typed by the users, and the ratio is over 52.74% when day_R>40. Besides using title terms, they also tend to recall more content keywords from the non-heading page segments, over 46.22% when day_R>40.

6.1.2 Experimental Results

(1) **Performance Comparison with Existing Approaches.** We compare the performance of our personal web revisitation approach with three conventional methods. From Fig. 12, *WebPagePrev* delivers the best average F1-measure, about 2.15 times, 1.51 times and 1.29 times than that of *Search Engine* method, *History List Searching* method and *Memento*. For the precision metric of *Search Engine* method, parameter n corresponds to the number of browsed pages in user's visual field before getting the desired targets. The finding rate of *WebPagePrev* is 92.10% compared to *Search Engine* method 81.11%, *History List Searching* method 84.40% and *Memento* 89.31%. Further, the average rank error of *WebPagePrev* is 0.3145, compared to *Search engine* method 0.6105, *History List Searching* method 0.4717 and *Memento* 0.4322. The reason includes several aspects. *History List Searching* method mainly utilizes the terms from page title, leaving out other useful content cues. While the query results and their rankings are frequently updated within the search engine, participants sometimes felt difficult to re-locate the target pages. *Memento* does not make full use of activity context and collects more redundant content terms, which causes irrelevant query results returned. We further perform *t-test* and all the *p-values* are < 0.01, which indicate that the improvement of *WebPagePrev* over the comparison methods are statistically significant.



(a) Finding Rate and Average Rank Error (b) Average Precision, Recall and F1-measure

Fig. 12. Performance comparison with existing approaches

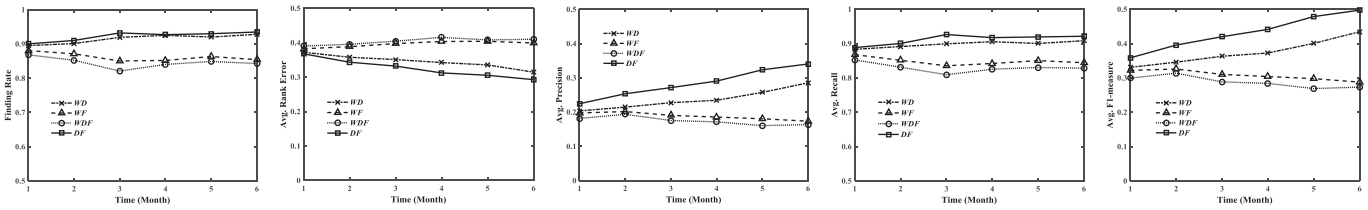
(2) **Effectiveness of Memories Decay and Relevance Feedback.** Through removing decay and relevance feedback mechanism from *WebPagePrev*, we evaluate the effectiveness by comparing four different cases: 1) Without decay (*WD*); 2) Without relevance feedback (*WF*); 3) Without decay and relevance feedback (*WDF*); 4) With decay and relevance feedback (*DF*). From Fig. 10, *DF*'s finding rate increases by 0.88%, average F1-measure increases by 15.27%, and average rank error decreases by 4.71% than *WD*. In comparison with stable memory management strategy, *DF*'s finding rate increases by 9.82%, average F1-measure increases by 47.09%, and average rank error decreases by 19.44% than *WDF*. Considering relevance feedback, *DF*'s finding rate increases by 7.16%, average F1-measure increases by 39.22%, and average rank error decreases by 16.14% than *WF*. Here, $AvgRecall(\mathcal{Q})$ and $FindRate(\mathcal{Q})$ are quite close. This is because when there is only one wanted target, $AvgRecall(\mathcal{Q})$=$FindRate(\mathcal{Q})$. In the study, around 83.37% of re-finding queries look for one target page.

(3) **Effectiveness of Weight Adjustment in Relevance Feedback.** We evaluate the effectiveness of *Weak Partial Ordering Graph* (*WPOG*) by comparing the performance with *Possible Orderings Tree* (*POT*) as a baseline [37]. From Fig. 11, *WPOG*'s finding rate increases by 3.19%, average F1-measure increases by 11.38%, and average rank error decreases by 8.23% than *POT*. The main reason is that *POT* makes an assumption for the weight coefficients, which should satisfy a uniform distribution. However, this assumption does not always hold for different users. During the adjustment of weight vectors, there are a set of candidate solutions to minimize $AvgRankError(\mathcal{Q})$. *WPOG* can determine better weight coefficients considering user's preference instead of the mean value.

