Seeing is retrieving: Building information context from what the user sees

Karl Gyllstrom UNC-Chapel Hill / HP Labs
karl@cs.unc.edu

Craig Soules HP Labs
craig.soules@hp.com

ABSTRACT

As the user's document and application workspace grows more diverse, supporting personal information management becomes increasingly important. This trend toward diversity renders it difficult to implement systems which are tailored to specific applications, file types, or other information sources.

We developed *SeeTrieve*, a personal document retrieval and classification system which abstracts applications by considering only the text they present to the user through the user interface. Associating the visible text which surrounds a document in time, *SeeTrieve* is able to identify important information about the task within which a document is used. This context enables novel, useful ways for users to retrieve their personal documents. When compared to content based systems, this context based retrieval achieved substantial improvements in document recall.

ACM Classification Keywords
H.3.3 Information Search and Retrieval: Clustering

General Terms
Human Factors, Experimentation

Author Keywords
Contextual search, User modeling

INTRODUCTION

A document is much more than its contents. The activities, or tasks, that surround a document provides much of the user's *context* for a document, often expanding well beyond the document's content. While the result of a task might be a single artifact (e.g. a term paper document), the generation of that artifact requires the processing of information from many disparate sources (e.g. emails, web pages, images). When recalling a previously used document, users often recall the surrounding task that occurred when that document was used, in addition to — or in the stead of — the contents of the document itself [4]. For example, a user may not remember the name of a file they received attached to an email, but they may remember the contents of the email itself. This indicates that the information presented to the user throughout a task is valuable in classifying and retrieving files used during that task.

Traditional content-based schemes for classification and retrieval underperform on personal data sets because they fail to map the user's context to the file's contents. On one hand, they lose information presented to the user through other sources. For example, a user may be referring to information on an email while editing a related document; in this case, a content-based scheme has no way to include that contextual information when indexing the document. On the other hand, they integrate additional information that the user may be unaware of. For example, a user may open a PDF and look at only the first page; however, a content-based scheme will index all of the pages, which may include terms unknown to the user.

SeeTrieve addresses the disconnect between user-perceived context and file contents. Instead of indexing file contents, *SeeTrieve* captures and indexes text displayed at the user-interface level by applications. Using temporal locality, it creates a mapping between text snippets and files used while the snippets were displayed. This extends the traditional two-level mapping of terms to documents to a three-level mapping of terms to snippets to documents. Just as in a two-level index, *SeeTrieve* can use its three-level index to both classify documents, finding relevant terms from related text snippets, as well as retrieve documents, searching the index of text snippets and then following them to relevant documents.

Conceptually, the *SeeTrieve*'s three-level index performs *task-based* classification and retrieval by matching the contents of displayed text during a given task to the set of files used for that task. For example, a user who forgets the name of a file attached to a past email might remember the contents of the email. Issuing a query to *SeeTrieve* that matches the contents of the email would return the attached file.

To showcase the value of *SeeTrieve*'s three-level index for retrieval and classification, we implement two applications: document retrieval and context tagging respectively. Our user study with document retrieval shows that *SeeTrieve* recalls 70% more data on task-based retrieval without a loss of precision, and recalls 15% more data on known-item retrieval with only a slight drop in precision. Our user study with context tagging shows that *SeeTrieve*'s classification is considered accurate by users, even when documents contain no indexable data (e.g., images).

MOTIVATION AND RELATED WORK

Despite the continued abstraction of the user's document space, users' interactions with their documents have changed comparatively little. Users populate a spreadsheet in the same manner whether using Excel (local) or Google Spreadsheets (remote). Writing an email does not functionally change, regardless of whether the user does so with Outlook, Pine, or Yahoo's web mail. This signifies a trend of divergence between the user's activity in the *user interface layer* and its complementary activity in the *file system layer*.

This divergence makes the user interface layer an attractive space to capture activity context because it (a) is very tightly coupled with user interaction, (b) involves activity which is less sensitive to change (e.g., reading, typing), and (c) exposes the contents of the objects with which a user interacts, even when those objects have no "indexable" form (e.g., they are not local, they are application specific files, etc).

We have identified five aspects of divergence between the user interface layer and the file layer: *location*, *composition*, *presentation*, *interaction*, and *temporality*. Each of these aspects uniquely complicate personal information management, and have been a source of difficulty or failure for many existing systems.

Location

Location determines where file data is stored. In the past, most computer work took place on local files. With the advent of the Internet, more and more computer-based work has moved to remote files until, today, the functionality of many traditionally local applications has been replicated on web pages, such as Google mail [21] or Microsoft Office Live [20]. This trend presents difficulty to systems that try to support task management using strictly local data.

One common approach to this problem is to leverage web caching to capture remote data locally. For example, most desktop indexing tools [8, 9, 26] index the user's web cache and return these pages as results to searches. The failing of this approach is its reliance on particular application behavior to perform correctly. Already the introduction of uncached AJAX applications render such schemes inadequate, requiring further specificity (e.g., Google Desktop now indexes Google Mail data directly rather than treat it as web content). Furthermore, user activity is not limited to reading items; they also create content. While this content may manifest to a file (e.g., one saves their work), more and more this content is posted to the web (e.g., tagging photos on a web site) making it impossible to capture through the web cache.

Rather than entering into this data-capturing arms race, *SeeTrieve* proposes to capture information at the user-interface, rendering the location of the underlying data irrelevant.

Composition

In cases where file activity occurs on local files, systems still face the problem of file composition, or the fact that file management is often application-specific and difficult to generalize. While email clients vary little in terms of the mail reading functionality exposed to users, they vary widely in how an individual email manifests on the file system. Some mail clients store each message as an independent file, while others store emails to a single database. This convention precludes approaches which require fine grained file access information, like when a specific email was last opened [25, 23].

