Refinding Data using Context based Memory Technique

Kajalekar S. J.
M. B. E. Society's
College of Engineering,
Ambajogai, Maharashtra

Patil B. M.
M. B. E. Society's
College of Engineering,
Ambajogai, Maharashtra

Chandode V. M.
M. B. E. Society's
College of Engineering
Ambajogai, Maharashtra

ABSTRACT
Data retrieval is a major aspect of data mining. Many times users need to access the information they have previously come across, i.e. refinding the information. In this research, ReFinder, which is a context based information refinding system, is used. It uses natural recall characteristics of human memory. By this, users can refind files and web pages according to their previously accessed context. A query by context model is built over a context memory snapshot. These instances are organized in a clustered and associated manner and evolve in life cycles just like the human brain. An eight weeks study was observed and time, place and activity were found to be useful recall clues. Experimental results show that the technique of associative clustering leads to best precision and recall. On average, 16.5 seconds are needed to complete a refinding request against 86.32 seconds with other existing methods. Future challenges like automatic annotation and context degradation are also discussed.

Keywords
Information refinding, context memory, association based clustering, decay

1. INTRODUCTION

1.1 Motivation
Computer and internet are becoming the basic needs for human beings these days. It includes reading, writing and collecting different kinds of information from these sources. However, people tend to revisit the information that has been come across intentionally or occasionally.

What makes refinding a tedious job is explosion in the amount of personally accessed information. And sometimes it can become as challenging as finding the information itself, even though these two are different things. Information finding is an uncertain process because users do not know enough information, while refinding is a more directed process as users have already seen the information before [1].

Maintaining access logs is a general way to support refinding [2]. As logs grow with time, users prefer searching logs for information which was accessed a long time ago. But there is a major problem with logs, that because of human's dim memory of the past [3], sometimes its difficult and time consuming to refind by simply entering keywords of previously accessed context.

Context can serve as a powerful cue for information recall rather than detailed information content [4]. For example, it may be hard to recall a person's name whom we met a year ago, but time, place and surroundings leaves a deep impact, which can serve as useful cue to remember that person.

Episodic memory enables humans to be consciously aware of earlier experiences under context [5]. It stores episodes or events together with their temporal-spatial relations. Association and context are crucial in episodic memory. The memory trace which is the central representation of to be remembered event is a multidimensional collection of elements, features and attributes.

2. LITERATURE SURVEY
Refinding is mainly useful in two aspects. Firstly, the web search and secondly, personal information management communities.

2.1 Web Search
There are multiple methods to organize web information for reaccess and reuse. Typical of them involve bookmarks, search engines etc.

MacKay et al. proposed 'landmark' system [6]. It is an extension to the traditional bookmarks. It is a user-directed technique that helps users in returning to specific content within a previously visited Web page. 'Contextual Web history tool' improves the visual appearance of the history. It combines thumbnails of Web sites and snippets of contents. By this, assisting users to easily browse or search the history by time. Google's 'Web history' stores users' search requests and clicked pages. It then classifies them into different topics such as images, news, and so on. Users can then navigate or search accessed Web pages by keywords from page titles and contents. The 'SearchBar' [7] tool allows users to organize their search keywords and clicked pages under different topics. Users can make notes on the topics for easy navigation. Teevan built 'Re:Search' system supporting simultaneous finding and refinding on the Web. When a user's query is similar to a previous query, it obtains the current results from an existing search engine, and fetches relevant viewed results from its cache. The newly available results are then merged with the previously viewed results to create a list that supports intuitive refinding and contains new information.

2.2 Personal Information Management
Dittrich and Salles [8] presented an 'iMeMex' data model to represent unstructured, semi structured, and structured personal data inside a single model. Based on that, a system was implemented offering some contextual information (graph connections, time and lineage) on query results. Dumais et al. developed a system called 'Stuff I've Seen' to facilitate personal information reuse. It builds index for what a person has seen, and uses some cues (e.g., file-type, access date, and author) for filtering and sorting results. Cai et al. developed a SEMEX system that enables users to browse personal information by semantic associations created from data items on one's desktop. Chau et al. developed a system which supports multilevel associative retrieval of desktop information. Salles et al. presented association trails to define associations among items in a data space. Chen et al. built a desktop search system that exploits semantic associations among files, mining from contents such as similar-to relationship and users' such operations as jump-to, copy-from, same-task, and so on. Soules and Ganger developed a file

search tool combining content-based search with temporal relationships between files gathered from user's file operations. Hailpern et al. presented a contextual history-based search tool. It enables users to search relying on temporal relationships (before, during, and after) between data items.

