AccessRank: Predicting What Users Will Do Next

Stephen Fitchett
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
saf75@cosc.canterbury.ac.nz

Andy Cockburn
Department of Computer Science
University of Canterbury
Christchurch, New Zealand
andy@cosc.canterbury.ac.nz

ABSTRACT
We introduce AccessRank, an algorithm that predicts revisitations and reuse in many contexts, such as file accesses, website visits, window switches, and command lines. AccessRank uses many sources of input to generate its predictions, including recency, frequency, temporal clustering, and time of day. Simulations based on log records of real user interaction across a diverse range of applications show that AccessRank more accurately predicts upcoming accesses than other algorithms. The prediction lists generated by AccessRank are also shown to be more stable than other algorithms that have good predictive capability, which can be important for usability when items are presented in lists as users can rely on their spatial memory for target location. Finally, we present examples of how real world applications might use AccessRank.

Author Keywords
Revisitation; prediction; list stability

ACM Classification Keywords
H5.2 User Interfaces: Theory and methods

INTRODUCTION
Many forms of interaction with computer systems are repetitive – we use the same commands [7], visit the same websites [14], return to previously visited document regions [1], and so on. To improve the efficiency of accessing previously used items, many diverse interactive techniques and systems have been developed, with examples including command histories [8], web page recency lists [9], scrollbar marks showing previous areas of document use [1], and menu adaptations that emphasise probable upcoming selections [4, 5, 2].

While there are plentiful examples of research and commercial systems that provide support for retrieving previously used data, there is much less on the design and evaluation of the underlying algorithms that support the predictions presented to users (examples of relevant algorithmic work are presented in the paper). Improving the performance of these algorithms would have a strong impact on many areas of interaction. For example, Firefox's *AwesomeBar* uses the 'Places Frecency' algorithm [12] to populate recommendations in a drop-down list alongside its URL bar, and membership of this list is progressively pruned and presented to users as they type characters, allowing predicted items to be rapidly selected.

This paper first describes several predictive algorithms from diverse areas of computer science that can be used to predict upcoming user actions. We then explain the key objectives for this type of algorithm – to predict accurately, and to provide stability so that users can anticipate where items will be located in interfaces that provide access to the algorithm's predictions. We explain how AccessRank combines two previous algorithms and adds new components to incorporate time of day information and to enhance stability. Then, we compare AccessRank's performance with that of many pre-existing algorithms by running simulations using three sets of log-file data extracted from real user activities: command use, web navigation, and window switching. Finally, we discuss issues with deploying AccessRank in real interfaces.

PREVIOUS WORK
Several areas of work influence our research, including general information retrieval, recommender systems, search, and caching algorithms. For brevity, we focus on summarising the algorithms that can be adapted for predicting upcoming actions based on previous actions, as follows:

Most Recently Used (MRU) and *Most Frequently Used* (MFU) calculate ranks based solely on recency or frequency, respectively. *Split Recency and Frequency* (SR&F) [4] select n items with MRU, then the rest with MFU.

Combined Recency and Frequency (CRF) [10], used originally for cache management, considers every past access of an item. It is calculated by Equation 1, where w_f is the item's weighting, n is the number of past accesses, t is the current time and t_i is the time of access i (where time is in terms of discrete events). In our testing $p = 2$ and $\lambda = 0.1$ performed best.

$$w_f = \sum_{i=1}^{n} \frac{1}{p}^{\lambda(t-t_i)} \quad (1)$$

The *Adaptive* algorithm filters menus in software such as Microsoft Office 2000 [2]. Item counts are incremented when selected and decremented after multiple sessions of disuse.

The *Places Frecency* algorithm (PF) [12] is used in Firefox to order URL suggestions when typing a web address. The last ten accesses of each item are placed in time-based buckets with different weights based on recency. Other factors, such

CHI'12, May 5–10, 2012, Austin, Texas, USA.

as the method of website access, are also incorporated but can be stripped out for general purpose use.