Similarly to location, most classification and retrieval systems address composition through application assistance. These systems have plugins for common file formats (PDF, PST, web pages, etc.) that given them some access to data stored in proprietary formats. This requires *a priori* knowledge of the types of applications and files that a user will interact with. Furthermore, the proliferation of web-based applications distributes a user's files among different control domains that make a unified search interface problematic. For example, each new web application requires a new public API through which retrieval tools can access their documents. Instead, *SeeTrieve* acquires text seen through the user interface and ties to file system events, removing the need to support each application separately in building context.

Presentation

Presentation refers to the divergence between file contents and the application's presentation of those contents. A PDF document may contain a large number of pages, but PDF readers typically do not reveal more than two pages to the user at a time. Consider a case where a user is only interested in a single page of the PDF: indexing the PDF's contents captures non-relevant information.

On the web, there is often a large difference between a file's contents (i.e., HTML code) and its appearance to the user through the browser. Client side scripting allows HTML page elements to change visibility: collapsing large drop-down regions and opening others interactively. Due to the stateful nature of modern, interactive web pages, it is often difficult to infer the user's interest from the contents of the HTML page alone.

Presentation is a problem in virtually every existing classification and retrieval system. However, by limiting itself to contents displayed by the application, *SeeTrieve* only captures text that the user actually views.

Interaction

Interaction refers to the disconnect between application activity and user activity. A user may have numerous application windows open while all but one are inactive or minimized. Because the user is only interacting with the visible window, it is likely that background activity performed by other active applications are acting without direction from the user. For example, a word processor may continue to generate automated save events on all of its open files even though the user has only recently interacted with one of the files.

Some task-clustering systems, such as CAAD [23] and SWISH [22], identify the active applications or tasks through user interface tracing. CAAD (also a file classification system) uses this information to identify which recent file events are relevant to the user, however, without dealing with the problems listed above, its utility is limited.

SeeTrieve captures focus information to identify active applications, but instead of trying to use this to filter content information, it captures the visible contents directly from the user interface layer.

Temporality

Temporality refers to the problem of knowing when, and for how long, a user interacted with data. Many recent systems, Watson [5] and others [27, 3, 13, 6], collect information about recently accessed files or web pages to recommend, or personalize, data on the web. Unfortunately, capturing "recently accessed" information can be complex. For example, web browsers maintain caches of recently viewed sites, but they provide no information about how long a user viewed that site. Consequently, a link the user accidentally clicked through and returned from would be considered equally important to a page the user spent an hour reading. Inferring this information from the duration between their access times is not trustworthy; consider the case where one page is opened quickly after another, though in a different browser window. Should the user switch back to the previous window and spend a long time viewing the page, analysis of the browser history would indicate the wrong page was viewed for longer.

Connections is a context-enhanced search tool that tries to deal with this problem by capturing all file system activity, allowing it to determine when and for how long a user examined a particular file [25]. It then uses this information to identify web-like relationships among local files. While it is susceptible to all of the previously described problems, its information is complementary, and could potentially leverage *SeeTrieve* to improve the quality of its contextual relationships.

Solution

SeeTrieve's combination of user interface and file layer information mitigates or minimizes each of the aforementioned forms. *Location* and *composition* are dealt with by capturing the contents of remote files or files with proprietary formats at the user interface layer. *Presentation* issues are resolved by the fact that the user-interface layer displays only what the application intends the user to see (e.g., the currently read page of a large document). *Interaction* issues are, by definition, best handled at the user interface layer, where information about what is and is not visible is managed. *Temporality* issues are resolved by *SeeTrieve*'s algorithms that combine the timing of user interface events with the timing of file events to create a mapping from text snippets to files weighted by the level of user interaction.

SEETRIEVE

Figure 1 illustrates the design of *SeeTrieve*. As the user interacts with applications and data, *SeeTrieve* collects data about their context, capturing visible text into *snippets* and tracing all file activity. Using these traces, *SeeTrieve* creates a bipartite *context graph* that maps between snippets and files. Finally, as examples of the utility of the context graph, *SeeTrieve* provides two applications: a document retrieval tool that combines a traditional search index on the snippet contents with the context graph, and a context tagging tool that identifies relevant terms for files based on their snippets. This section discusses these two components (data collection, context graph) and the example applications in more detail.

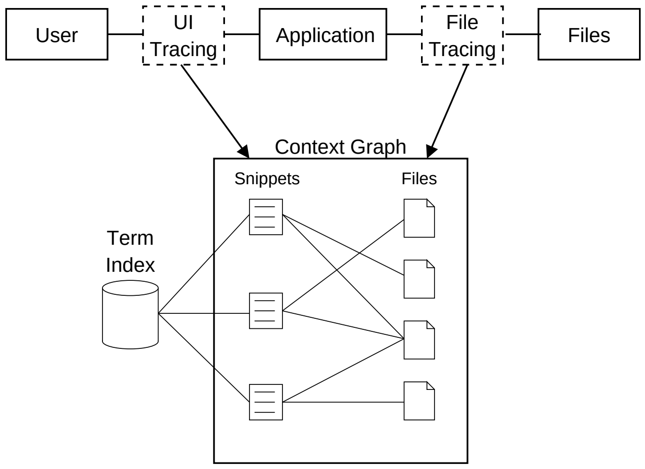

Figure 1. ***SeeTrieve*** **architecture.**

Data collection

Context-aware systems must be able to understand the user's behavior, independent of which applications and file formats they use. *SeeTrieve* solves this problem by tracing both user interface and file system events at the operating system layer.

These traces capture four pieces of information that *SeeTrieve* requires: text snippets of what the user sees, the times at which these snippets become visible to the user, the duration over which these snippets are visible, and the times at which files are accessed.