(4) **Contribution Analysis of Context and Content Factors.** To examine the importance of different factors in *WebPagePrev*, we divide revisit queries into three types,

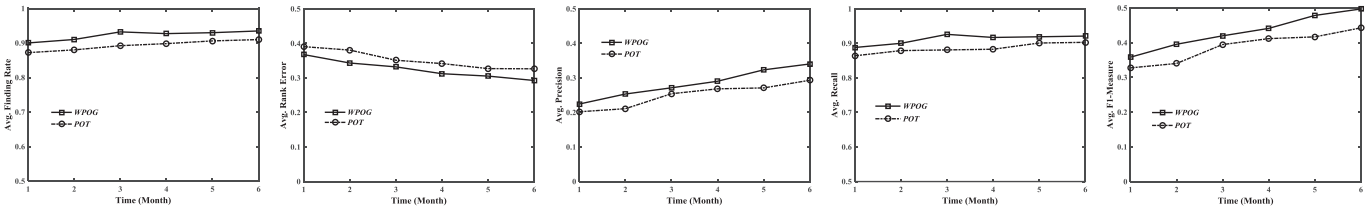
1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering



(a) Finding Rate (b) Average Rank Error (c) Average Precision (d) Average Recall (e) Average F1-measure

Fig. 10. Evaluating the effectiveness of memories decay and relevance feedback



(a) Finding Rate (b) Average Rank Error (c) Average Precision (d) Average Recall (e) Average F1-measure

Fig. 11. Evaluating the effectiveness of weight adjustment method in relevance feedback

i.e., 1) querying based on content keywords only, 2) context keywords only, 3) context and content keywords. In the study, around 16.09% of queries belong to query type 1, 24.77% belong to query type 2, and 59.14% belong to query type 3. Table 1 shows that queries using context and content keywords perform the best in finding rate (93.88%), average rank error (0.2849) and average F1-measure (0.4733). Queries using context keywords perform better than queries using content keywords by increasing 3.04% in finding rate, decreasing 19.27% in average rank error, and increasing 29.57% in average F1-measure. The main reason for these performance differences could lie in the number of query keywords used. Content based queries use around 2.71 keywords on average, context based queries use around 4.83 keywords, and context+content based queries use around 6.34 keywords (4.18 context keywords and 2.16 content keywords). We observe that the users tend to enter more than one activity element in the contextual hierarchy like "busy" followed by "programming".

TABLE 1

Performance comparison between context and content factors in *WebPagePrev*

Query Keywords	Finding Rate	Average Rank Error	Average Precision	Average Recall	Average F1-measure
Content	0.8745	0.4033	0.1903	0.8577	0.3115
Context	0.9011	0.3256	0.2619	0.8830	0.4036
Context+ Content	0.9388	0.2849	0.3185	0.9212	0.4733

Furthermore, we investigate the behaviors of time, location, and activity contextual elements in *WebPagePrev*. For query type 2 and 3, about 9.68% of queries use time, 6.32% use location, 16.94% use activity keyword(s), 8.27% use time plus location, 25.74% use time plus activity, 14.72% use location plus activity, and 18.33% use all the three. From Table 2, we find that activity context is the best recall cue, followed by time and location context. The reason is due to the smallest search space of activity context compared to that of time and location context. More candidate accessed pages are associated with a time or location cue. As the activity context is inferred from user's computer programs, it binds with page access more closely, leading to the best performance.

TABLE 2

Performance comparison of different context factors in *WebPagePrev*

Context Factor	Finding Rate	Average Rank Error	Average Precision	Average Recall	Average F1-measure
Time	0.8873	0.3474	0.2574	0.8681	0.3971
Location	0.8716	0.3907	0.2433	0.8533	0.3786
Activity	0.9192	0.3221	0.2950	0.8922	0.4434
Time+Loc.	0.9066	0.3423	0.2805	0.8784	0.4252
Time+Act.	0.9491	0.2630	0.3149	0.9344	0.4711
Loc.+Act.	0.9379	0.2857	0.3013	0.9207	0.4540
Time+Loc. +Act.	0.9443	0.2489	0.3414	0.9309	0.4996

(5) **User Satisfaction Analysis**. After a 6-month user study, the participants completed a questionnaire to express their attitudes towards *WebPagePrev* as shown in Table 3. These questions were answered through the *Likert-type scale* approach, where 1 represents strongly disagree and 5 represents strongly agree. From Table 3, we can see that participants were basically satisfied with *WebPagePrev*'s re-finding results, where the average rating score is 4.42. Meanwhile, some participants suggested the user interaction module needs to be improved.