3. CONTEXT MEMORY

Context memory is the backbone of ReFinder system, and it is built as follows:

3.1 Framework of Context Memory

Human brain stores information which is repeatedly accessed or too important to lose. And this is done by creating link between multiple neurons. To mimic this feature, this framework is divided into 2 parts, called as Short Term Context Memory (SCM) and Long Term Context Memory (LCM).

- Short-term Context Memory lasts for a short period of time i.e. mere seconds. It has a limited capacity.
- Long-term Context Memory lasts as short as few days or even decades. It is unlimited in capacity. It is further divided into two memory units: permanent and evolving. Evolving unit will eventually decay but permanent unit will keep record of lifelong accessing experiences.

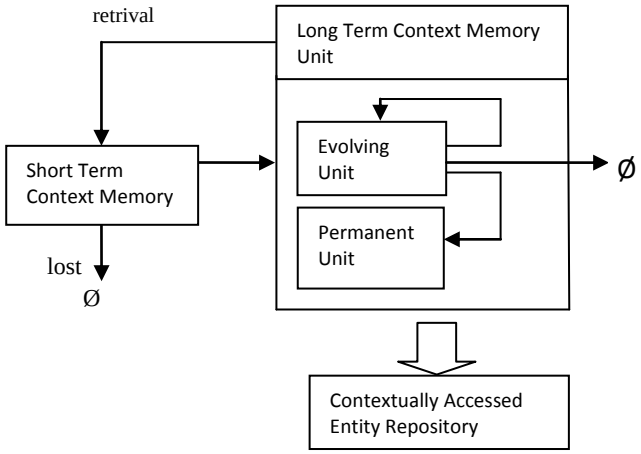

Figure 1. Framework of context memory

Contextual information can be user related (like user name, activity, agenda etc.) or external environment (like time, date, place, surrounding people, etc.)[5]. If accessed information is of interest to user, a linkage between access context instance and information identifier is created. It is stored in contextually accessed entity repository.

Information transition across the two memory units as follows:

1. For an accessing event received by SCM, if the user engages in "rote" rehearsal of it by storing the information into the contextually accessed entity repository, it will be transferred to LCM; otherwise, it will be lost very quickly.
2. In LCM, if the access context is profound or harmful to the user (e.g., dangerous disaster situation), it will be stored in the permanent storage; else in the evolving unit. Most effective accessing events are memorized in the evolving unit due to the infrequency of permanent cases in one's life.
3. Contextual information in the evolving unit will decay gradually in life-cycles as time goes by.
4. When a context instance in LCM is recalled, it is brought back to SCM to strengthen its freshness and retention, thus slowing down the degradation.

3.2 Static Status of Context Memory

3.2.1 Contextual attribute and context

Access context is comprised of n contextual attributes (A1, A2,..,An). Each contextual attribute domain forms an ordered hierarchy of levels of abstraction. The hierarchy of context attribute A is a lattice ($H \prec h$) where H= (h_1, h_2,.., h_{s-1}, All)of s levels corresponding to the levelId (1, 2, . . . , s-1, s).

The edge linking two consecutive hierarchical levels h_i and h_{i+1} in H has a weight in [0, 1] to express the hierarchical similarity between hi and hi+1. The attribute values at two higher levels are more common and have less discrimination than those at two lower levels, which means if level is high, then similarity between two attributes at same level will be low.

3.2.2 Context Instance

Context instance is represented as tuple C= (c_1, $c_{2,\ ....}$, c_n) where c_i is from domain of A_i. Therefore, context instance can be defined as instantiation of its n contextual attributes.

4. CONTEXT BASED REFINDING

In this refinding technique, the context memory snapshot is organized into a hierarchical, clustered and associated manner, and evolves dynamically into life cycles.