SeeTrieve acquires text snippets through the accessibility functionality, which has been historically used to enable third party applications that programmatically interact with the user interface to support impaired users. Accessibility data is exposed by most mainstream operating system graphical interfaces, including Windows XP, Mac OS X's Aqua, and Gnome. Accessibility support enables custom programs to query arbitrary applications for information about their UI state, such as which tab or pane is currently focused, and the contents of a text area. While accessibility information can be designed by an application's developers, the use of system components in UI construction means that much of this information is already provided.

SeeTrieve traces activation and minimization of application windows, informing it when windows go in and out of visibility and the duration over which they remain visible. Whenever a window changes visibility, *SeeTrieve* does a full capture of all visible text on the currently active window into a snippet and inserts that snippet into the stream of trace events. *SeeTrieve* also does periodic captures once every 3 seconds to handle cases where focus doesn't change, but the visible text does.

SeeTrieve traces file READ and WRITE operations to identify which files are accessed when. This trace of file system events is later merged with the trace of user interface events when creating the context graph.

Context graph

The relationship between snippets and files is represented by a bipartite *context graph*, with links between nodes indicating the strength of the contextual relationship between them. Creating the context graph requires two steps. The first is to merge similar snippets together, the second is to pair merged snippets with their related files.

Merging snippets

User activity often involves switching among multiple applications or windows. Because *SeeTrieve* treats every focus event as a new source of text, such activity can generate many snippets of identical text that originate from the same conceptual document (e.g., the same web page). Most classification and retrieval techniques rely on a discriminating value of terms in the corpus (often inverse document frequency). If a term appears frequently within a snippet while relatively infrequently in the rest of the corpus, it is considered informative. Consequently, populating the corpus with many duplicate snippets reduces *SeeTrieve*'s effectiveness at classifying and retrieving documents. Hence, we implemented a document similarity technique to merge similar and identical snippets to substantially reduce this effect.

By merging similar snippets, and not just identical snippets, *SeeTrieve* can deal with slight changes in visible text (e.g., status bar updates, open menus) while still identifying completely separate snippets (e.g., the next page in a PDF, a new web page). *SeeTrieve* identifies similar snippets using the *Max Hash* algorithm [12]. *Max Hash* uses landmark chunking (implemented with Rabin fingerprinting) to break snippets into variable sized chunks. Landmark chunking has the advantage that, because the chunk boundaries are chosen based on content rather than a fixed size window, small changes to the file will only change a small number of the chunks. Each chunk is then hashed using the *MD5* function, and the hashes are sorted numerically. If the top n hashes of two snippets' chunks match, then it is very likely that the snippets are similar[1]. In practice, we treat any two snippets that share more than half of their hashes as identical. Since snippet size is governed by the amount of text which can appear on a screen, the number of hashes for a snippet is small and sharing half of these hashes indicates with highly probability that the two snippets are very similar.

We chose *Max Hash* as our similarity metric because it is (a) robust to small changes in content and (b) efficient in performance and space. To find if a snippet has an existing similar snippet, *SeeTrieve* maintains two hashtables. The first contains *Max Hash* values as keys and snippets containing that hash within their top n hashes as values. The second is the reverse: snippets are keys and their top n hashes are values. When *SeeTrieve* witnesses a new snippet, it is chunked and hashed. For each of the top n hashes, *SeeTrieve* queries the hashtable for any snippets that contain the hash. It then finds the top n hashes for each matching snippet. If at least $\frac{n}{2}$ of the new snippet's top n hash values match an existing set of hash values, the two snippets are considered similar. This process requires only n lookups to find a similar file and the list of hashes for each file is n 32-bit values, thus both the computational and storage requirements are small.

[1]The higher n, the more similar the snippets must be.

Pairing snippets to files

The link weight between a snippet and a file node is increased when snippet S is seen in close temporal proximity to an event on file F. *SeeTrieve* captures this proximity through a *context interval*, a time period during which witnessed snippets are considered to be related to that file. A context interval of n seconds means that any snippet S witnessed less than $\frac{n}{2}$ seconds before or after an event for file F is related to F. Thus, snippets and files that are more frequently proximal will, generally, have higher relative link weights between them.

SeeTrieve strengthens links using two factors: *duration* and *temporal proximity*. Duration measures the length of time over which a snippet was visible. Intuitively, this captures the relative importance of the data contained within it. Let S_{start} be the point at which snippet S is seen that is not similar to the previous snippet in the trace, S_{end} be the point at which a new snippet that is not similar to S is seen[2], t_F be the time at which file event F occurs, and ci be the duration of the context interval. Then, Equation 1 defines the duration value for a snippet S and file F.

$$dur(S,F) = \frac{min(t_F + \frac{ci}{2}, S_{end}) - max(t_F - \frac{ci}{2}, S_{start})}{ci} \tag{1}$$

Temporal proximity measures the temporal distance between the snippet and a file event. The closer in time a snippet appears to a file event, the more likely it is to be related to the file event. Weighting by temporal proximity relates events over a longer period of time without introducing too much noise (e.g., an infinite context interval equally relates all snippets to all files). Then, Equation 2 defines the temporal proximity weight between snippet S and file F.

$$prox(S,F) = \begin{cases} S_{start} < t_F < S_{end} & 1 \\ o.w. & 1 - \frac{\min(|t_F - S_{start}|, |t_F - S_{end}|, \frac{ci}{2})}{\frac{ci}{2}} \end{cases} \tag{2}$$

When snippet S is visible at some point within the context interval of file F, *SeeTrieve* increases the value of the link between them by the product of duration and temporal proximity.