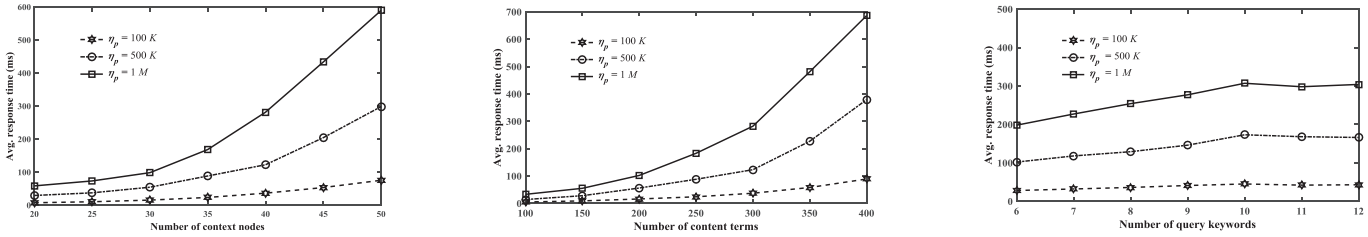
TABLE 3

Questionnaire Result

Question	Rating
I can re-find previously viewed web pages easily with *WebPagePrev*.	4.13
I am satisfied with *WebPagePrev*'s re-finding results.	4.42
The web revisitation interface is friendly.	3.68
Contextual information (time, place, and activity) constitutes useful cues for web revisitation.	4.35
The provided context hierarchical trees are helpful when I could only remember the past access context vaguely.	3.87
WebPagePrev can replace the commonly used browser tools (e.g., history list, search engine) for web revisitation.	4.17
I prefer to continue to update the software, and use it for a long time.	4.21

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering



(a) η_t=300, η_k=10 (b) η_n=40, η_k=10 (c) η_t=300, η_n=40

Fig. 13. Average revisitation response time on synthetic data

6.2 Experiment on Synthetic Data

Assume a user has accessed and focused $total\_page\_num$ ($\eta_p = 100K, 200K, \cdots, 1M$) web pages. Associated with each web page, there is a probabilistic context tree and a probabilistic content term list, over which user's re-finding requests are executed. Each context tree has 4 hierarchical levels with $context\_node\_num$ ($\eta_n = 20, 25, \cdots, 50$) tree nodes in total. The node number at level 1, 2 and 3 occupies 70%, 20%, and 10% of the total node number. Except for root node representing context ALL, nodes at the same level have the same fan-out value. For time context, we generate a set of time stamps during 12 months. For location context, we select a set of regions and extract a set of POIs (point of interests) from a city map (Beijing). The POIs are randomly assigned to each context tree, where the leaf nodes' overlap ratio is 35%. For activity context, we create a dataset by capturing 30115 computer programs from working environment of 7 participants. The association score of each child node at the lowest level is a random value from 0 to 1. Each probabilistic content term list accommodates maximally $content\_term\_num$ ($\eta_t = 100, 150, \cdots, 400$) unique stemmed terms extracted from pages crawled from common internet web sites (e.g., shopping site, blog site, Quora, etc). The impression score of each term is randomly assigned in the range of $(0, 1)$. Assume a user raises $\frac{\eta_p}{100}$ revisit queries, each query contains $query\_keyword\_num$ ($\eta_k = 6, 7, \cdots, 12$) keywords, where about half of them are context keywords, and the other half are content keywords. Both are randomly taken from the context tree and content term list bounded with a certain web page.

From the result presented in Fig. 13(a),(b), we can find that the average response time increases accordingly with the increase of η_n and η_t. The reason is obvious, as more context nodes and content terms linked with every page access need to be examined and matched with the user's query keywords. More pages being accessed, more context trees and term lists need to be checked, thus more time to process the revisit queries. On the other hand, the response time does not increases sharply with η_k, as illustrated in Fig. 13(c). For example, when the user's query contains 6 keywords, the response time is 28 ms (when $\eta_p = 100K$), 102 ms (when $\eta_p = 500K$), and 198 ms (when $\eta_p = 1M$). Further, it increases to 45 ms, 173 ms and 307 ms, respectively, when $\eta_k = 10$. Then the response time cost decreases a little when η_k continues goes up. This might be because more query keywords quickly filter mismatched context trees, thus reducing the search space of candidate term lists.

