BIGFile: Bayesian Information Gain for Fast File Retrieval

Wanyu Liu[1,2] **Olivier Rioul**[1] **Joanna McGrenere**[3,2] **Wendy E. Mackay**[2] **Michel Beaudouin-Lafon**[2]

[1]LTCI, Telecom ParisTech
Université Paris-Saclay
F-75013, Paris, France

[2]LRI, Univ. Paris-Sud, CNRS,
Inria, Université Paris-Saclay
F-91400, Orsay, France

[3]Dept of Computer Science
University of British Columbia
Vancouver, BC, Canada

{wanyu.liu,olivier.rioul}@telecom-paristech.fr, joanna@cs.ubc.ca, {mackay, mbl}@lri.fr

ABSTRACT

We introduce *BIGFile*, a new fast file retrieval technique based on the Bayesian Information Gain framework. *BIGFile* provides interface shortcuts to assist the user in navigating to a desired target (file or folder). *BIGFile*'s split interface combines a traditional list view with an adaptive area that displays shortcuts to the set of file paths estimated by our computationally efficient algorithm. Users can navigate the list as usual, or select any part of the paths in the adaptive area. A pilot study of 15 users informed the design of *BIGFile*, revealing the size and structure of their file systems and their file retrieval practices. Our simulations show that *BIGFile* outperforms Fitchett et al.'s AccessRank, a best-of-breed prediction algorithm. We conducted an experiment to compare *BIGFile* with *ARFile* (AccessRank instantiated in a split interface) and with a Finder-like list view as baseline. *BIGFile* was by far the most efficient technique (up to 44% faster than *ARFile* and 64% faster than *Finder*), and participants unanimously preferred the split interfaces to the Finder.

ACM Classification Keywords

H.5.2. [Information Interfaces and Presentation]: User Interfaces-Interaction styles

Author Keywords

Navigation-based file retrieval; Split adaptive interfaces; Bayesian approach; Mutual information

INTRODUCTION

Navigating through a file hierarchy is one of the most common methods for accessing files. Previous studies [1, 3, 6, 15] have shown users' continued preference for hierarchy-based file navigation over alternative methods, such as search. One of the most important reasons is that the locations and mechanisms of navigation-based retrieval remain consistent and reliable, whereas the organization and content of search results, which are extracted without context, can vary from one retrieval to the next, resulting in extra cognitive effort [3, 15, 30].

CHI 2018, April 21–26, 2018, Montreal, QC, Canada

 ISBN 978-1-4503-5620-6/18/04…$15.00

DOI: https://doi.org/10.1145/3173574.3173959

In addition to using search to potentially improve file access, e.g. [8], researchers have tried to tackle this problem by looking at different aspects of file retrieval: how users organize personal information [5] and how visualization [27] and prediction algorithms [10, 14, 31] can improve efficiency. Fitchett & Cockburn introduced AccessRank [14], an algorithm to predict what users are going to look for next, and demonstrated using three real datasets that it outperforms other prediction algorithms such as MRU (Most Recently Used) and Markov chain models [34]. Fitchett et al. [16, 17] later embedded AccessRank into three different navigation-based file retrieval interfaces. Their controlled experiment [16] showed the effectiveness of these interfaces in comparison to the Mac's Finder on a three-level file system hierarchy. In the present work we replicate their core study methodology and extend it to a six-level hierarchy in order to compare AccessRank to the new algorithm that we introduce below.

Liu et al. [32] recently introduced Bayesian Information Gain (BIG), an information-theoretic framework where the computer attempts to maximize the information gained from each user input by generating views that challenge the user, leading to more efficient identification of the intended target. Liu et al. [32] applied BIG to multiscale navigation and showed that the resulting technique, called BIGnav, was much more efficient than standard pan and zoom. However, users felt that they lost some control since they could not easily predict the next view generated by the system in response to their input.

In this paper, we introduce *BIGFile*, a fast file retrieval technique based on the BIG framework where the computer tries to gain information from the user to determine which file or folder she is looking for. Unlike BIGnav, *BIGFile* features a *split adaptive interface* that combines a traditional file navigation interface with an adaptive part that displays the shortcuts selected by BIG. Previous studies have demonstrated that split adaptive interfaces can improve both performance and satisfaction compared to static designs [11, 12, 19]. The adaptive shortcuts are calculated by our computationally efficient algorithm BIGFileFast. BIGFileFast improves over the original BIG algorithm by working efficiently with a large and evolving hierarchical structure at run time.

