The Re:Search Engine: Simultaneous Support for Finding and Re-Finding

Jaime Teevan
Microsoft Research
Redmond, WA 98052 USA
Tel: 1-425-421-9299
teevan@microsoft.com

ABSTRACT
Re-finding, a common Web task, is difficult when previously viewed information is modified, moved, or removed. For example, if a person finds a good result using the query "breast cancer treatments", she expects to be able to use the same query to locate the same result again. While re-finding could be supported by caching the original list, caching precludes the discovery of new information, such as, in this case, new treatment options. People often use search engines to simultaneously find and re-find information. The *Re:Search Engine* is designed to support both behaviors in dynamic environments like the Web by preserving only the memorable aspects of a result list. A study of result list memory shows that people forget a lot. The Re:Search Engine takes advantage of these memory lapses to include new results where old results have been forgotten.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. – Graphical user interfaces; H5.4: Hypertext/Hypermedia. – User issues.

General terms: Design, Experimentation, Human Factors

Keywords: Re-finding, search, dynamic information

INTRODUCTION
A recent Pew Internet and American Life report found that search is a top Internet activity, second only to email [15]. Search engines are often used to re-find previously viewed information [25]. While many search engines have begun to support re-finding by, for example, caching query history, these efforts are just a beginning. Most search tools focus solely on the finding of new information. Support for re-finding is likely to significantly improve people's ability to find in general, and may reduce organizational overhead as re-finding replaces filing.

However, finding and re-finding can be in conflict; finding new information means retrieving information that has not been seen before, while re-finding requires retrieving information that has. For example, improving the search results for a query can help a searcher encounter new, more relevant results, but can also interfere with that person's ability to re-find previously viewed results that no longer appear where expected [14, 25]. Because people regularly find and re-find simultaneously [25], search tools need to seamlessly support both activities. Yet currently tools that address re-finding at all treat it in isolation.

The *Re:Search Engine* is a search system that addresses the conflicting goals of providing new information and facilitating the re-finding of old information. It consists of a Web browser toolbar plug-in that interfaces with a preexisting search engine, such as Live or Google. When a person issues a query to the Re:Search Engine that is similar to a previous search, the engine fetches the current results for the query from the preexisting search engine and fetches relevant previously viewed results from its cache. The newly available results are then merged with the previously viewed results to create a list that supports intuitive re-finding and contains new information.

An example is shown in Figure 1. When the search for "breast cancer treatments" is repeated, memorable results from the original list are preserved, while others are replaced by new, better results. As this paper will demonstrate, the merged list appears unchanged. The list supports re-finding as well as if it were the same, but is in fact fresh.

By replacing forgotten stale content with new content, the Re:Search Engine uses the searcher's limited memory to the searcher's advantage. Doing this requires understanding of which aspects of a search result list are memorable and thus might disorient the user if changed, and which are not and thus can change freely. Following a discussion of related work, this paper presents a study that reveals the memorable aspects of search result lists. This study informs the Re:Search Engine's architecture and function, which are then presented in detail. The paper concludes with an evaluation of the engine that finds it supports re-finding better than existing solutions while not interfering with the searcher's ability to find new information.

RELATED RESEARCH
Re-finding behavior has recently attracted considerable interest [2, 3, 4, 25]. Repeat searches appear to be very common as a way to revisit information [14]. Teevan et al. [25] found that 33% of all queries have been issued before by the same user. There have been a number of search tools developed to support re-finding [5, 9, 10]. However, these tools tend to ignore a common result of many be-

Figure 1. On the left is the result list originally returned by the Re:Search Engine for the query *breast cancer treatments*. On the right is the result list returned at a later date. It contains the results that the searcher remembers having seen before where she expects them, while still including the new results.

havioral studies of re-finding: the path taken to find information is very important when re-finding [4, 26]. Instead existing re-finding tools support a process of returning to information that can be very different than the process by which the information was originally encountered.

Because people return to information guided by their previous interactions, changes that should help can interfere. For example, dynamic menus were developed to help people access menu items more quickly than traditional menus by bubbling commonly accessed items to the top of the menu. Rather than decreasing access time, research revealed dynamic menus slow users down as commonly sought items no longer appear where expected [11, 22].

Problems resulting from change have been observed for search results as well [14]. In a study of search result stability, Selberg and Etzioni [19] noted that, "Unstable search engine results are counter-intuitive for the average user, leading to potential confusion and frustration when trying to reproduce the results of previous searches." Teevan et al. [25] demonstrated the veracity of this statement via large scale log analysis. They found that searchers took significantly longer to click on a repeat search result during a repeat query when the result list had changed. Another example of the difficulties caused by result list change can be found in a study by White, Ruthven, and Jose [29]. In this study, the authors tried to help people search by giving them lists of relevant sentences that were dynamically re-ranked based on implicit feedback gathered during the search. However, people did not enjoy the search experience as much or perform as well with the dynamic system as they did when the sentence list was static.

Information management systems that do preserve consistency of interaction despite change permit their users to choose to interact with a cached version of their information space [8, 16]. For example, Rekimoto [16] developed a system that allows people to use their desktop to "time travel" to specific information environments that existed in the past. However, operating within a static world denies users the opportunity to simultaneously discover new information. With such systems, searchers cannot, for example, revisit previously found information on breast cancer treatments while still learning about newly available treatments. Support for simultaneous finding and re-finding is important because the finding of new information while re-finding is common. Teevan et al. [25] found that 27% of repeat searches involve clicks on new results as well as previously clicked results.