7 Discussion

When a user does re-finding, s/he usually has certain purposes in mind, like preparing a project proposal, writing codes, etc. *WebPagePrev* strives to support users to re-find what they accessed through previous access time, location, concurrent activities, and content keywords. Beyond that, more user-centric context factors (e.g., access purpose, expertise, background, interest, etc.), as well as social context factors (e.g., external events, surrounding people, etc.), could be inferred from user's profile, agenda, and external service providers, and bounded with the accessed pages. In this way, not only the user him/herself could benefit from such rich contextual cues during re-finding process, but also other users with similar access purpose and background could share the more directed page access. This is in line with the spirit of social search [38], [39], [40], which advocates two paradigms (namely, *library* paradigm and *village* paradigm) in information retrieval. According to [39], in a library, people use keywords to search documents, and the trust is based on authority, while in a village, people use natural language to ask questions, answers are generated in real-time by anyone with the expertise in the community, and trust is based on intimacy.

In social search, a lot of data about the people is used, bringing in privacy protection issues. Life-cycle management of people's information with degradation policies from high to low precision, as done with the context memory mechanism in this study, could be exploited. We leave this issue to our further study.

8 Conclusion

Drawing on the characteristics of human brain memory in organizing and exploiting episodic events and semantic words in information recall, this paper presents a personal web revisitation technique based on context and content keywords. Context instances and page content are respectively organized as probabilistic context trees and probabilistic term lists, which dynamically evolve by degradation and reinforcement with relevance feedback. Our experimental results demonstrate the effectiveness and applicability of the proposed technique. Our future work includes 1) prediction of users' revisitation, 2)

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2017.2672747, IEEE Transactions on Knowledge and Data Engineering

extending the technique to support users' ambiguous re-finding requests, and 3) incorporating social context factors in information re-finding.

REFERENCES

[1] A. Cockburn, S. Greenberg, S. Jones, B. Mckenzie, and M. Moyle. Improving web page revisitation: analysis, design and evaluation. *IT & Society*, 1(3):159–183, 2003.

[2] L. Tauscher and S. Greenberg. How people revisit web pages: empirical findings and implications for the design of history systems. *International Journal of Human Computer Studies*, 47(1):97–137, 1997.

[3] J. Teevan, E. Adar, R. Jones, and M. Potts. Information re-retrieval: repeat queries in yahoo's logs. In *SIGIR*, pages 151–158, 2007.

[4] M. Mayer. Web history tools and revisitation support: a survey of existing approaches and directions. *Foundations and Trends in HCI*, 2(3):173–278, 2009.

[5] L. C. Wiggs, J. Weisberg, and A. Martin. Neural correlates of semantic and episodic memory retrieval. *Neuropsychologia*, pages 103–118, 1999.

[6] M. Lamming and M. Flynn. "forget-me-not": intimate computing in support of human memory. In *FRIEND21 Intl. Symposium on Next Generation Human Interface*, 1994.

[7] E. Tulving. What is episodic memory? *Current Directions in Psychological Science*, 2(3):67–70, 1993.

[8] C. E. Kulkarni, S. Raju, and R. Udupa. Memento: unifying content and context to aid webpage re-visitation. In *UIST*, pages 435–436, 2010.

[9] J. Hailpern, N. Jitkoff, A. Warr, K. Karahalios, R. Sesek, and N. Shkrob. Youpivot: improving recall with contextual search. In *CHI*, pages 1521–1530, 2011.

[10] T. Deng, L. Zhao, H. Wang, Q. Liu, and L. Feng. Refinder: a context-based information re-finding system. *IEEE TKDE*, 25(9):2119–2132, 2013.

[11] T. Deng, L. Zhao, and L. Feng. Enhancing web revisitation by contextual keywords. In *ICWE*, pages 323–337, 2013.

[12] H. Takano and T. Winograd. Dynamic bookmarks for the WWW. In *HYPERTEXT*, pages 297–298, 1998.

[13] S. Kaasten and S. Greenberg. Integrating back, history and bookmarks in web browsers. In *HCI*, pages 379–380, 2001.

[14] J. A. Gamez, J. L. Mateo, and J. M. Puerta. Improving revisitation browsers capability by using a dynamic bookmarks personal toolbar. In *WISE*, pages 643–652, 2007.

[15] R. Kawase, G. Papadakis, E. Herder, and W. Nejdl. Beyond the usual suspects: context-aware revisitation support. In *HT*, pages 27–36, 2011.

[16] D. Morris, M. R. Morris, and G. Venolia. Searchbar: a search-centric web history for task resumption and information re-finding. In *CHI*, pages 1207–1216, 2008.

[17] B. MacKay, M. Kellar, and C. Watters. An evaluation of landmarks for re-finding information on the web. In *CHI*, pages 1609–1612, 2005.