Application 1: Document retrieval

SeeTrieve implements document retrieval by combining a traditional content index[3] built over the snippet contents (shown as the term index in Figure 1) with the context graph. It maintains the content index by adding new snippets (i.e., snippets with no similar existing snippets) as they are seen.

To retrieve a document given a user query, *SeeTrieve* first passes the query to the content index to identify relevant

[2]The definitions of S_{start} and S_{end} merge sequences of similar snippets into a single snippet for the purposes of measuring visibility time. Due to polling, multiple snippets may correspond to a single window that has maintained focus.

[3]*SeeTrieve* can use either Indri [16] or Google Desktop [8] for the content index.

snippets and then uses the context graph to identify related documents. Specifically, the content index returns a pool P that contains a list of $\langle S_i, V_i \rangle$ tuples where S_i is a snippet and V_i is its corresponding relevance score. *SeeTrieve* then does a search on the context graph to identify R, the set of files most related to P.

R starts as an empty result pool to be composed of 2-tuples containing a file and its relevance score. For each snippet $\langle S_i, V_i \rangle \in P$, *SeeTrieve* retrieves each link to a local document $\langle F_j, L_j \rangle$ where F_j is the local file and L_j is the value of the link. It inserts F_j into R (if it doesn't already exist) and increases its relevance score by $(L_j \times V_i)$. Thus, in cases where a file contains incoming weight from numerous snippets, its relevance score contains the sum of each individually contributed relevance score. Finally, R is sorted by relevance score and returned.

Application 2: Context tagging

In *context tagging*, *SeeTrieve* takes a given file, finds related snippets, and uses their contents to create a textual summary – or *context zeitgeist* – of that file. Unlike content classification, which uses a file's contents to identify relevant terms for that file, context tagging uses the contents of the activity that surrounds a file while it is used to identify relevant terms, offering terms that the file's contents might not even contain.

For example, an image file on a user's computer might have no useful information text content within it. Let us assume that after downloading the image from their camera, the user uploaded the image to Flickr and entered a title, description, and tags for that image through the website. Because these operations generated a set of content events surrounding the file event for the image (e.g., the time it was uploaded), they will share links with that image on the context graph. The textual contents of these content events will contain useful pieces of information about the image: its title, tags, and description as entered by the user.

SeeTrieve's context tagging operates much like an inverted search. Given a file F, let P be the set of snippets related to F in the context graph. Let T be the set of tuples $\langle t_i, f_i, c_i \rangle$ where t_i is a unique term from the contents of the snippets in P, f_i is the total number of occurrences of term t_i, and c_i the count of snippets containing t_i[4]. Let D be the set of all snippets in the context graph. Let D_t be the set of all snippets containing a term t, identified through the term index. For each $\langle t_i, f_i, c_i \rangle \in T$, *SeeTrieve* computes a score for each t_i using a variant of *tf-idf* defined in Equation 3.

$$tf_i = \frac{f_i}{\sum_{f_k \in T} f_k}, idf_i = \log \frac{|D| - |P| + 1}{|D_{t_i}| - c_i + 1} \quad (3)$$

The effect of Equation 3 is to treat the set of snippets P as a single logical snippet. Thus, it calculates term frequency (tf_i) across the contents of all snippets in P, and calculates inverse document frequency (idf_i) as if all of the snippets in P were removed from the corpus and replaced with a single snippet containing the term.

[4]*Stop words*, or words considered too common to be useful in retrieval, are omitted from T.

SeeTrieve calculates the final *tf-idf* score for each term as the product of the term's tf and idf values, sorts the terms by their scores, and returns the list of terms as the file's context zeitgeist.

EVALUATION

The goal of our evaluation is to show the effectiveness of *SeeTrieve*'s two applications: task-based document retrieval and context tagging. Unfortunately, unlike traditional content-based retrieval and classification, context-based tools require that users interact with the data in realistic usage scenarios in order to gather the necessary traces, ruling out the use of an existing document corpus. Thus, our evaluation employs a two-phase user study in which users first interact with a data set while being traced by *SeeTrieve* and then later are asked to evaluate *SeeTrieve*'s two applications with respect to that data.

We chose to run all of the users in our study on a single machine under a single account. Limiting the scope of the content or context information to a single user would trivialize the task of finding related data (since all available data would be relevant). Merging the traces of several users simulates a single user performing a set of similar tasks over a longer period of time, providing a more realistic usage scenario for *SeeTrieve*.

User study design

Phase one

Phase one consisted of two user tasks. We chose the user tasks to include a mix of local and remote data, as well as a mix of content-rich (e.g., text) and content-free (e.g., image) data. Including all kinds of data highlights how *SeeTrieve* identifies useful context information regardless of the source, and even applies it to data that cannot be indexed through traditional means.

Task 1 was the creation of a conference trip report using a wiki, a web-based collaboration tool interfaced through a standard web browser, installed on a separate machine. The user was asked to create a wiki page briefly describing three papers from a fictitious conference. The user was instructed to choose three papers at random from a pre-generated corpus of conference paper files[5], skim each paper, write a brief (1-2 lines) summary of the paper on the wiki page, and upload the paper to the wiki. Once the user selected a PDF it was removed from the corpus to prevent overlap with other users.

Task 2 was the creation of an online photo album. The user started by creating a photo album using a photo album website installed on a separate machine. The user was given a topic (e.g., marine animals) and asked to identify three items within that topic (e.g., dolphins, manatees, and orcas). For each item, the user was asked to acquire an image of that item online, download it to their machine, and then upload it to the photo album. The user was then asked to provide a brief description of that item as researched online (e.g., through Wikipedia) and place that description within the "description" category of the photo on the photo site.

[5]Papers were selected from the ACM Digital Library.