The Re:Search Engine is a way for searchers to easily interact with old and new search results at the same time. Perceived consistency is maintained, so that result lists appear unchanged even though it includes new and potentially better results. The way this is done is modeled on the concept of *change blindness.* Change blindness is a visual phenomenon where obvious changes to a scene occur without the viewer's notice as a result of limitations on human memory capacity and attention [21]. As an example, the difference between the two photographs in Figure 2 is obvious when they are viewed side by side – one picture has a crosswalk and the other does not. But when the two pictures are flashed sequentially, separated by a small gap in time, most people cannot identify a difference – even when they actively look for a change.

Several researchers in human-computer interaction have explored how change blindness might affect users' ability to interact with computer-based information [6, 13, 28]. Their research, however, has focused on the fact that

Figure 2. A large change that can go unnoticed due to change blindness. Viewed side-by-side, it is obvious a crosswalk appears in one picture and not the other. But when flashed sequentially, most cannot identify the difference.

people may miss important changes due to change blindness, and the solutions presented try to draw users' attention to changes, rather than trying to take advantage of such holes in memory to present useful new information in an unnoticeable manner. In the research presented here, the changes to the search results in Figure 1 are intended to pass unnoticed like the changes to the picture in Figure 2.

Although the list returned by the Re:Search Engine may appear the same to the user, evaluation of the system demonstrates that the inclusion of new and better results can nonetheless help satisfy the user's information need. Usability improvements do not need to be noticed to benefit the user. A classic example of this is the Macintosh design for cascading submenus, where flexibility in navigating to menu items is built into the menu design. The tolerance for errors in navigation goes unnoticed by almost all users, but leads to fewer errors overall [27]. Similarly, a study of an improvement to cascading submenus showed all users performed better even though only three out of the 18 participants actually noticed any change [1].

Rather than hiding any changes made to a repeat result list, the Re:Search Engine could make changes explicit by, for example, highlighting old or new results. However, any alternative that presents both new and old information in a single list (whether it does so visibly or invisibly) faces the merging challenges that this work addresses.

RESEARCH USED TO BUILD THE RE:SEARCH ENGINE

The preceding discussion of related research helps to motivate the need for the Re:Search Engine and provides an overview of the principles it is built on. However, to actually construct a search system that takes advantage of a person's memory of previously viewed search results, it is necessary to understand what is memorable about result lists. A study was conducted to elicit this information. The results of the study are highlighted here because of their importance to the Re:Search Engine's design and architecture. Further details can be found in previous work [23].

In the study, 119 participants were asked to interact naturally with a list of results for a self-generated query. Queries were issued to a search engine via a Web form accessed from the participant's own computer, and clicked results were logged. An hour later, participants were emailed a survey that asked them to recall the result list without referring back to it. The survey asked participants to remember the text of their query, the number of results returned, and basic information about each result, including its rank, title, snippet, URL, whether the URL was clicked, and if so, what the corresponding Web page was like.

Because a result's recalled rank may not correspond to its true rank, the description of each recalled result had to be matched to one of the originally viewed results. Two independent coders performed this matching with an 84% inter-rater reliability. The 189 results that were described richly enough for both coders to make the same match were considered to have been "memorable". These memorable results were analyzed to provide insight into how to predict which results will be remembered, and how to understand the relative likelihood that various different types of changes that can occur in a result list will be noticed.

What Makes a Result Memorable

Participants recalled little about the result list that they originally saw. Although only a few hours elapsed between the first search and the follow-up survey, only 15% of all results displayed were memorable. Two main factors emerged from the data as affecting how likely a result was to be remembered: where in the result list it was ranked and whether or not the result was clicked.

Figure 3 shows the probability that a result was remembered given its rank for clicked results (solid line) and unclicked results (dashed line). The shape of the curves is similar to what has been observed in cognitive psychology literature [12]. Those results that are presented first are more memorable than later results and the results presented last are somewhat more memorable than earlier results.

Results that were clicked were significantly ($p<0.01$) more likely to be recalled. Forty percent of the time a result was clicked it was remembered, compared to only 8% of results

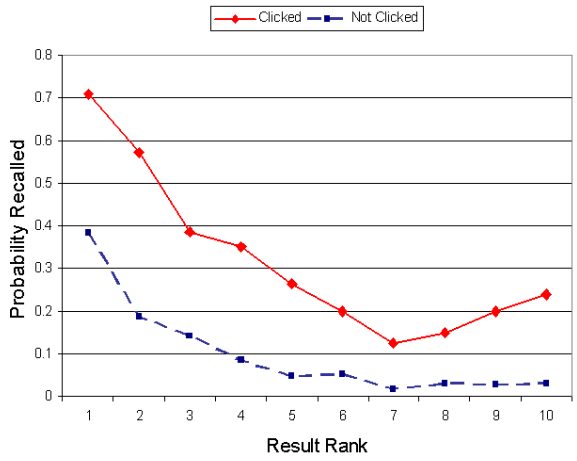

Figure 3. The probability of recalling a result given rank. The probability generally decreases as a function of rank. Clicked results were significantly more likely to be recalled (p<0.01).