[18] L. Tauscher and S. Greenberg. Revisitation patterns in world wide web navigation. In *CHI*, pages 399–406, 1997.

[19] S. S. Won, J. Jin, and J. I. Hong. Contextual web history: using visual and contextual cues to improve web browser history. In *CHI*, pages 1457–1466, 2009.

[20] T. V. Do and R. A. Ruddle. The design of a visual history tool to help users refind information within a website. In *ECIR*, pages 459–462, 2012.

[21] F. Rizzo, F. Daniel, M. Matera, S. Albertario, and A. Nibioli. Evaluating the semantic memory of web interactions in the xmem project. In *AVI*, pages 185–192, 2006.

[22] P. Qvarfordt, S. Tretter, G. Golovchinsky, and T. Dunnigan. Search-panel: framing complex search needs. In *SIGIR*, pages 495–504, 2014.

[23] S. Tyler and J. Teevan. Large scale query log analysis of re-finding. In *WSDM*, pages 191–200, 2010.

[24] J. Teevan. The re:search engine: simultaneous support for finding and re-finding. In *UIST*, pages 23–32, 2007.

[25] E. Adar, J. Teevan, and S. T. Dumais. Large scale analysis of web revisitation patterns. In *CHI*, pages 1197–1206, 2008.

[26] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A customizable general-purpose information management tool for end users of semistructured data. In *CIDR*, pages 13–26, 2003.

[27] J. Gemmell, G. Bell, and R. Lueder. Mylifebits: a personal database for everything. *Communications of the ACM*, 49(1):88–95, 2006.

[28] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff i've seen: a system for personal information retrieval and re-use. In *SIGIR*, 2003.

[29] R. Sorabji. *Aristotle on memory*. University of Chicago Press, 2rd edition, 2006.

[30] H. C. Ellis and R. R. Hunt. *Fundamentals of human memory and cognition*. William C. Brown, 3rd edition, 1983.

[31] R. Durrett. *Probability: theory and examples*. Cambridge University Press, 4rd edition, 2010.

[32] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: ranked keyword search over xml documents. In *SIGMOD*, pages 16–27, 2003.

[33] J. Li, C. Liu, R. Zhou, and W. Wang. Top-k keyword search over probabilistic xml data. In *ICDE*, pages 673–684, 2011.

[34] H. Georgiadis and V. Vassalos. Improving the efficiency of xpath execution on relational systems. In *EDBT*, pages 570–587, 2006.

[35] D. C. Rubin and A. E. Wenzel. One hundred years of forgetting: a quantitative description of retention. *Psychological Review*, 103(4):734–760, 1996.

[36] I. Ruthven and M. Lalmas. A survey on the use of relevance feedback for information access systems. *Knowledge Engineering Review*, 18(2):95–145, 2003.

[37] M. A. Soliman, I. F. Ilyas, D. Martinenghi, and M. Tagliasacchi. Ranking with uncertain scoring functions: semantics and sensitivity measures. In *SIGMOD*, pages 805–816, 2011.

[38] M. V. Vieira, B. M. Fonseca, R. Damazio, P. B. Golgher, D. d. C. Reis, and B. Ribeiro-Neto. Efficient search ranking in social networks. In *CIKM*, pages 563–572, 2007.

[39] D. Horowitz and S. D. Kamvar. The anatomy of a large-scale social search engine. In *WWW*, pages 431–440, 2010.

[40] D. Carmel, N. Zwerdling, I. Guy, S. Ofek-Koifman, N. Har'el, I. Ronen, E. Uziel, S. Yogev, and S. Chernov. Personalized social search based on the user's social network. In *CIKM*, pages 1227–1236, 2009.

Li Jin is a Ph.D. candidate in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include context-aware data management and context-based information re-finding.

Gangli Liu is a Ph.D. candidate in the Department of Computer Science and Technology, Tsinghua University, Beijing, China. His current research interests include context-aware data management and context-based information re-finding.

Chaokun Wang is a professor in the School of Software at Tsinghua University in China. His research interests include social network analysis, graph data management, and music computing.

Ling Feng is a professor of computer science and technology at Tsinghua University in China. Her research interests include context-aware data management toward ambient intelligence, knowledge-based information systems, data mining and warehousing, and distributed object-oriented database management systems. She has published more than 150 scientific articles in high-quality international conferences or journals, and received the 2004 Innovational VIDI Award by the Netherlands Organization for Scientific Research, the 2006 Chinese ChangJiang Professorship Award by the Ministry of Education, and the 2006 Tsinghua Hundred-Talents Award.

1041-4347 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.