Phase one included **15** users, **12** of which completed Task 1, **13** of which completed Task 2, and **10** of which completed both Task 1 and Task 2. Each task took users between 20 and 45 minutes to complete.

Phase two: Retrieval

We evaluate two aspects of *SeeTrieve*'s document retrieval: task-based retrieval and known-item retrieval. Task-based retrieval, specific to context systems, returns all of the items related to the task described by the user's query. Known-item retrieval, the more common form of document retrieval, returns a single item desired by the user.

Three to seven days after completing phase one, users were asked to return for the retrieval task. Because most users would be unfamiliar with task-based retrieval, we felt that asking them to perform and evaluate such retrieval tasks might introduce a bias toward *SeeTrieve*. Instead, we asked users to locate each document used in a task by performing known-item retrieval through *Google Desktop*, a traditional content-only desktop search tool. For the trip report task, this included the wiki page containing their report along with each paper they summarized. For the photo album task, this included each photo file, each Wikipedia page, and the page depicting their album[6]. Users were allowed to issue three queries for each document with the goal of generating a query that would return it as the first result.

We then used the most effective queries users formulated for each document, over 130, to evaluate *SeeTrieve*'s task-based retrieval and known-item retrieval, comparing its results against those identified by Google Desktop. In our evaluation, *SeeTrieve* began context building immediately when a user started their first task, and ended when their last task finished. Times were adjusted such that the end time of a user's task immediately preceded the start time of the next user, allowing events from one user's task to be present within a context interval of the following user's task (and vice versa), more faithfully emulating a single user switching between tasks.

In these experiments, *SeeTrieve* was parameterized with a 30 minute context interval, allowing any text viewed within 15 minutes of a file to be related. For task-based retrieval we report recall and precision values. For each document's query, we measure the task-based recall as the percentage of local documents from that document's task that were returned. We do not consider results where neither method was capable of producing one task item. We measure the task-based precision as the ratio of correct documents returned to total documents returned at the point where the last correct document is returned. For known-item retrieval we report recall and average position values. We measure average known-item recall as the percentage of queries that return the desired item. We measure average position based on the document's position within the result list, including only those queries that return the document.

To even the comparison between *SeeTrieve* and Google Desktop, we filtered the results of Google Desktop in two ways. First, we remove results that were not accessed at least once by a user (and hence, would not be within *SeeTrieve*'s index). Second, when comparing Google Desktop and *SeeTrieve*, we remove any web-cache results. Because *SeeTrieve* only indexes local files that have been accessed and the retrieval task only considers local files as correct results, to include other files (e.g., unaccessed files or web-cache results) would unfairly penalize Google Desktop. We also exclude results from *SeeTrieve* for files within known system directories (e.g., *Local Settings*), as Google Desktop considers these files irrelevant, and to include them in *SeeTrieve* would unfairly penalize it.

[6]Note that although the web documents were not stored locally, Google Desktop indexes the browser cache, allowing users to identify terms that would successfully retrieve the page as if it were local.

	SeeTrieve		Google Desktop	
Task	Recall	Precision	Recall	Precision
$Task_1$	0.945	0.500	0.314	0.557
$Task_2$	1.000	0.712	0.177	0.491
$Task_{all}$	0.964	0.561	0.267	0.538

Table 1. Task-based retrieval.

	SeeTrieve		Google Desktop	
Task	Recall	Position	Recall	Position
$Task_1$	0.701	3.535	0.657	1.465
$Task_2$	0.963	2.000	0.519	1.214
$Task_{all}$	0.777	3.048	0.617	1.404

Table 2. Known-item retrieval.

Table 1 lists the task-based recall and precision values for both *SeeTrieve* and Google Desktop. For task-based retrieval *SeeTrieve* achieves nearly 100% recall with the same precision as Google Desktop. This indicates that users could retrieve any document used in a task by remembering just one document from that task. Even in the case of remote documents (e.g., the wiki page) this holds true, highlighting *SeeTrieve*'s ability to utilize information from any source when retrieving local data. Note that given the task sets were small, 4-7 documents, a precision of 50% indicates that all of the documents would be listed in the first 15 results.

We believe that the results of *SeeTrieve*'s task-based retrieval should be considered in isolation. Because Google Desktop was not designed with task-based retrieval in mind, a direct comparison against *SeeTrieve* is less meaningful. Furthermore, in the retrieval task users issued queries intended to recall individual items. Had they issued queries to find as many familiar items as possible, their search strategies might have been more general.

Table 2 lists the known-item recall and average position for both *SeeTrieve* and Google Desktop. As compared to Google Desktop, *SeeTrieve* recalled more items but, on average, positioned those items slightly further down the result list. This illustrates two results. First, despite the increase in average position, *SeeTrieve* placed results well within the first page of results, indicating that its known-item retrieval could replace traditional content-only retrieval with little effect on the user.

Second, *SeeTrieve* found documents when Google Desktop did not, especially in the image retrieval task, again showing the relevance of user-interface text when applied to content-free data. For example, a search for "James Gleick" through Google Desktop was unable to retrieve the image file "log1.jpg" because neither the contents nor the name of the image were relevant, while the same search in *SeeTrieve* was able to retrieve the image. In cases of PDF recall, the slight improvement in recall was largely due to users unknowingly placing too much information in their query for Google Desktop to work. For example, a search for "hierarchy projection paper" failed in Google Desktop because the term "paper" was not present in the document itself, though present in the context (e.g., the wiki summary was titled "paper review").

The success of *SeeTrieve* in task-based retrieval shows that (a) as an element of task, a document contributes some content to that context, (b) a query that identifies a document can also identify the context of which it is a part, and (c) a query that identifies a context should identify all files which were used as part of that context.