We begin with a review of related work, and then describe *BIGFile*'s interface and its underlying algorithm: the optimal algorithm BIGFileOpt and the computationally efficient algorithm BIGFileFast. After describing a pilot study of the actual file structures of real users and their file navigation practices,

we report on two studies: a set of simulations that compare BIGFileFast with AccessRank (Study 1), and an experiment that compares *BIGFile* with ARFile, a split file interface using AccessRank for prediction, with the Finder as baseline (Study 2). Finally, we discuss the limitations of this work and how BIGFileFast can be applied to other hierarchical structures.

RELATED WORK

We discuss related work on personal file systems, file retrieval techniques, and adaptive user interfaces.

Personal File Systems

Prior studies have investigated how people manage and retrieve information from their personal file systems. Users have thousands of files and folders in their file systems (Goncalves & Jorge [21] found 8000 on average and Henderson & Srinivasan [26] 5850). They create hierarchical structures that reflect their activities, as reported by Bergman [2, 3, 4, 5], Henderson [24, 25, 26] and Jones [28]. Although users tend to have different habits for building these structures (e.g. *Filer vs. Piler* [39]), Bergman and Henderson [4, 26] found that users' file hierarchies tend to be shallow (mean retrieval depth of 2.86 folders), with small folders (a mean of 11.82 files per folder) containing many subfolders (mean = 10.64). Based on this data, Fitchett et al. [16] used a three-level hierarchy in their experiment. Our pilot study, reported below, investigates current file system structures and user practices.

Regarding file retrieval, both Bergman [3] and Fitchett [15] found strong preferences for navigation-based retrieval, regardless of the substantial improvements in search tools. This might be because navigation-based retrieval requires less cognitive effort. Bergman [5] also found a difference in retrieval time between Windows and MacOS users (17.3 seconds on Windows vs. 12.6 on MacOS on average). Furthermore, as with command usage [22] and email messages [9], Fitchett [15] also found that the frequencies of file retrievals can be approximated by Zipf's Law [40], suggesting that people's patterns of file retrieval are strongly repetitive, with a small number of frequently revisited files, and a large number of infrequently visited ones. Based on this finding, *BIGFile* incorporates usage history information to improve its accuracy.

Enhanced File Retrieval Techniques

Researchers have explored how to use search, visualization and prediction to enhance file retrieval. Search is an essential tool for file retrieval as it provides direct access to the folders or files that are associated with the search keyword [3, 8].

Visualization techniques such as tree maps [27] and hyperbolic trees [29] have been introduced to depict hierarchical structures. However, this type of technique has not been widely adopted for file systems. Fitchett & Cockburn [14] introduced a prediction algorithm called AccessRank that blends Markov chains [34], combined recency and frequency [31] and time weighting to predict what users will do next. This algorithm was used to create three augmented navigation-based file retrieval techniques [16]: *Icon Highlights*, where the predicted items are highlighted to provide visual aids and reduce visual search time; *Hover Menus* where the predicted folder content is presented as shortcuts when hovering over an item; and *Search Directed Navigation* where predictive highlighting is provided to guide users through the hierarchy in response to query terms. While all three interfaces were more efficient than the baseline Finder, Hover Menus was the least preferred by the participants, and was removed from the field evaluation [17]. *BIGFile* builds on this work and is closest to Hover Menus, which provides shortcuts to reduce the number of levels to traverse. However *BIGFile* uses a different prediction algorithm and a split adaptive interface.

Adaptive User Interfaces

Adaptive user interfaces (AUIs) are a class of interfaces that adapt the presentation of functionality automatically, in response to individual user behavior or context. Research results on adaptive user interfaces are mixed: AUIs can lead to better user performance and satisfaction [13, 20, 36], but can also lead to the user being confused and feeling a loss of control over the interface [10, 19, 35].