A *Markov* chain [11] can be used to make predictions, where

$$P(X_{n+1} = x | X_n = x_n) = \frac{|x_n \rightarrow x|}{|x_n|} \quad (2)$$

Here, $|x_n|$ is the number of previous occurrences of state x_n, and $|x_n \rightarrow x|$ is the number of previous transitions from state x_n to x. X_i represents the state at time i. Given the most recent access x_n, the calculated probabilities provide a ranking, and MRU can be used to break ties.

ACCESSRANK

AccessRank goals are twofold: first, to accurately predict the next action based on past ones; and second, to maximise list stability. The importance of prediction accuracy is obvious, but the need for stability is also important because it allows users to learn item locations over time, facilitating expertise with the interface used for list presentation [3].

AccessRank has three components, described below: *AccessRank Score*, which combines the stability of *CRF* with the accuracy of *Markov*; *Time Weighting*, which weights items based on the current time and day; and *Switching Threshold*, which improves stability. Computing AccessRank is fast, as its model can be updated in $O(1)$ time, all scores can be updated in $O(n)$ time, and predictions made in $O(n \log n)$ time.

AccessRank Score. A raw AccessRank score w_n is calculated for each previously accessed item using Equation 3. w_{m_n} is the Markov weight, w_{crf_n} is the CRF weight with $p = 2$ and $\lambda = 0.1$ and w_t is a time weighting. The parameter $\alpha > 0$ can be adjusted based on whether accuracy or stability is more important and determines the blend between the Markov and CRF algorithms.

$$w_n = {w_{m_n}}^{\alpha} w_{crf_n}{}^{\frac{1}{\alpha}} w_{t_n} \quad (3)$$

The Markov weight is altered to always give non-zero weights:

$$w_{m_n} = \frac{|x_n \rightarrow x| + 1}{|x_n| + 1} \quad (4)$$

Time Weighting. The time weighting w_{t_n} gives higher weighting to items that have historically been more frequently accessed at the current time of day or day of week. From informal observations, we have observed that many aspects of interaction are temporally predictable, for example habitually accessing a news webpage on arrival at work. Let c_h be the current hour of the day. For item n, let h be the ratio of the number of previous accesses of n in hours in the range $[c_h - 1, c_h + 1]$ compared to the average number of previous accesses of n for a three hour slot. Similarly, let d be the ratio of the number of previous accesses of n on the current day of the week to the average across all days of the week. h and d are set to one if fewer than 10 accesses in total have occurred in the corresponding slot. The time weighting is then calculated as in equation 5.

$$w_{t_n} = max(0.8, min(1.25, hd))^{0.25} \quad (5)$$

Switching Threshold. The switching threshold improves stability. Consider items A and B at positions $r_A < r_B$ in the previous prediction list. Pairwise comparisons between item weights are made during sorting to generate a new prediction list. If A and B are compared and their new weights w_A and w_B are such that $w_B > w_A$, then B will only be ranked higher than A if $w_B > w_A + \delta$, where $\delta \geq 0$ is an AccessRank parameter. An item k not in the previous list is assumed to have $r_k = \infty$. This comparison is not transitive, however it is deterministic with a given sorting algorithm. In our implementation we used merge sort.

LOG-BASED ANALYSIS OF ALGORITHM PERFORMANCE

We compared performance of these algorithms by analysing their predictive capability and stability for each successive action by each of many users recorded in several log datasets. The log datasets were collected from previously published studies conducted by other researchers, summarised in Table 1. Revisitation rates in the datasets ranged from 26% for URLs (from [14]) to 93% for commands (from [6]).

Study	Participants	Duration
Window switching [13]	25	3 weeks
Web browsing [14]	28 (experienced)	5-6 weeks
Command line use [6]	168 (4 exp. levels)	4 months

Table 1: Log datasets used in our analysis.

The log files were first converted to a standardised format, consisting of a series of *Visit*, *Addition* and, in some domains, *Removal* events. Then, simulations were run over each file, with each algorithm generating a prediction list prior to each *Visit* event, and the other events used to update the set of possible items to revisit. The prediction list was then compared with the actual logged *Visit* event. Stability was also continually calculated.