4.1 Context based Refinding

A context-based refinding query can be denoted as a function RF(Q;CM) = $\langle$ C1; C2; . . . ; Cm $\rangle$, where Q is the query request formulated in the form of a context instance, CM is the query target that is the context memory snapshot, and the intermediate query result of Q upon CM is a ranked list of context instances in CM, $\langle$ C1, C, . . . , Cm $\rangle$, whose ranking is determined by a ranking function. There 3 ranking methods taken into consideration, named as simple similarity, negative dissimilarity and weighted similarity.

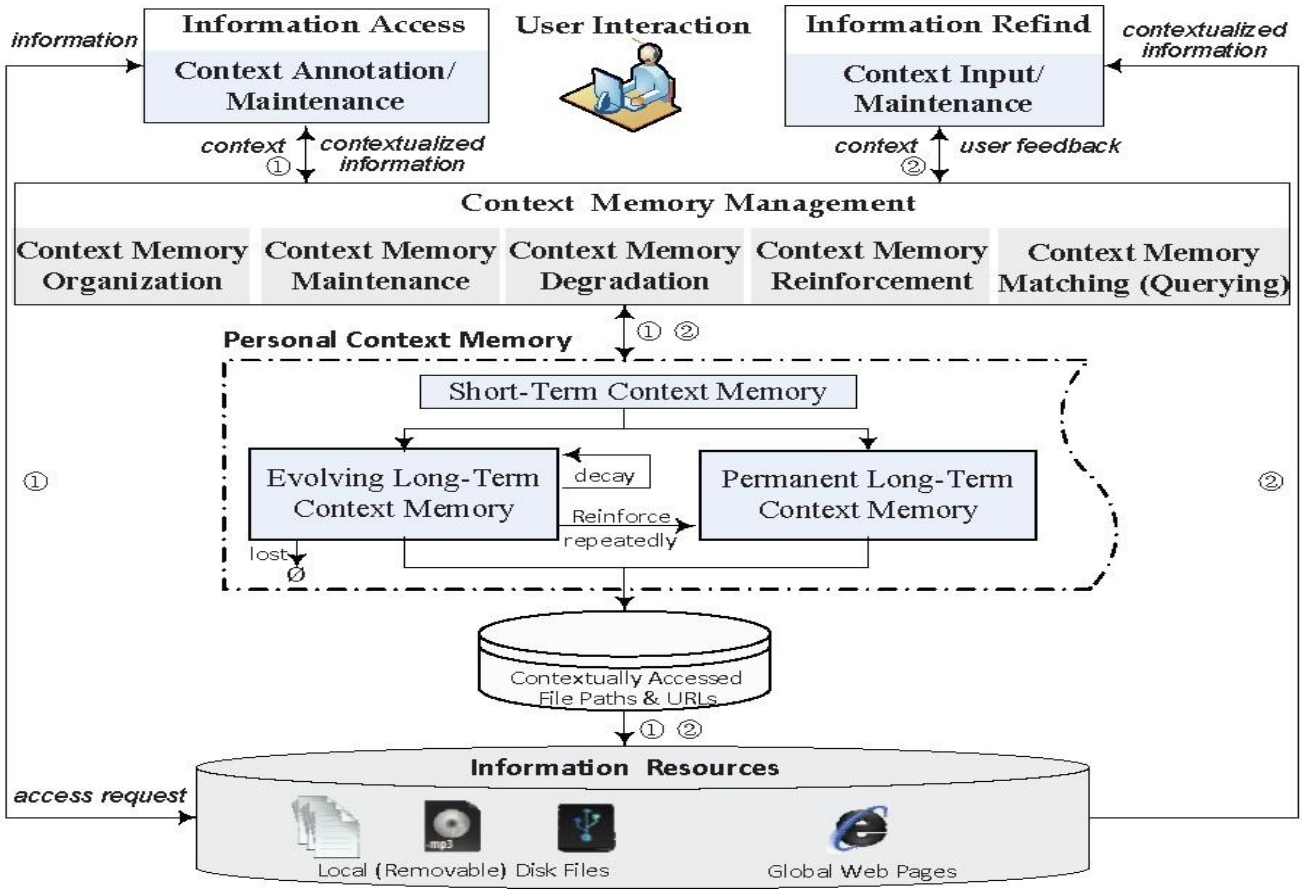

Figure 2. Architecture of Refinder

4.1.1 Context based refinding processing

Following is an approach making use of clusters and association rules among context instances.