Split interfaces, which are based on Split Menus [37], are a type of AUI that are particularly effective [11, 12, 19]. A split interface typically has two parts: a static part that represents the "status quo" original interface, and an adaptive part that augments the static part. The adaptive part changes its contents based on what the system believes the user needs. Users have the choice between interacting with the static part or the adaptive part. Thus, the learnability of the original interface is not hindered, and user performance can be enhanced if users take advantage of the adaptive part. Most split interfaces to date have been designed for menu selection [13], but other interface elements have also been split, such as the toolbar [19], emails [7], and relevant documents on Google Drive [38]. *BIGFile* introduces split interfaces for hierarchical file systems.

BIGFILE INTERFACE

BIGFile improves navigation-based file retrieval efficiency by providing *shortcuts* that can reduce the number of steps (user inputs) to reach the target file or folder: the user can skip levels in the hierarchy by selecting one of the shortcuts. We first present the *BIGFile* interface before describing the algorithm that determines the shortcuts.

BIGFile features a split adaptive interface (Fig. 1): the shortcuts are presented in the *adaptive area* at the top, while the *static area* at the bottom is a traditional list view of the current folder. The shortcuts in the adaptive area are the paths to the items selected by the *BIGFile* algorithm, relative to the current folder. Displaying the relative paths, rather than just the items, offers users contextual information that helps them determine if they correspond to the target they are looking for. It also lets users navigate directly to any folder in the path by clicking on it, typically when the target is not in the shortcuts, but a partial path to it is. Finally a back button (visible in Fig. 5) lets users go back to the previous state of the interface.

Both the shortcuts in the adaptive area and the items in the static area are updated after each user input. Similar to many other split adaptive interfaces [10, 37], if the system correctly estimates the user's target item, the user can select the shortcut,

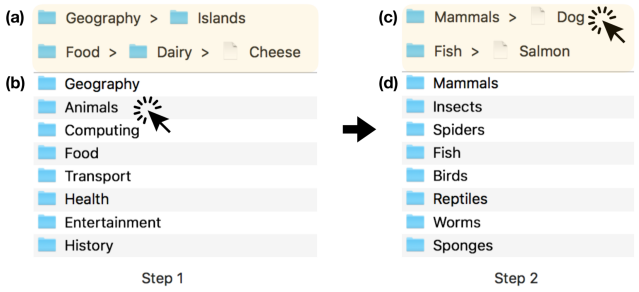

Figure 1. The *BIGFile* interface as the user navigates to "Dog" in a file retrieval task. (a) and (c) show the adaptive part with two shortcuts, (b) and (d) the static part. In Step 1, the shortcuts do not help and the user selects "Animals" in the static part, leading to Step 2 where the user directly selects "Dog" in the first shortcut, saving one step.

or navigate the hierarchy using the static part as usual. If none of the system's estimates are correct, the impact for the user is minimal since the items remain at their usual locations in the static part of the interface.

For example, in Fig. 1 (left), "Islands" and "Cheese" are the estimated items, presented along with their paths in the designated adaptive area (a). The static area (b) presents the usual hierarchy. A user could, for example, click on "Dairy" to access dairy products other than "Cheese" inside the folder (not shown in the figure). If the user clicks on "Animals", the static area is updated, showing the items inside the "Animals" folder (Fig. 1.d). The adaptive area is also updated with a new set of estimated targets ("Dog" and "Salmon", Fig. 1.c). If the user is looking for the item "Dog", she can save one step ("Mammals") by clicking the shortcut in the adaptive area. The number of shortcuts is user-customizable.

We created and considered a number of alternative designs for the interface, including an integrated view where each shortcut is displayed, together with its path, next to the corresponding root folder in the list view. However, we found that this integrated view makes it difficult to display shortcuts of arbitrary depth. Moreover, scrolling the view often hides shortcuts, which partially defeats their purpose. In addition, this design only works for the list view, while the split interface can work with any of the traditional views in the static area, e.g. the icon and column views of the Mac OS Finder. Therefore, we chose what seemed to be the simplest and most obvious option for our first implementation and comparison.

Note that the split interface design is not specific to *BIGFile* and can be used with any algorithm that predicts potential targets. For example, we used it with the AccessRank algorithm in Study two, described below.