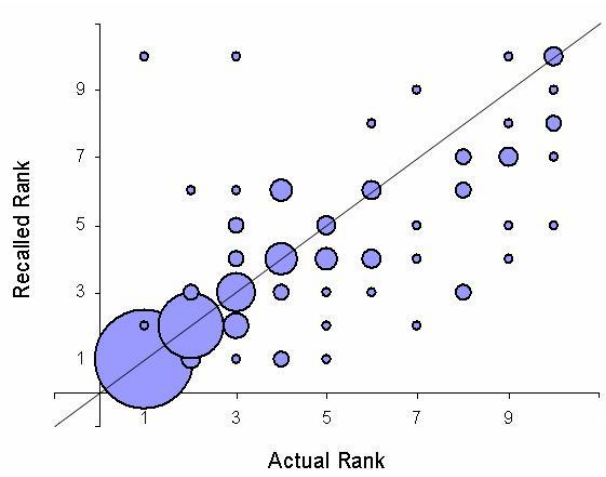

Figure 4. The result's location in the result list as the participant remembered it, compared with its actual location. Size represents the number of people remembering that combination.

that were not clicked. The last result clicked was particularly memorable, with a 12% increase in recall compared to other clicked results.

How Result Ordering Was Remembered

Subjects' memories of result ordering were also analyzed to understand how changes to ordering might affect their ability to interact with a search result list. Participants regularly made mistakes when recalling a result's rank. The recalled rank differed from actual rank 33% of the time. Mistakes were less common for early-ranked results. For example, the first result's rank was correctly recalled 90% of the time. Accuracy dropped as rank dropped. This can be seen graphically in Figure 4, which shows recalled rank as a function of actual rank. This trend suggests moving a result from the number one position in a result list is more likely to be noticed than moving a lower ranked result.

Figure 4 also illustrates another trend in the data. The greater weight of the data occurs to the right of the identity line. This means that remembered results were much more likely to be recalled as having been ranked higher than they actually were. Those results moved up in the result list 24% of the time, significantly more often than they moved down (10% of the time, p<0.01). The trend to remember results as highly ranked could reflect the fact that remembered results were more likely to be relevant to the participant's information need and thus in the participant's mind "should have been" ranked more highly than they were.

It is interesting to consider the ramifications of the fact that people misremember result ranking. It suggests that it may be possible for a result list to look *more* like the result list a person remembers having seen than the actual list they saw. In evaluations of the Re:Search Engine, there was a trend for the engine's results to be perceived as static more often than unchanged result lists. While these findings are not significant, they could suggest that the Re:Search Engine does a good job of placing results where they are expected – even when that is not where they originally occurred.

How the Query Was Remembered

Analyzing the difference between how participants remembered their query and the query they actually issued gives insight into the way people may express re-finding needs. A large scale log analysis of the differences between re-finding queries can be found in work by Teevan et al. [25]. In the study presented here the original query was misremembered 28% of the time. A majority of the differences between the original query and the remembered query fell under the following four categories (the percentage of misremembered queries for each category is in parentheses):

Capitalization (31%). A common change between the initial query and the remembered query was that the words in one query would be capitalized differently than they were in the other query. For example, the query "Buddha belly" became "Buddha Belly" an hour later.

Word form (28%). Another common change observed was in word form. For example, a query term that was originally listed as plural might be remembered as singular, as in the case where "sample television scripts" became "sample television script".

Word ordering (28%). The order individual terms occurred in a query also often changed. For example, the query "porsche 356" was remembered as "356 Porsche".

Phrasing (31%). The helper words used to place the primary query terms in context often varied. An example of this is that one participant originally queried, "I'm looking for a Burberry Scarf" but remembered the query as "Where can I find Burberry Scarves?" Unimportant terms like "for" and "where" are commonly referred to as stop words.

THE RE:SEARCH ENGINE ARCHITECTURE

The Re:Search Engine was designed to use the results of this study to preserve memorable aspects of old result lists when people appeared to be re-finding while also incorporating new results. The engine consists of a Web browser toolbar plug-in that interfaces with a preexisting search

engine (e.g., Live or Google). When a person issues a query to the Re:Search Engine that they have issued before, the engine fetches the current results for that query from the underlying search engine and merges any new information with what the user is likely to remember about the previously returned search results.

The architecture of the Re:Search Engine is shown in Figure 5. The system consists of four major components: an index of past queries that the user has issued, a result cache containing previously viewed results, a user interaction cache, and a merge component. The index of past queries is implemented as a hash table that maps query terms to queries. Similarly, the result cache maps queries to result lists, and the user interaction cache maps query/result pairs to interactions. The merge algorithm is what drives the engine, making use of these pieces to create a result list.

All of the data collected by the Re:Search Engine is stored locally on the user's machine. This has the disadvantage of tying the use of the Re:Search Engine to a particular machine, but such a design decision ensures that the relatively large amount of personal information that the Re:Search Engine stores will remain private.

HOW THE RE:SEARCH ENGINE FUNCTIONS

This section describes how each of the components of the Re:Search Engine work together to produce the search result list returned to the user. In order to identify relevant previously viewed results, the user's query is initially matched to an index of the past queries that the user has issued. The index returns queries that are similar to the one just issued and a match score for each representing how similar it is to the current query. Robust query matching is necessary because, as suggested by the earlier study, people do not always use exactly the same query when repeating a search. Matched queries are then used to retrieve the previously viewed results for each query from the result cache. This set of potentially memorable results, along with the live results for the current query from the underlying search engine, are merged together using the query match scores to weight how important each different result set is. The new query is added to the index of past queries and the merged result list is added to the result cache. Finally, the resulting list of search results is presented to the user, and the user's interactions with the list are logged. Each component is described in greater detail below.