1. Firstly identification of an attribute value r for a new cluster CC (A_i, r) is done. From set of unclustered context instances in CM, the one whose attribute value is at higher hierarchical level should be found. Then it is taken as representative value r of the cluster.
2. For each unclustered context instance C in CM, firstly its attribute value is compared to r. If it is at the same level or descendent of r (provided that their similarity is greater than or equal to threshold) then C is put into CC(A_i, r)
3. Repeat steps 1 and 2 until all instances in memory are clustered.
4. For each contextual attribute and every value in its hierarchy, an association chain Chain(A_i, v) is built.

   It consists of all context instances within same attribute value of A_i.
5. Association chains should be extended to include all descendents based on contextual hierarchy and obtain

   EChain(A_i, v) so that every context instance belongs to EChain(A_i, v) when (c_i =v) or ($C_i \prec_a$ v).

   Given a query Q, check matching of each chained context instance, starting from extended chain with shortest length against other values requested in Q. If there are association chains for context instances within each cluster, irrelevant chains can be wiped out.

The pseudo code of cluster association based refinding is as follows:

Algorithm 1. Cluster-δ-Association-based Refinding [9]

Input: A context memory snapshot CM and query Q

Output: A ranked list of context instances L that match Q

1: Let CLUSTER-SET = {CL(A_1), CL(A_2),..CL(A_n) be a set of cluster sets, where CL(A_i) = {CC(A_i, r_1), CC(A_i, r_2),…CC(A_i, r_s)} for every ($1 \leq i \leq n$);

2: L =Ø;

3: A_i = SelectAtt(CLUSTER-SET, Q);

4: for each CC(A_i; r_j) Є CL(A_i) do

5: if ($r_j = q_i$) ∨ ($q_i \prec_a$ rj ∧ sim(A_i, q_i, rj) ≥ δ)∨ ($r_j \prec_a q_i$) then

6: A_k= GetAssociationAtt(CC(A_i, r_j));

7: EChainy(A_k; q_k) = GetEChain(CC(A_i, r_j); q_k);

8: for each C Є EChain(A_k; q_k) do

9: if (C = Q) ∨ (C $\prec$ Q) then

10: Add C to L;

11: L = Rank (L,Q);

12: return L;

5. IMPLEMENTATION OF REFINDER SYSTEM

ReFinder is made up of four major components as shown in fig.2 which include information access, information refind, context memory management and a database of contextually accessed file paths and URLs [9].

- Information access. This component facilitates users to annotate their accessed interesting files/Web pages with the access context.
- Information refind. This component accepts users' context-based refinding requests, and returns the result files/Web pages.
- Context memory management. To process contex t based information refinding requests, the core context memory management component needs to do a bundle of work related to the organization, maintenance, degradation, reinforcement, and matching (i.e., querying) of the personal context memory.
- Database of contextually accessed file paths and URLs. Each context instance in the context memory links to the accessed files or Web pages, whose file paths and URLs as well as the titles are kept in the database of contextually accessed file paths and URLs.

For making use of ReFinder, users have to create an account and log in. Then they can annotate the files or sites of his interest in the ReFinder system, along with access context. ReFinder contains five contextual attributes, which are time, date, activity, place, use (and extension for local files). Time and date are automatically entered by the system. Users have to manually enter other attributes. To refind previously accessed local files or web pages, users request their queries by indicating corresponding access context through UI. For example, users can type time, place, activity, etc.

Users' inputs may not always be precise because of degradation of human memory as time passes. In ReFinder system, time and date are provided by the system, but other information is user specified. ReFinder identifies closely matching context units from context memory and returns linked files or pages stored in personal linkage repository.

Demanded target sets should be in the result set and their size should be minimum.

6. EVALUATION

Two performance measurements are adopted throughout the experiments: refinding response time and refinding quality.

1. Refinding response time: Refinding response time is the amount of time required by the system to produce results. Fig. 3 shows response time for 100 refinding requests in milliseconds. When users give more accurate information (i.e. accurate context), the machine will take less time to respond. Also. When user gives more number of context dimensions, it means that system can fetch the result using that.