BIGFile ALGORITHM

We briefly review the Bayesian Information Gain (BIG) framework [32] and introduce an optimal algorithm for *BIGFile*, BIGFileOpt. Since its computational cost is too high, we then present BIGFileFast, a suboptimal but very efficient version, which is used in the rest of the paper. The code for both algorithms is available at https://github.com/wanyuliu/BIGFile.

Bayesian Information Gain Framework

The Bayesian Information Gain (BIG) framework provides a quantifiable measure of the information transmitted by the user to the computer to let the computer know what the user wants. Let us assume that the user wants to select a target among a set of potential targets Θ. Each potential target θ is associated with a probability $P(\Theta = \theta)$ representing the computer's knowledge about the user's goal. When the system presents a view X to the user, e.g. a set of targets, and receives an input Y from the user, e.g. whether or not the target is in the view, the system gains information about which target is or is not the real target. It can then update its knowledge accordingly, by adjusting the probability of each potential target. This adjustment requires a user behavior function $P(Y = y|\Theta = \theta, X = x)$, representing the likelihood that the user would give a certain input $Y = y$ when given a view $X = x$ and looking for the target $\Theta = \theta$. In order to maximize the information that the system expects to gain at each step, the system selects a view $X = x$ such that, according to its current knowledge, the different possible user inputs are equiprobable. In information-theoretic terms, equiprobable choices maximize entropy, and therefore information gain. A detailed description can be found in [32].

BIGFileOpt: Optimal Algorithm

In order to apply the Bayesian Information Gain framework to file retrieval, we consider a regular hierarchical file system. Without loss of generality, we consider a single window, with a current folder F. We define the following:

- Θ represents all the folders and files that a user might be interested in. Θ may include all the files and folders in the file system, but is more likely to be narrowed to a subset based on user preference or the task at hand. For example, it can be reduced to a subset of the user's home folder and/or to a category of files such as documents of a certain type. In the simulations and the experiment, we used only the files as potential targets and excluded the folders.
- For each potential target $\Theta = \theta$, the initial probability, at the beginning of a retrieval task, that it is the actual intended target is $P(\Theta = \theta)$. This probability distribution is calculated using the Combined Recency and Frequency (CRF) algorithm[1] [31] and is updated after each retrieval of a target by the user, to reflect interaction history. At each step of the retrieval task, i.e. after each user input, $P(\Theta = \theta)$ is updated using Bayes' rule, as described in Algorithm 1.
- X represents the view generated by the system when first opening a window and after receiving each user input in that window. This view is composed of the *static part* S, which shows the folders and files of the current folder F, and the *adaptive part* A, which shows the N folders and files that are produced by the *BIGFile* algorithm to serve as shortcuts at this step. A view $X = x$ is therefore represented by $S \bigcup A$. The number of shortcuts N is user customizable.

[1] CRF defines the weight of item f as $w_f = \sum_{i=1}^{m} \frac{1}{p}^{\lambda(t-t_i)}$ where m is the number of past accesses, t the current time and t_i the time of access i. p and λ are parameters; we use $\{p = 2, \lambda = 0.1\}$. The probability that a file θ is the target is calculated by normalizing its weight: $P(\Theta = \theta) = w_\theta / \sum w_f$

Algorithm 1: BIGFileOpt
Search the optimal set of N shortcuts.

Data: $\Theta, X, Y, P(Y=y|\Theta=\theta, X=x), IG_{max}=0$
Result: Return set A that, together with set S, has the maximal expected information gain (IG).

Receive user input $Y=y$
Update the probability distribution of Θ (Bayes rule):
$P(\Theta=\theta|X=x, Y=y) = \frac{P(Y=y|\Theta=\theta, X=x)P(\Theta=\theta)}{P(Y=y|X=x)}$
for *all the combinations A of N nodes that are below the current folder F in the hierarchy* **do**
  Compute $IG(S \cup A) = I(\Theta; Y|X(S \cup A))$
  $= H(\Theta) - H(\Theta|X(S \cup A), Y)$
  // I is mutual information and H is entropy
  if $IG(S \cup A) > IG_{max}$ **then**
    $IG_{max} = IG(S \cup A)$
    $A_{max} = A$
return A_{max}