For the window switching logs, only user generated switches sustained for more than a second were considered, and we processed the raw logs as both window and application switching. We process the web browsing logs with full URLs and with domains only, where consecutive accesses on the same domain were collated into one visit. The command-line logs were processed as logs of full command lines, and as logs of just the first command from each line.

Measures

We used two types of measures to assess the algorithms' performance: *accuracy measures*, which assess prediction accuracy; and *stability measures*, which assess prediction lists variability over time.

Accuracy Measures. These include the *Average Rank* of revisitations in prediction lists, and the percentage of correct predictions (*Percentage Revisitations Predicted*). We also analyse how the latter is affected by accepting matches in the top k predicted items rather than just the top item, and by providing fixed length typed prefix 'hints' of a series of characters to the algorithm, which is common in interfaces such as Firefox's *AwesomeBar*: for example, cnn.com might become the first recommendation after typing 'c'.

Stability Measures. In their analysis of stability measures, Webber *et. al* [15] state that stability measures can be categorised based on two main properties: whether they are unweighted or top-weighted, and whether they require conjoint rankings. *Weightedness* refers to measures that give greater importance to items at certain ranks (e.g., items near the top of the list). *Conjointness* refers to whether or not two lists contain the same items (regardless of order): they are conjoint when membership is the same, and non-conjoint otherwise. We are most interested in top-weighted non-conjoint measures, as people tend to look at the top of a list, and the set of items being ranked changes over time.

We consider three stability measures: *Average overlap* (AO) and *rank-biased overlap* (RBO) (both described in [15]), and *Learnability* [3]. AO and RBO are both top-weighted non-conjoint measures. If comparing just the top k items of each list, AO is non-convergent as $k \rightarrow \infty$. RBO addresses this and can extrapolate accurately when using suggested values of $k = 50$ and $p = 0.9$ to calculate an extrapolated point estimate of list similarity, RBO_{EXT}. *Learnability* [3], is estimated as "one minus the average distance that items move as a proportion of half of the total menu length". While this is non-weighted, we used $k = 10$, focusing list comparisons on the most important items. Items that were in only one list were treated as having moved half the total menu length.

For all stability measures, we calculated the average similarity score across comparisons of every pair of consecutive lists for each *log-algorithm* pairing in the simulation.

Analysis Summary

The log-based analysis investigates performance by comparing the algorithms' predictions before each activity with the actual activity recorded in the logs. Several measures of prediction accuracy (e.g., percentage of first predictions, average position of the action in the prediction ranked list, etc.) and of stability are analysed. The factors under study are as follows:

Algorithm	$\in$	{*MRU, MRF, Adaptive+MRU, PF, SR&F* ($n = 5$)*, CRF* ($\lambda = 0.1$)*, and six AccessRank variations* ($\lambda = 0.8$ *and* 1.65, $\delta = 0$, 0.2 *and* 0.5)}
Log dataset	$\in$	{*window switching, application switching, web URLs, web domains, Unix command lines, Unix commands*}

Results

AccessRank ($\lambda = 1.65$, $\delta = 0$) has the highest *Percentage Revisitations Predicted* score with 41% overall (top in four datasets and second in one). All AccessRank configurations with $\delta = 0$ or 0.2 performed best. It also had the lowest average rank of 10.3 (lowest in four domains and second lowest in the other two), followed by *Markov* (10.7).

Relatively short prediction lists contain most revisitations. Figure 1 shows this for three datasets that incorporate all log data and illustrates both AccessRank's best and worst case accuracy. While AccessRank and *Markov* generally perform best, there is variation between domains – *Markov* performs best for web pages; AccessRank for Unix commands.

We also analysed how the algorithms perform when user-typed 'hints' are given (Figure 2). Normally, just a few characters will identify the target as the top prediction. Web browsing logs, however, often require more, in order to determine which *page* within a domain is targeted.