Index of Past Queries

The index of past queries uses the current query and past queries to determine if the user intends to retrieve previously viewed information during the current search, and, if so, which past queries as associated with the current search. Once past relevant queries are gathered, the current query is added to the past query index for use in future searchers.

The index of past queries functions in a similar manner to a traditional document index used in information retrieval, except that the "documents" that are indexed are past query strings. Matching queries using an index deemphasizes the commonly misremembered query features described earlier. Query strings are tokenized, stemmed, changed to lower case, and stop words are removed. Each past query (*pq*) is given a score based on how closely it matches the current query (*cq*). The score (S_{pq}) is computed using a standard information retrieval scoring function known as *tf.idf* [18]:

$$S_{pq} = \sum_{t \text{ in } cq} pq_t \log(N/n_t)$$

where N is the number of past queries the user issued, and n_t is the number of past queries in which term t occurs. This scoring function reflects the fact that past queries that match on terms the user searches for rarely are more likely to mean the same thing than commonly used terms. The match score determines how much weight the results for each query carry in the merge process.

Earlier research has shown that not all queries with similar text are repeat queries [25]. For example, if a user is in the middle of a search session, it is likely that when a user issues several variants of the same query, that user actively wants to see new results with each variant. The results

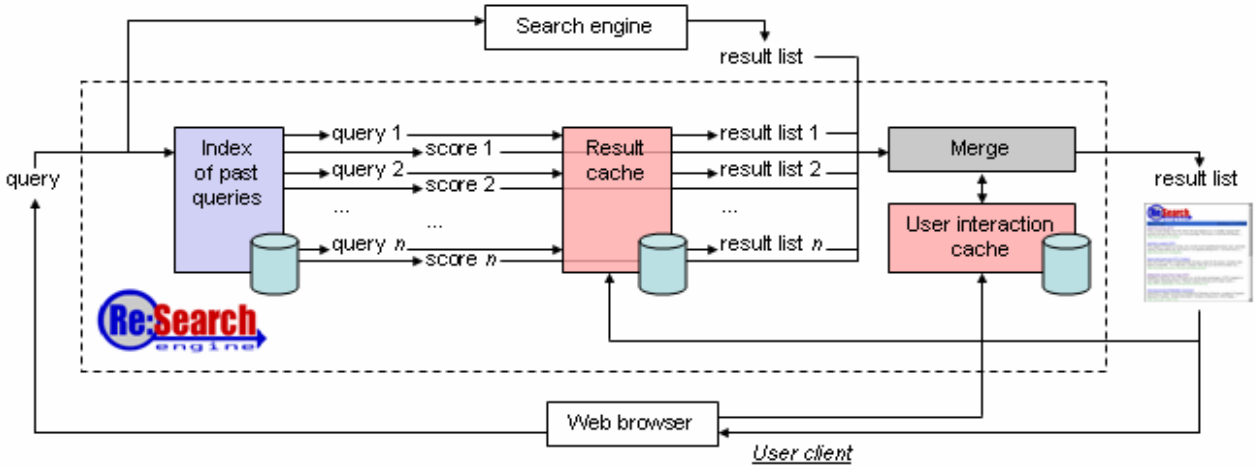

Figure 5. The architecture of the Re:Search Engine. The user's current query is matched to past queries, and the results for the past queries are retrieved from a cache. These results are then merged with the live search engine results based on how memorable the results are, and the resulting result list is presented to the user.

from query reformulations should not be merged into the results from the query issued immediately prior even when there is significant overlap in the query. For this reason, past queries that are similar but that occurred recently are ignored. In future sessions, any query from the session of overlapping queries may match. A future improvement may be to weight queries towards the end of a session more highly, on the assumption that these queries were more relevant to the user's goal.

While queries that occurred a long time ago may be repeat queries, it is also likely that the user has forgotten the results associated with the query. To account for progressive forgetting, the query's score is discounted by the log of the amount of elapsed time, a function commonly used in cognitive psychology literature to model human forgetfulness [17]. When enough time has passed that a user is likely to not even remember having issued the query before, the query and its associated results could be expunged from the system, relieving potential long term storage burdens. This functionality is not currently implemented.

Although the index of past queries permits flexible query matching, the Re:Search Engine's interface is designed to encourage users to communicate re-finding intent by encouraging them to exactly duplicate previously issued queries. Existing query histories can be difficult to use [14]. The index of past queries is used to support sophisticated query completion in the search box. Past queries that match the query being typed in are suggested, weighted by the query's score. Thus if the searcher who had previously searched for "breast cancer treatments" (e.g., Figure 1) began typing, "cance..," into the search box, her previous query for will be suggested.

Result Cache

If the query the user issued is determined to be related to one or more previous searches, the results corresponding to the previous searches are fetched from a result cache using the pervious queries returned by the past query index. The result cache is a straightforward cache that maps an exact query string to the search results list presented to the user for that query. Only the most recently viewed set of results for a particular query is stored in the cache. For example, when the query "breast cancer treatments" was issued a second time in Figure 1, the merged results shown on the right replaced the old results in her result cache.

User Interaction Cache

Once past results that might be relevant to the user's current query are fetched, they are merged with the live search results to produce a result list consisting of old and new information to return to the user. Because the merge algorithm is designed to help users take advantage of what they learned about the current query during past searches, it needs to estimate how likely the past results the user interacted with are to be memorable. The user's browser is instrumented to gather implicit information about the user's interactions with previously viewed results. This information is stored in the user interaction cache.