While users worked with specific applications in this experiment (e.g., a PDF reader, a browser), it is important to note that the way in which context was collected and applied was application independent. Had the users been instructed to report their summaries in an email rather than a web page, the text they generated would have still been available to *SeeTrieve* and useful in retrieval. Given the contents of this email would be acquired by its screen text rather than its file contents, *SeeTrieve*'s access to the information would persist regardless of whether users' emails were through Outlook or Google Mail. Hence, *SeeTrieve* enables context retrieval without making any assumptions about applications beyond the fact that they must eventually present text that is meaningful to the user through the UI.

Phase two: Classification

To test *SeeTrieve*'s context tagging, we need to show that the terms it identifies as relevant are familiar to the user of that file. We chose one local file at random from each of the two tasks (i.e., one PDF and one image) for each user and generated a zeitgeist for each file using context tagging, which we term the *context* zeitgeist. We also placed each of the PDF's into a single content index using Indri [16], and asked it for the set of keywords it considered most relevant for each PDF, creating a content zeitgeist for each PDF, which we term the *decoy* zeitgeist. In these experiments, a context interval of 10 minutes was used. We then presented users with five zeitgeists for each of their two randomly chosen files[7]. For the PDF we presented the context zeitgeist, the decoy zeitgeist, and three other randomly chosen context zeitgeists for other files not accessed by that user, which we term *incorrect*. For the image, we presented the context zeitgeist, and four incorrect zeitgeists. We asked the user to rate each zeitgeist on a 3-point Likert scale, where 3 indicates that the terms describe the file well, and 1 indicates that the terms are irrelevant for the file.

Figure 2 illustrates an example zeitgeist produced for an image chosen by a user during the photo album task on the topic "philosophers." We draw three points from this example. First, 15 of the first 20 words are relevant to the file, either describing the topic, task, or filename. Second, both the topic of choice, philosophers, and the source of information, Wikipedia, are represented in the zeitgeist, either of which the user may recall when trying to retrieve an item. Third, many of the irrelevant words are included because there is not enough overall system data to exclude them. For example, words such as thumbnail, item, add, sort, administrate, etc. would reduce in significance as a user interacted with the photo album software during other tasks, as their discriminating value would weaken.

plato, ***philosopher***, ***socrates***, **album**, thumbnail, item, **subalbum**, **upload**, file, ***hegel***, ***bc***, ***philosophy***, ***athens***, **photo**, platon, ***kant***, use, time, **caption**, **wikipedia**, size, add, sort, ***ancient***, default, ***greece***, edit, apply, description, ***oracle***, summary, administrate, megabyte, set, ***philosophic***, ***argue***, date, option, create, ***western***, charge, **gallery**

Figure 2. The zeitgeist produced for an image from a user's photo album task for the topic "philosophers". ***Bold italicized*** **words describe the topic, bold words describe the task, and the underlined word was contained within one of the images' file name.**

Task	Target	$\bar{\chi}$	σ	t-test	u-test
1	Correct	2.50	0.80	—	—
1	Decoy	2.08	0.79	0.213	0.092
1	Best Incorrect	1.58	0.79	0.010	0.008
2	Correct	2.91	0.30	—	—
2	Best Incorrect	1.27	0.47	<0.001	<0.001
1 + 2	Correct	2.70	0.63	—	—
1 + 2	Best Incorrect	1.43	0.66	<0.001	<0.001

Table 3. Classification results. The mean scores of *decoy* **and** *best incorrect* **are compared to the mean score of** *correct* **using the t-test and u-test. The P-values from these tests are depicted in the final two columns.**

Table 3 lists the results of our classification experiment. When considering the incorrect zeitgeists, we took the highest scored incorrect zeitgeist for each user's task and averaged that score across users. For example, if a user for task 1 scored the incorrect zeitgeists 1, 1 and 2 respectively, we considered 2 as the best incorrect score and averaged those scores across users for task 1. For each zeitgeist, we present the average score, standard deviation, and P-value as calculated by the Student's t-test and Mann-Whitney U test between that zeitgeist and the context zeitgeist. We show the results for each task, and the average across both tasks.

We draw three conclusions from these results. First, the context results are significantly better than the best incorrect result in all cases, indicating that context tagging is successful. Second, the context results in Task 1 perform as well the decoy results[8]. This indicates that *SeeTrieve*'s snippets are able to capture the relevant text of an indexable document at least as accurately as document content alone. Third, the context results for Task 2 are extremely accurate, achieving an average score of nearly 3. This indicates that *SeeTrieve* accurately classifies documents that contain

[7]To avoid triggering memories with users for the retrieval evaluation, this phase always followed the retrieval evaluation, typically by 1-2 days.

[8]Although the average score is higher for context, we did not have enough users to show a statistically significant difference, as evidenced by the p-value.

no indexable terms at all, an impossible task with traditional content-based schemes.

We believe *SeeTrieve* classification could be applied in cases where users have documents whose origin or use they do not recall. For example, when discovering an unfamiliar document in a long-before used folder, enabling the user to see important words from the surrounding activity might reveal important insight (e.g., the paper was downloaded in a previous literature review).

Summary

The evaluation of personal information management tools remains a difficult problem [11, 19]. While our evaluation was designed to illuminate the abilities of our system, we believe there are lessons from the evaluation that could constitute a contribution to personal information evaluation. Hence, we detail the advantages, disadvantages, challenges, and nuances in our approach.

Controlled vs. Field study

The effects of time and data set size are important to consider in personal information management. In a controlled study, there is the danger of too little data, making retrieval tasks trivial or uninteresting. This is less likely in a live deployment; however, building a sizeable data set for a single user requires a large amount of time and may be impractical.