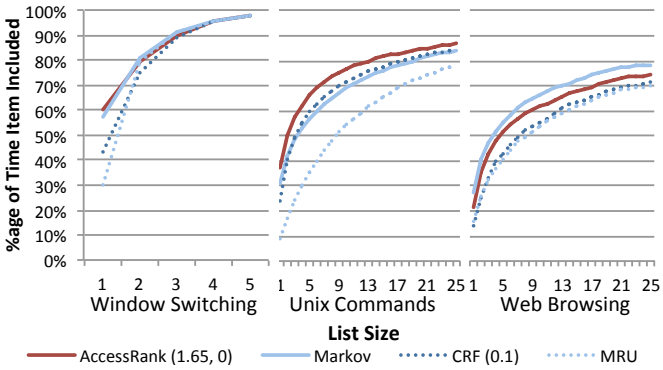

Figure 1: Percentage of revisitations that are included in a prediction list of a given size, across three datasets.

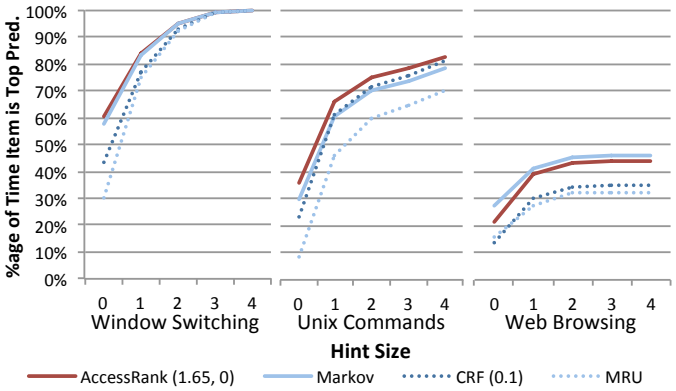

Figure 2: Percentage of revisitations that are the top match when filtered by a prefix of a given size, across three datasets.

The three stability measures (AO, RBO and Learnability) produced very similar results. *MFU* was the most stable algorithm, followed by *Places Frecency*, with *Markov* least stable. AccessRank stability improved significantly with δ.

An ideal algorithm will give accurate predictions *and* high stability. Figure 3 graphs *Stability* (using *Learnability*, which is the most applicable to HCI and had no strong bias) against *Average Rank* and *Percentage Revisitations Predicted*. The latter is especially important as the top result is most useful – interfaces provide rapid 'press to confirm' access for such items. AccessRank performs best in this important category, but it also matches the best algorithms in both accuracy and stability. Both graphs illustrate the flexibility of AccessRank, in that it can be tailored to optimise either accuracy or stability as required, while still outperforming other algorithms.

DISCUSSION AND FUTURE WORK

AccessRank is a flexible algorithm that outperforms existing algorithms in a variety of contexts. Based on our results, we recommend (λ, δ) values of (1.65, 0.2) to give the best

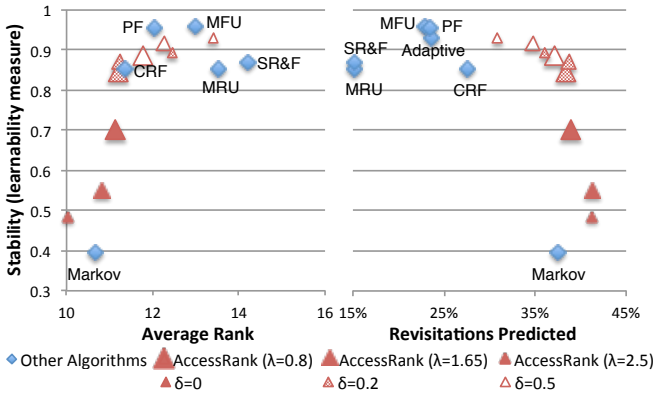

Figure 3: Learnability vs Average Rank and Percentage Revisitations Predicted over all datasets. Lower average ranks are better, while a higher percentage of revisitations predicted is better.

compromise between accuracy and stability. When stability is unimportant, values of (1.65, 0) give the best top prediction accuracy, while (2.5, 0) may be better if the average rank is the primary goal. When stability is particularly important, high values for both parameters can be used, e.g. (2.5, 0.5).