Currently the user interaction cache only records the results that the user clicks on. But there are many other possible implicit cues that could use to understand which results are memorable to the user. Possible cues worth investigating include dwell time on the result's Web page, the number of times a particular result is accessed, and more sophisticated measures such as mouse tracking or eye tracking. Additionally, active information could be gathered. For example, the system could easily be extended to allow users to mark results that they believe are worth remembering.

Merge Algorithm

This paper has argued that for a result list to be useful for re-finding, it must preserve the results the searcher remembers having seen during earlier searches where the searcher expects to see them. New results can be added where old results have been forgotten. To merge a new result list with old result lists, the value of the new information presented needs to be balanced with the cognitive cost of presenting old information in unexpected ways.

The quality of a result list is a function of how much high quality new information is included in the list, ranked so that the new information will be seen, and how closely the list matches the user's memory. By iterating over all possible lists, the highest quality list l can be chosen to return.

$$\underset{\text{possible lists } l}{\operatorname{argmax}} \sum_{r=1}^{10} B(l(r), \mathrm{r}) + M(l(r), \mathrm{r})$$

The function B returns the *benefit of new information* provided by a result in the list l when shown at position r. The function M returns the *memorability* of the result is when shown at position r. Each of these two functions is described in greater detail below, followed by a discussion of how the best possible list can be chosen efficiently.

Benefit of New Information (B)

The most relevant new results for a query need to be identified for inclusion in the returned result list. New results are found by running the query on an underlying search engine such as Live or Google. The Re:Search Engine uses a result's rank in the underlying engine's result list as a proxy for relevance to calculate the potential benefit of the new result. Scoring information could be used if available.

The expected benefit that a new result will provide in the returned result list is also a function of how likely it is to be encountered. The closer a result is ranked to the top of the returned result list, the greater the benefit it provides.

The *benefit of new information* (B) is defined to be zero if the result is not in the list currently returned by the underlying search engine. Otherwise, it is:

$$B(i, r) = (11\text{-}r_n(i))\ (10+(11\text{-}r))$$

where $r_n(i)$ is the rank of the new result i in the result list returned by the underlying search engine. Thus results that rank highly in the underlying engine's result list are more beneficial when they occur anywhere in the merged list than results that are ranked later by the underlying engine.

Note that in the current implementation, results that occur in the underlying result list but that were seen before contribute a non-zero benefit of new information and a non-zero memorability score. This reflects the fact that these results are likely to be both relevant to the current information need and memorable. However, an alternative approach that places a higher value on the inclusion of new information – at the risk of including information that is currently judged less relevant – would be to filter previously viewed results from the new result list and only assign a benefit of new information score to un-viewed results.

Memorability (M)

Value is assigned to how memorable a result is using the two main factors identified earlier: where the result was ranked and whether it was clicked. The probability of a result being recalled can be modeled as $\Pr(\text{recall}(i)|r_o(i), c(i))$, where $r_o(i)$ is the previous rank of the result, and $c(i)$ is whether the link was clicked or not. A smoothed version of the results shown in Figure 3 is used in the calculation of this probability. The probability of remembering a result that has not been seen before is zero.

The value of preserving a previously viewed result in the final result list is a function not only of how memorable that result is but also of how likely it is to appear where the user remembers having seen it. For this reason, memorable results ranked near where they were originally ranked receive higher memorability scores than others. The value of a result being remembered at a particular rank is calculated using a smoothed version of the empirical probabilities of a particular rank being recalled at a different rank, $\Pr(\text{recall}(r)|r_o(i))$, shown in Figure 4. Thus *memorability* (M) is computed as follows:

$$M(i, r) = \Pr(\text{recall}(i)|r_o(i), c(i)) \Pr(\text{recall}(r)|r_o(i))$$

Because the results clicked last during earlier searches are empirically more memorable, those results are given a corresponding boost in memorability. A result's memorability is also weighted by the match score of the query associated with it, since queries that do not match the current query very well are unlikely to have returned results that the user finds memorable during their current search task.

Choosing the Best Possible List

During the merge process, all permutations of possible final lists that include at least a few old results and a few new results are considered, and the result list with the highest total benefit of new information memorability is selected. There is obviously a trade-off between preserving a lot of the information seen during previous queries and presenting as much new information as possible. Requiring that both old and new results be included in the final list ensures that some context is maintained while not allowing the list to stagnate. While the minimum number of results preserved and added could be determined dynamically as a function of how likely the query is to be a re-finding query, the value is currently set to three for each case.

Although considering all permutations of possible result lists naively is expensive, the merge algorithm can be implemented efficiently by representing the problem as a min-cost network flow problem [7]. The complexity is, in practice, $O(m)$, and the implementation runs in 180 milliseconds (including Java startup time) on a standard machine. This performance is achieved by representing the list selection problem as the network shown in Figure 6.

Ten units of flow are sent through the graph, each unit representing one result in the final result list. Seven units are passed to nodes representing the new results, and seven are passed to nodes representing the old results. This ensures that at least three units must pass through the old results and at least three through the new results. The nodes representing new results are connected to the ten slots representing the result list with unit capacity edges that have costs inversely proportional to each result's benefit of new information. The nodes representing old results are similarly connected to the ten result lists slots with unit capacity edges that have costs inversely proportional to each result's memorability. All other edges have zero cost.

The best list is found by finding the maximum flow through the graph with the minimum cost. Because only one unit of flow can travel from each result slot to the sink, only one unit of flow can travel into each slot. The candidate node from which that unit of flow arrives represents the result that should be ranked in that slot's position.