We address this problem by having multiple users share the same computer at different times to simulate the effect of a single user working on multiple, similar tasks. While this enhances the amount of data that can be collected within a limited time period, it introduces a challenge during retrieval that users issue queries on a corpus that contains large amounts of "personal" data of which they are unaware. This blind spot occasionally manifested in unsuccessful queries (e.g., a user searches for "ACM pdf" on a system containing over 100 ACM papers). We addressed this problem both through the design of the data creation tasks and the design of the retrieval task.

Data creation

When designing our phase one tasks, we considered two points. First, that users should be made aware of the expanded corpus of "personal" data, outside of the files they directly interact with. Second, that because the other data on the machine is unknown to the user, the users should have discriminating information about their files that they can use to avoid overlapping terms that might, unintentionally, retrieve another user's data, as this would not occur if a single user performed all tasks.

To inform users of existing data in the paper review task, users selected their papers from a local folder that was populated with a large number of papers. This tacitly communicated that they might need to use more specific keywords when later retrieving the document. To prevent overlap in the paper review task, papers that were read by one user were removed from the papers directory after they completed their summary to avoid two users sharing the same item. In the photo album task, each user was assigned a unique topic area to research, resulting in distinct sets of relevant keywords among users.

Retrieval

We used query refinement during the retrieval task to further mitigate the problem of unknown "personal" data. For each document, users were given three opportunities to generate a query that placed the document as the first result in the list.

Although this approach improves a user's ability to isolate his or her own files, we also want to prevent users from utilizing information within a set of results to refine their query, as this context information, if used in a query, could falsely boost the results of *SeeTrieve*. Although use of context information independent of the query results is our expectation, we did not want to guide users to use context information during query refinement. To ensure this we developed a thin wrapper to Google Desktop that reveals only the file name in the result, and all papers were given cryptic names from the beginning to ensure that author or title information could not be derived from the results.

DISCUSSION

It is important to consider not only how the systems differed in performance, but *why* they differed. In this section, we identify the major cases in which the content based system was not amenable to the way in which the user recalled their document.

Implicit Linking

During their retrieval task, many of the users reported — especially when unable to recall documents — their preferred retrieval method would have been to first find the wiki page or photo album page, which typically linked to the forgotten documents. This illustrates a case in which users would have applied relationships between documents as a retrieval tool. Since *SeeTrieve* was able to implicitly recreate this linking through the user's activity, it was successful in retrieving task files even when the user forgot enough details about them to form a successful content query.

Abstract vs. Detailed recall

Of the 12 users who completed the paper reviewing task, only 2 were able to recall all three of the reviewed papers. Users were more successful with the photo album task, with 9 of 13 users recalling all three photos used. The most apparent quality observed was that in most cases users remembered their documents abstractly rather than in detail. Below, we discuss two areas where this tendency manifested.

Author vs. Reader

Although users tended to forget at least one read paper, in almost every case they recalled the summary page they created. We attribute this to the personalization derived from authorship.

Many users recalled keywords from the paper they had placed on their summary pages, and used them as queries to retrieve their papers. This suggests that users place words in their summaries that they believe are effective descriptors of the read document, and, by identifying these words and using them in authoring, are more likely to recall them. In practice, these descriptors were very effective in retrieval.

Also, users often remembered words that were of their own origin and successfully applied them in searches for their summary page. For example, a number of users recalled words from the unique or clever title they produced for the document.

We believe that the act of authoring summaries forces users to engage with the documents they read in more depth. When summarizing, users choose words that intuitively describe the document to them; these words reflect the user's own concept of the document and do not necessarily have to exist within the document itself. Because users are more prone to recalling information through an authoring task, we believe that this property of recollection, not accomplished well by existing content based systems, is important to support.

Topic vs. Item

In the photo album task, every user recalled the broad topic for which they acquired images, and were able to retrieve their album web page in every instance. However, they were not always able to retrieve each individual photo. We believe there are two reasons that explain this observation.

First, as in the paper review case, there were instances where an item within the topic was forgotten. Some of the users were able to remember specific aspects of their topic that lead them to originally choose to research the forgotten items. These aspects were usually captured in the summary of their photo album. For example, one user, given the topic of "dinosaurs," selected a specific time period within which to select particular instances. In this case, a search for "Triassic period" retrieved their photo page.

Second, users often remembered specific items without recalling important features of those items that would be necessary for retrieval. One user, researching politicians for the photo album task, recalled a specific politician whom he or she chose but could not recall the exact spelling of the name, preventing a successful retrieval of the image file. When trying to recall the web page from which the photo originated, the user applied alternate information about the candidate, issuing a successful query including the state which the candidate represents. This was an application of knowledge *about* the item that simply did not exist *within* the item itself.

These scenarios indicate that users are more likely to remember broad topic information about a document than specific details about it. By capturing task-based context information, *SeeTrieve* is more likely to contain keywords a user is likely to remember, improving document recall.

Summary

An important lesson in this work is that users are generally more able to recall the context in which a file used than the file itself. One of the primary reasons for this is that this context often contains information about the personal ways in which a user conceptualizes a document.

In the process of doing a literature search for a research paper on contextual retrieval, one might issue the query "papers on contextual retrieval", to which a search engine like Google might be able to return papers on a conceptually similar topic like "personalized search". This retrieval is enabled in part by the fact that the hyperlinked structure of the web can leverage the multiple ways in which the universe of users organizes information. For example, an individual might link to a "personalized search" paper within their "context retrieval" web page, enabling search tools to connect the similar concepts. In local document retrieval, this structure cannot be leveraged. However, being able connect the user's initial query to the document which was ultimately retrieved through a system like *SeeTrieve* allows the user to implicitly describe their own documents through their behavior.