AccessRank could be incorporated in interfaces for any domain containing patterns of reuse. For example, a file browser could use an *AwesomeBar*-like interface that suggests files based on filename portions, eliminating the need to traverse hierarchies. A separate section could display the top weighted items in the file hierarchy subtree rooted at the current location. Window switching interfaces could highlight windows that are more likely to be acquired, to help users find relevant windows without confusing them by ordering items unpredictably. Many other domains could also benefit.

While AccessRank is a powerful algorithm, there are still improvements that could be made. The time weighting is based exclusively on the absolute time, but might be more effective if, for example, events were considered relative to the computer's first use each day. Location awareness could be incorporated; for example, if a user is in range of their work wireless network, accessed items will differ from when in range of their home network. Furthermore, while AccessRank is domain-independent, domain specific improvements could further improve its performance; for example, for file browsing past access methods could be considered (such as browsing, search or *Open Recent* menus), as could the interaction between hints and path components.

CONCLUSIONS

We have described AccessRank, a customisable revisitation prediction algorithm that is more accurate than existing algorithms while also producing stable results. Performance simulations over a range of datasets confirm its applicability in a wide variety of contexts, and demonstrate that different parameters can be used to tailor AccessRank to specific situations and design goals. In ongoing and further work, we will deploy the AccessRank algorithm in a range of applications and empirically assess its performance.

ACKNOWLEDGEMENTS

We thank Susanne Tak and Saul Greenberg for providing the log data used in our study. This work was partially funded by Royal Society of New Zealand Marsden Grant 10-UOC-020.

REFERENCES

1. Alexander, J., Cockburn, A., Fitchett, S., Gutwin, C., and Greenberg, S. Revisiting read wear: analysis, design, and evaluation of a footprints scrollbar. In *Proc. CHI '09* (2009), 1665–1674.
2. Arcuri, M., Coon, T., Johnson, J., Manning, A., and van Tilburg, M. Adaptive menus, Sept. 19 2000. US Patent 6,121,968.
3. Cockburn, A., Gutwin, C., and Greenberg, S. A predictive model of menu performance. In *Proc CHI '07* (2007), 627–636.
4. Findlater, L., and McGrenere, J. A comparison of static, adaptive, and adaptable menus. In *Proc. CHI '04* (2004), 89–96.
5. Findlater, L., Moffatt, K., McGrenere, J., and Dawson, J. Ephemeral adaptation: the use of gradual onset to improve menu selection performance. In *CHI '09*, ACM (2009), 1655–1664.
6. Greenberg, S. Using unix: Collected traces of 168 users. Tech. rep., Research Report 88/333/45, Department of Computer Science, University of Calgary, 1988.
7. Greenberg, S., and Witten, I. Supporting command reuse: empirical foundations and principles. *International Journal of Man-Machine Studies 39*, 3 (1993), 353–390.
8. Greenberg, S., and Witten, I. Supporting command reuse: Mechanisms for reuse. *International Journal of Man-Machine Studies 39*, 3 (1993), 391–425.
9. Kaasten, S., and Greenberg, S. Integrating back, history and bookmarks in web browsers. In *Proc. CHI '01* (2001), 379–380.
10. Lee, D., Choi, J., Kim, J.-H., Noh, S. H., Min, S. L., Cho, Y., and Kim, C. S. On the existence of a spectrum of policies that subsumes the least recently used (lru) and least frequently used (lfu) policies. *SIGMETRICS Perform. Eval. Rev. 27*, 1 (1999), 134–143.
11. Markov, A. The theory of algorithms. *American Mathematical Society Translations* (1960).
12. Mozilla. The Places frecency algorithm. `https://developer.mozilla.org/en/The_Places_frecency_algorithm`, 2008.
13. Tak, S. *Understanding and Supporting Window Switching*. PhD thesis, University of Canterbury, 2011.
14. Tauscher, L., and Greenberg, S. How people revisit web pages: empirical findings and implications for the design of history systems. *IJHCS 47* (1997), 97–138.
15. Webber, W., Moffat, A., and Zobel, J. A similarity measure for indefinite rankings. *TOIS '10* (2010).