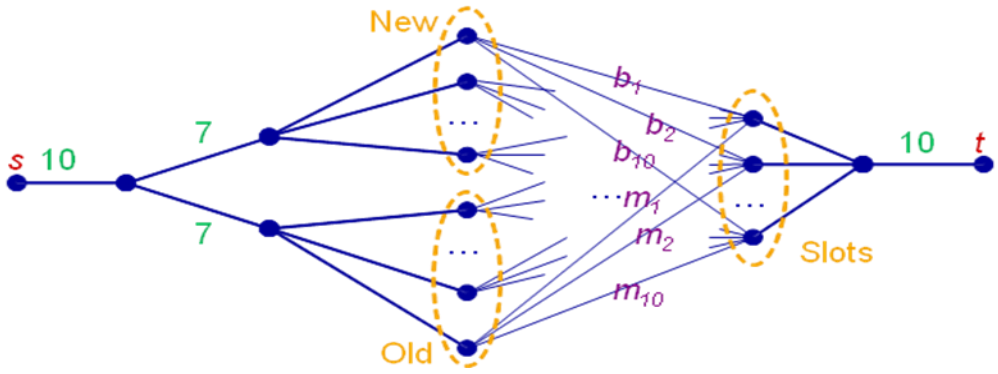

Figure 6. Graph representation of the merge algorithm. All edges have unit flow, except edges labeled in green. All edges have zero cost, except edges connecting the nodes representing the new and old results to the slots.

Table 1. The rank of new results and results from the original result list after merging.

***Merged* Rank**	**Results clicked in original result list:**		
	None	9	1, 2, 6, 8
1	Old result 1	Old result 1	Old result 1
2	Old result 2	Old result 2	Old result 2
3	Old result 3	Old result 3	Old result 3
4	Old result 4	*New result 1*	*New result 1*
5	*New result 1*	*New result 2*	*New result 2*
6	*New result 2*	*New result 3*	Old result 6
7	*New result 3*	Old result 9	Old result 8
8	*New result 4*	*New result 4*	*New result 3*
9	*New result 5*	*New result 5*	*New result 4*
10	*New result 6*	*New result 6*	*New result 5*

The cost of change and the benefit of new information can be weighted to express the relative value of new and old information. . For example, if the benefit of new information score for each result is much higher than the memorability score of each old result, then only the minimum three required results will come from previously viewed result list. The emphasis placed on each class of information should be a function of the individual using the Re:Search Engine, the elapsed time since the original list was seen, and the engine's certainty that the person wants new information versus old information. In the implementation tested, when no results were clicked the merging produced a list that began with four old results and ended with six new results. When low ranked results from the original result list were clicked, the clicked results were preserved in the new merged result list while higher ranked previously viewed results were dropped. Several examples of merged lists are shown in Table 1.

For simplicity, users are assumed to remember perfectly which result page a result occurred on (e.g., whether the result occurred in the top ten, or in results 11-20). Because the results for a query are never expected on a different result page than where they were seen, each old result page can be treated independently of other result pages during the merge. The highest ranking new information available is always is merged in, regardless of what particular page is requested. Although it is very likely that people do not really accurately remember which page a result for a query occurred on, in practice so few people visit subsequent result pages [20] that supporting the movement of results across result pages may not be worth additional overhead.

UNDERSTANDING THE RE:SEARCH ENGINE

Understanding how well the merge algorithms functions is essential to understanding how well the Re:Search Engine can support the simultaneous finding and re-finding of information. It is not obvious that people can use a changed result list to re-find, even when the memorable aspects are preserved. Nor is it obvious that new results will be useful for the finding of new information when they are hidden in the result list. For this reason, the merge process is explored in greater depth here. A longitudinal study of how the Re:Search Engine is used and how it affects finding and re-finding behavior in the long run stands as future work.

The merge algorithm was evaluated by comparing how it ranked results with three other possible ways to merge old results with new. The Re:Search Engine's merge will be referred to as:

> ***Intelligent Merge***. Results are merged according to the Re:Search Engine's intelligent merging algorithm so that memorable aspects of the original list are maintained. On average, four new results are included.

The other three other possible merging are:

1. ***Dumb Merge***. Six old results are randomly maintained and the top four new results included in random places.
2. ***Original***. No merging is done. The new result list is exactly the same as the old list. This is what a user of a system that caches previous results would see.
3. ***New***. The list is comprised of entirely new results.

To reflect the desired usage scenario where the new information to be included is more relevant to the user's needs, the results in the original list were chosen to be less relevant than the new result list, as indicated by search engine rank. The *Original* list consisted of results 11 through 20, and the *New* list consisted of results 1 through 10.

Two studies were conducted to compare these different merge types. The first study (Study I) establishes the ability of the Re:Search Engine to invisibly include new information in result lists. The second study (Study II) demonstrates that the research engine supports re-finding as well as if the result list never changed, while still supporting the finding of new information almost as well as if the list contained only new information.

Study I: New Information Can Be Included Invisibly

The ability of the Re:Search Engine to invisibly include new results in a list was studied by looking at how well 132 people could recognize a result list as being one they had seen before. As with the study presented earlier in this paper used to elicit the memorable aspects of result lists, participants were first asked to run a query of their choosing and interact with the results as they normally would. Also similar to the previous study, participants were asked about that search an hour later. During the follow-up session, however, instead of being asked to *recall* information about the result list, participants were asked to *recognize* whether a result list was the same or different from what they originally saw. The study was between-subjects; each subject was asked about only one of the four possible lists.