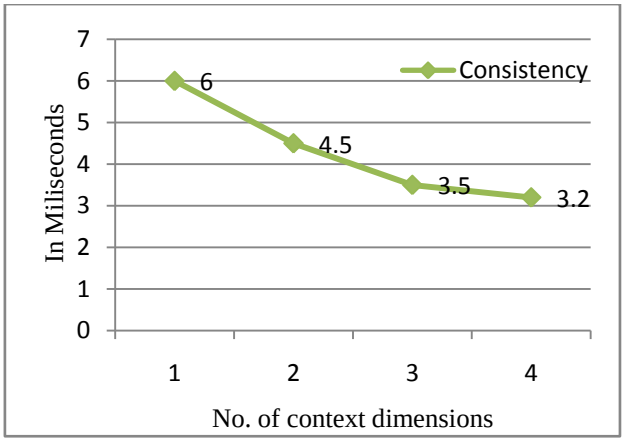

Figure 3. Average response time on synthetic data for n=1000

As shown in figure 3, when response time is taken into consideration, it is more when less number of context dimensions are selected. When user provides only one context dimension, system needs 6 milliseconds of response time. The time goes on decreasing with increasing number of context dimensions. When 2 context dimensions are provided, response time is 4.5 milliseconds. When user provides 4 context dimensions, response time reduces to 3.2 milliseconds.

2. Refinding Quality: It is based on refinding precision, recall and F-measure, where num_of_true_results_being _matched is the number of context instances which satisfy refinding request. No_of_matched_results is the result instances returned by system.

$$\text{Precision} = \frac{(\text{num\_of\_true\_results\_being\_matched})}{(\text{No\_of\_matched\_results})} \quad (1)$$

$$\text{Recall} = \frac{(\text{num\_of\_true\_results\_being\_matched})}{(\text{No\_of\_true\_results})} \quad (2)$$

$$\text{F-measure} = \frac{2.(\text{Precision}.\ \text{Recall})}{(\text{Precision}+\text{recall})} \quad (3)$$

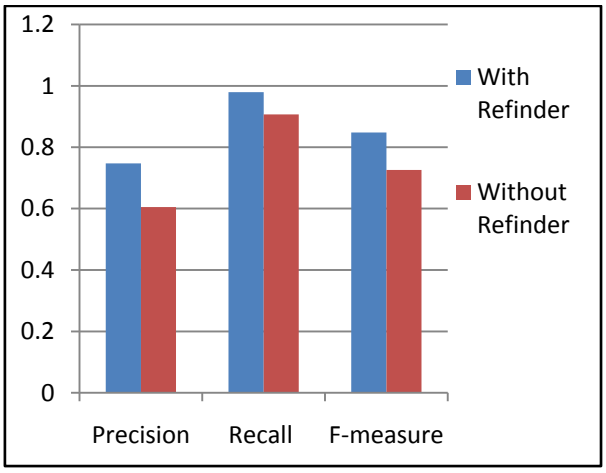

Figure 4. Refinding quality with and without Refinder

In the experiment, 100 refinding activities with ReFinder and 100 without ReFinder were examined. The average precision,

recall and f-measure values with and without ReFinder are as shown in table 1 and fig. 4. Precision without ReFinder is 0. 6051 and that with ReFinder is 0.7472. When it comes to recall, ReFinder gives an excellent value of 0.97 as compared to 0.9070 of that without ReFinder.

When Refinding files without ReFinder, the denominator is the number of browsed folders through file explorer and in case of web pages, the number of checked folders using bookmarks.

Table 1. Precision, recall and F-measure values with and Without ReFinder

	Precision	Recall	F-measure
With ReFinder	0.747265	0.979592	0.847794
Without ReFinder	0.605102	0.907041	0.725923

As shown in fig. 5, users needed 16.5 seconds to refind data using ReFinder and 86.32 seconds without using ReFinder. This included using bookmarks for web data and direct searching for data on system.