FUTURE WORK

Our evaluation of *SeeTrieve* was designed to show a few cases in which context might be useful in retrieval. The next important step in this research is to evaluate how the system will perform in more natural settings, where context can be leveraged in the myriad ways in which the modern user interacts with their documents. Additionally, we are interested in identifying the effects of time and multiple use on documents. For example, it would be useful to know what features of context would be useful in documents with which the user frequently interacts.

We also plan to extend *SeeTrieve* to enable the retrieval of snippets themselves. Due to the abstract nature activity on the web, there is not always a clear mapping between a set of text and a retrievable item. For example, personalized "start pages" like *iGoogle* can show a listing of recent news article headlines which frequently changes. As dynamic content, these pages have no concrete or permanent representation; future accesses to the same URL will likely depict different content, and a user might not be able to recall a headline for an article they later wish to read. As a set of text, such a page would be indexable to a system like *SeeTrieve*; by returning this item upon a query, an important information need could be satisfied.

CONCLUSIONS

This paper presents three contributions to the personal information management and task retrieval communities. First, it details the advantages to building context from the text revealed in the user interface and addresses the challenges in interpreting this information in the absence of application information. Second, it presents methods for correlating text in the user interface with local documents using temporal locality and the results of its user study provide strong evidence that these techniques are effective. Third, its evaluation methodology can inform the evaluation of future task-based or personal information management systems.

ACKNOWLEDGEMENTS

We thank Diane Kelly and Alistair Veitch for their assistance in designing and implementing the user study, and the participants in the study for their valuable time and feedback.

REFERENCES

1. R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

2. D. K. Barreau. Context as a factor in personal information management systems. *Journal of the American Society for Information Science*, 46(5):327–339, 1995.

3. T. Bauer and D. Leake. Using document access sequences to recommend customized information. *Intelligent Systems, IEEE*, 17(6):27–33, 2002.

4. T. Blanc-Brude and D. L. Scapin. What do people recall about their documents? Implications for desktop search tools. In *IUI '07*, pages 102–111, New York, NY, USA, 2007. ACM Press.

5. J. Budzik and K. J. Hammond. User interactions with everyday applications as context for just-in-time information access. In *IUI '00*, pages 44–51, New York, NY, USA, 2000. ACM Press.

6. P. Chirita, C. S. Firan, and W. Nejdl. Personalized query expansion for the web. In *SIGIR '07*, pages 7–14, New York, NY, USA, 2007. ACM Press.

7. E. Cutrell, S. T. Dumais, and J. Teevan. Searching to eliminate personal information management. *Commun. ACM*, 49(1):58–64, 2006.

8. Google Desktop. http://desktop.google.com.

9. Yahoo! Desktop. http://desktop.yahoo.com.

10. S. Dumais, E. Cutrell, JJ Cadiz, G. Jancke, R. Sarin, and D. C. Robbins. Stuff I've seen: a system for personal information retrieval and re-use. In *SIGIR '03*, pages 72–79, New York, NY, USA, 2003. ACM Press.

11. D. Elsweiler and I. Ruthven. Towards task-based personal information management evaluations. In *SIGIR '07*, pages 23–30, New York, NY, USA, 2007. ACM Press.

12. K. Eshghi and H. K. Tang. A framework for analyzing and improving content-based chunking algorithms. Technical Report 30, Hewlett-Packard Labs, 2005.

13. X. Fu, J. Budzik, and K. J. Hammond. Mining navigation history for recommendation. In *IUI '00*, pages 106–112, New York, NY, USA, 2000. ACM Press.

14. K. Gyllstrom, C. Soules, and A. Veitch. Confluence: enhancing contextual desktop search. In *SIGIR '07*, pages 717–718, New York, NY, USA, 2007. ACM Press.

15. D. M. Hilbert and D. F. Redmiles. Extracting usability information from user interface events. *ACM Computing Surveys*, 32(4):384–421, 2000.

16. Indri. http://www.lemurproject.org/indri/.

17. V. Kaptelinin. Umea: translating interaction histories into project contexts. In *CHI '03*, pages 353–360, New York, NY, USA, 2003. ACM Press.

18. D. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A General Purpose Information Management Tool for End Users of Semistructured Data. In *CIDR*, 2005.

19. D. Kelly. Evaluating personal information management behaviors and tools. *Commun. ACM*, 49(1):84–86, 2006.

20. Microsoft Office Live. http://office.microsoft.com/en-us/officelive.

21. Google Mail. http://mail.google.com.

22. N. Oliver, G. Smith, C. Thakkar, and A. C. Surendran. SWISH: semantic analysis of window titles and switching history. In *IUI '06*, pages 194–201, New York, NY, USA, 2006. ACM Press.

23. T. Rattenbury and J. Canny. Caad: an automatic task support system. In *CHI '07*, pages 687–696, New York, NY, USA, 2007. ACM Press.

24. J. Shen, L. Li, T. G. Dietterich, and J. L. Herlocker. A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *IUI '06*, pages 86–92, New York, NY, USA, 2006. ACM Press.

25. C. A. N. Soules and G. R. Ganger. Connections: using context to enhance file search. In *SOSP '05*, pages 119–132, New York, NY, USA, 2005. ACM Press.

26. Mac OS X Spotlight. http://www.apple.com/macosx/features/spotlight/.

27. K. Sugiyama, K. Hatano, and M. Yoshikawa. Adaptive web search based on user profile constructed without any effort from users. In *WWW '04*, pages 675–684, New York, NY, USA, 2004. ACM Press.

28. J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. In *SIGIR '05*, pages 449–456, New York, NY, USA, 2005. ACM Press.

29. R. W. White and D. Kelly. A study on the effects of personalization and task information on implicit feedback performance. In *CIKM '06*, pages 297–306, New York, NY, USA, 2006. ACM Press.