Differences were noticed most often for the two cases where new information was included in the follow-up list without consideration of what the searcher found memorable. When the follow-up results list was comprised of entirely new results (*New*), participants reported the list had changed 81% of the time. When four random results were held constant (*Dumb Merge*), the change to the remaining six results was noticed 62% of the time. The difference between the two cases was not significant.

Table 2. The time it took participants to complete the tasks in the second session.

Task Type	List Type Used in Second Session	Task Time (seconds)	
		Mean	Median
New-Finding	*Dumb Merge*	153.8	115.5
	Intelligent Merge	120.5	85.5
	New	139.3	92
Re-finding	*Dumb Merge*	70.9	37.5
	Intelligent Merge	45.6	23
	Original	38.7	26

The remaining two cases (*Original* and *Intelligent Merge*), represent instances where information from the original result list that might be memorable to the participant was not permitted to change – in the former case to the point of not including any additional new information. Even when the result list did not change at all, participants sometimes believed a change had occurred (31% of the time). In fact, participants were more likely to believe the result list had changed when all results were the same than for the *Intelligent Merge* case, where differences were noted only 19% of the time. This disparity is not significant, but as mentioned earlier could reflect the fact that the intelligently merged list may actually look more like the list the participant remembers than the actual original result list. While there was no significant difference between the two, the result lists from both the *Intelligent Merge* and *Original* cases were significantly more likely to be considered the same as the original list than the other two cases ($p<0.01$).

Study II: Even Invisible New Information is Useful

Although the previous study reveals it is possible to invisibly include new results in a result list, it is not clear from the study that invisible results are actually useful for finding new information, nor that a list that appears unchanged is useful for re-finding previously viewed information. To test the value of invisible new information, a second more controlled study was conducted. Like the previously described studies, the study involved two parts: 1) An initial session where participants conducted finding tasks, and 2) A follow-up session where participants conducted finding and re-finding tasks using the four different list types.

Unlike the previous between-subject studies, the study design was within-subject. Each of the 42 participants in this study conducted 12 search tasks during the initial session and 12 search tasks during the follow-up session. Six of the follow-up tasks were re-finding tasks, and six were new-finding tasks. For re-finding tasks, participants were given either the original list or one of the two merged lists. For new-finding tasks, they were given either the new list or one of the two merged lists. For each task, timing and interaction information was logged and questionnaire data was elicited. By comparing how well participants performed during the second session with how they performed during first, it is possible to understand the value of information re-use across sessions. A more detailed description of the study can be found elsewhere [24].

Table 2 presents how long it took participants to perform new-finding and re-finding tasks, broken down by list type. Task completion time can be a proxy for ease, and is one of several measures that showed a similar trend. The results reveal that the Re:Search Engine's intelligent merging of new information makes re-finding virtually as easy as if the results had not changed at all. Although the amount of time taken to re-find was the lowest when a static result list was used, there was no significant difference in re-finding time when new results were merged in intelligently. On the other hand, re-finding was significantly faster than the *Dumb Merge* for both the *Intelligent Merge* ($p<0.05$) and the *Original* list ($p<0.01$).

For new-finding tasks, the *Intelligent Merge* used by the Re:Search Engine was weakly significantly faster ($p<0.05$) than the random *Dumb Merge*. The *Intelligent Merge* also supported the finding of new information more quickly than a list of entirely new information, but the difference was not significant. However, the trend could suggest that people posses some ability to re-use knowledge even when finding new information – perhaps, for example, participants were able to skip over memorable old results.

Given these findings, the Re:Search Engine's intelligent merging seems to be the best compromise to support both finding and re-finding. A static, unchanging result list works well for re-finding but does not support the finding of new information. In contrast, a result list with new information works well to support the finding of new information, but does not support re-finding well. The intelligent merging performs closely to the best of both in both cases, while the dumb merging does comparatively worse.

CONCLUSION AND FUTURE WORK

This paper presented the Re:Search Engine, a search tool designed to support simultaneous information finding and re-finding. Currently re-finding is made difficult because when people issue repeat queries they receive new results. While results may be more relevant ignoring prior context, they are not necessarily more relevant to the re-finding task. Although the ability to find new information may appear at odds with the ability to re-find, the Re:Search Engine resolves this conflict by including new results where changes to the result list will not be noticed. This allow people to find new information as easily as if they were given all new information, while still allowing people to re-find information as easily as if nothing had changed.

In its current implementation, the new information to be included in Re:Search is assumed to become available as a result of natural changes in the results returned by the underlying search engine. Web search results can change over time as new information is indexed or as the search algorithms are updated. As search engines begin to support personalization based on their users' ever-changing context, the rate of change to result lists is likely to increase. Additionally, new information could also be proactively included in search result lists, at the expense of potential

relevance, to increase the diversity of information the searcher is exposed to.

The current implementation of the Re:Search Engine also assumes a significant period of time passes between repeat visits to search result lists. It will be interesting, however, to explore how new results can be snuck into lists that are actively being used, much as was done during the initial paper prototype. This would allow search engines to improve results using real time implicit relevance feedback without disrupting the user's search. Research into this domain is currently under way.