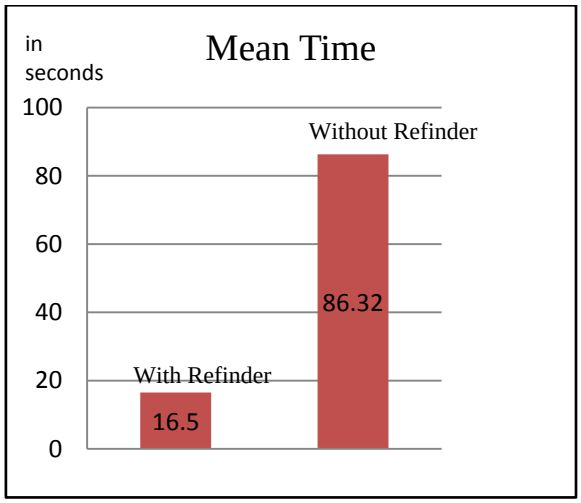

Figure 5. Mean time with and without Refinder

7. FUTURE WORK AND CONCLUSION

ReFinder refinds information based on a query-by-context model over a context memory snapshot. It links to the accessed information contents. Context instances in the memory snapshot are organized in a clustered and associated way, and dynamically evolve by degradation and reinforcement in life cycles.

Users access enormous web pages and files from their system. Compared to this, the number of data to be re-found is very less. Therefore it is uneconomical to store all the files in context memory. Humans tend to forget information stored long time ago. Using this phenomenon, degrading very old access context shrinks the search space. For this purpose, context degradation is done.

Results using ReFinder are better in both response time and quality point of view as compared to traditional methods for both web data and personal information management. Mean time for ReFinder was 16.5 seconds, whilst that for other methods (bookmarks and direct search) was found to be 86.52 seconds.

Values of precision, recall and F-measure are also better for ReFinder as shown in table 1.

Even though ReFinder's results are encouraging, there is scope for improvements in future work. Two of the issues are discussed below:

- Automatic Annotation: In ReFinder, users need to annotate the attributes manually but for many users this is a bit annoying to stop their work and annotate the information. To relieve users from this distraction, system can annotate the attributes according to users' history. The main challenge is to let the system identify which of the access context will be recalled later, according to users' interest. Analysis of user's access behavior, access history, accessed information, and user's activity could support decision making.
- Context degradation: In ReFinder, for degradation hierarchical approach is followed. But in reality, user's memorized contextual information may not decay strictly along such a hierarchy. There can be different degradation process for different information. Thus, the decay strategies for context memory should consider the specific characteristics of diverse contextual information.

8. REFERENCES

[1] R. Capra, M. Pinney, and M.A. Perez-Quinones, "Refinding Is Not Finding Again," technical report, Aug. 2005.

[2] S.K. Tyler and J. Teevan, "Large Scale Query Log Analysis of ReFinding," Proc. Third ACM Int'l Conf. Web Search and Data Mining (WSDM), 2010.

[3] J. Teevan, "The Re:Search Engine: Simultaneous Support for Finding and Re-Finding," Proc. 20th Ann. ACMSymp.User Interface Software and Technology

[4] M. Lamming and M. Flynn, "'Forget-Me-Not'-Intimate Computing in Support of Human Memory," Proc. FRIEND21 Int'l Symp. Next Generation Human Interface, 1994.

[5] E. Tulving, "What is Episodic Memory?" Current Directions in Psychological Science, vol. 2, no. 3, pp. 67-70, 1993.

[6] B. MacKay, M. Kellar, and C. Watters, "An Evaluation of Landmarks for Re-Finding Information on the Web," Proc. Extended Abstracts on Human Factors in Computing Systems (CHI '05 EA), 2005.

[7] D. Morris, M.R. Morris, and G. Venolia, "Searchbar: A Search- Centric Web History for Task Resumption and Information Re-Finding," Proc. SIGCHI Conf. Human Factors in Computing Systems (CHI), 2008.

[8] J.P. Dittrich and M.A. Salles, "iDM: A Unified and Versatile Data Model for Personal Dataspace Management," Proc. 32nd Int'l Conf Very Large Data Bases (VLDB), 2006.

[9] Deng et all, "ReFinder: A Context-Based Information Refinding System", IEEE Transactions on Knowledge and Data Engineering, vol. 25, no.9, September 2013