Effectively supporting expectation is essential to successfully supporting people's complex finding behavior. This is particularly true as the growing ease of electronic communication and collaboration, the rising availability of time dependent information, and the introduction of automated agents, suggest information is becoming ever more dynamic. Even traditionally static information like a directory listing on a personal computer has begun to become dynamic; Apple, for example, has introduced "smart folders" that base their content on queries and change as new information becomes available. As Levy [10] observed, "[P]art of the social and technical work in the decades ahead will be to figure out how to provide the appropriate measure of fixity in the digital domain." The solution presented here is a good first step towards that end.

ACKNOWLEDGEMENTS

This research owes much to valuable discussions with David Karger, Sue Dumais, Mark Ackerman, and Rob Miller.

REFERENCES

1. Ahlström, D. (2005). Modeling and improving selection in cascading pull-down menus using Fitts' law, the steering law and force fields. In *Proceedings of CHI '05*, 61-70.
2. Aula, A., Jhaveri, N., and Käki, M. (2005). Information search and re-access strategies of experienced Web users. In *Proceedings of WWW '05*, 583-592.
3. Bruce, H., Jones, W. and Dumais, S. (2004). Keeping and refinding information on the Web: What do people do and what do they need? In *Proceedings of ASIST '04*.
4. Capra, R. and Pérez-Quiñones, M.A. (2005). Using Web search engines to find and refind information. *IEEE Computer*, 38 (10), 36-42.
5. Dumais, S. T., Cutrell, E., Cadiz, J. J., Jancke, G., Sarin, R. and Robbins, D. C. (2003). Stuff I've Seen: A system for personal information retrieval and re-use. In *Proceedings of SIGIR '03*, 72-79.
6. Durlach, P. J. (2004). Change blindness and its implications for complex monitoring and control systems design and operator training. *Human-Computer Interaction*, 19(4): 423-451.
7. Goldberg, A.V. (1997). An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22(1): 1-29.
8. Hayashi, K., Nomura, T., Hazama, T., Takeoka, M., Hashimoto, S., and Gudmundson, S. (1998). Temporally-threaded workspace: A model for providing activity-based perspectives on document spaces. In *Proceeding of HyperText '98*.
9. Komlodi, A., Soergel, D., and Marhionini, G. (2006). Search histories for user support in user interfaces. *JASIST*, 57(6): 803-807.
10. Levy, D. (1994). Fixed or fluid? Document stability and new media. In *Proceedings of European Conference on Hypertext*.
11. Mitchell, J. and Shneiderman, B. (1989). Dynamic versus static menus: An exploratory comparison. *ACM SIGCHI Bulletin*, 20(4): 33-37.
12. Murdock, B. B. (1962). The Serial Position Effect of free recall. *Journal of Experimental Psychology*, 64, 482-488.
13. Nowell, L., Hetzler, E., and Tanasse, T. (2001). Change blindness in information visualization: A case study. In *Proceedings of INFOVIS* '01, 15-22.
14. Obendorf, H., Weinreich, H., Herder, E., and Mayer, M. (2007). Web page revisitation revisited: Implications of a long-term click-stream study of browser usage. In *Proceedings of CHI '07*, 597-606.
15. Rainie, L. and Shermak, J. (2005). Pew Internet and American Life Project: Data memo on search engine use. Retrieved January, 2006 from http://www.pewinternet.org/pdfs/PIP_SearchData_1105.pdf.
16. Rekimoto, J. (1999). Time-machine computing: A time-centric approach for the information environment. In *Proceedings of UIST '99*, 45-54.
17. Rubin, R. C. and Wenzel, A. E. (1996). 100 years of forgetting: A quantitative description of Retention. *Psychological Review*, 103, 734-760.
18. Salton, G. (1998). Automatic text indexing using complex identifiers. In *Proceedings of the ACM conference on Document processing systems*, 135-144.
19. Selberg, E. and Etzioni, O. (2000). On the instability of Web search engines. In *Proceedings of RIAO '00*.
20. Silverstein, C., Marais, H., Henzinger, M., and Moricz, M. (1999). Analysis of a very large Web search engine query log. *ACM SIGIR Forum*, 33(1): 6-12.
21. Simons, D. J. and Rensink, R. A. (2005). Change blindness: Past, present, and future. *Trends in Cognitive Sciences*, 9(1):16-20.
22. Somberg, B. L. (1986). A comparison of rule-based and positionally constant arrangements of computer menu items. In *Proceedings of CHI/GI '86*, 255-260.
23. Teevan, J. (2006). How people recall search result lists. In *Proceedings of CHI '06.*
24. Teevan, J. (2007). Supporting finding and re-finding through personalization. Doctoral thesis, Massachusetts Institute of Technology.
25. Teevan, J., Adar, E., Jones, R., and Potts, M. (2005). History repeats itself: Repeat queries in Yahoo's query logs. In *Proceedings of SIGIR '06*, 703-704.
26. Teevan, J., Alvarado, C., Ackerman, M. S., and Karger, D. R. (2004). The perfect search engine is not enough: A study of orienteering behavior in directed search. In *Proceedings of CHI '04*, 415-422.
27. Tognazzini (1999). A quiz designed to give you Fitts. http://asktog.com/columns/022DesignedToGiveFitts.html
28. Varakin, D. A., Levin, D. T., and Fidler, R. (2004). Unseen and unaware: Implications of recent research on failures of visual awareness for human-computer interface design. *Human-Computer Interaction*, 19(4): 389-422.
29. White, R., Ruthven, I., and Jose, J.M. (2002). Finding relevant documents using top ranking sentences: An evaluation of two alternative schemes. In *Proceedings of SIGIR '02*, 57-64.