- Y represents any user input. At each step, the user issues an input $Y=y$ to the system: The user can select any of the items in the static and adaptive parts, or go back to the previous view with the back button in case of an error.
- $P(Y=y|\Theta=\theta, X=x)$ represents prior knowledge about user behavior: given view x and target θ, what is the probability that user input is y at this step. For simplicity, one can assume that the user does not make mistakes and therefore that this probability is 1 if the user is issuing the correct input, 0 otherwise. Alternatively, one can use a calibration session, as in [32]. Note that since the user may select items that are not in Θ during the steps that lead to a selection, user behavior must be known for any item in the file system.

At each step, i.e. after each user input, the static part S of the interface is updated if the current folder has changed, i.e. if the user has clicked on a folder to navigate to it. Then the adaptive part A of the interface is updated to display the N items selected by the *BIGFile* algorithm.

Algorithm 1 presents BIGFileOpt, an optimal algorithm that finds the N items that, together with S, maximize the expected information gain from the user's next input. This slight modification of the original Bayesian Information Gain algorithm [32] lets us calculate an optimal view $S \cup A$. However, considering the sizes of typical personal file systems, this algorithm is not practical: the number of sets to test grows like f^N, where f is the number of files and folders and N the number of items in the adaptive part. We now present a suboptimal but computationally efficient algorithm to address this problem.

BIGFileFast: Efficient Algorithm

We take advantage of the hierarchical structure of the file system to select a set A' of targets that, together with S, has one of the highest, if not the highest, expected information gain. At each step, BIGFileFast creates a tree whose leaves are the n targets ($n > N$) with highest probability as follows: First, the tree contains the n targets and their parent nodes up to the root (Fig. 2b); Then BIGFileFast compresses this tree by replacing nodes that have a single child with that child (Fig. 2c). This much smaller tree vastly reduces the number of sets of N targets to try. For $N=4$, we found that $n=6$ gave good results; $n=10$ gave slightly better results, but was too slow for large hierarchies. Note that since the set is recomputed after each user input, it adapts dynamically to the user's navigation.

Algorithm 2: BIGFileFast
Efficiently search a suboptimal set of shortcuts.

Result: Return set A that, together with set S, has a suboptimal maximal expected information gain.
// Create the compressed tree
Create the minimal subtree of the current folder F that contains the n most probable targets
for *each element t of the tree* **do**
  if *t is the only child of its parent* **then**
    Replace the parent by t and remove the parent
// Search this simplified tree
Create A, a set of N nodes such that no node is in the subtree of another node
$IG_{max} = IG(S \cup A)$
while *there are more sets to explore* **do**
  if *Node $a_i \in A$ has a child a'_i not yet explored* **then**
    $A' = A - a_i + a'_i$
    Compute $IG'(S \cup A')$
    if $IG(S \cup A) > IG'(S \cup A')$ **then**
      Skip the subtree rooted at a'_i
    else
      $IG(S \cup A) = IG'(S \cup A')$
      $A = A'$
  else
    $A = A - a_i +$ the root of the next branch
    Compute $IG(S \cup A)$
  if $IG(S \cup A) > IG_{max}$ **then**
    $IG_{max} = IG(S \cup A)$
    $A_{max} = A$
return A_{max}

We further optimize the search as follows: Consider a candidate set $A = a_0, a_1, ...a_N$ and an item a_i of A with a child a'_i. Let A' be the set where a_i is replaced by its child a'_i: $A_1 = a_0, a_1, ..., a_{i-1}, a'_i, a_{i+1}, ..., a_N$. If the expected information gain for the set A' is lower than that of A, then we do not consider any set with an item in the subtree of a'_i, effectively pruning that subtree. Fig. 2 and Algorithm 2 describe the implementation of BIGFileFast used in the simulations and the experiment reported in the rest of the paper.

BIGFileFast dramatically reduces search time, making it real time, and selects sets of targets with near-optimal expected information gain. We ran simulations comparing it to BIGFileOpt and found that, for example, for 1000 targets and a 12-level hierarchy, BIGFileOpt takes roughly three minutes while BIGFileFast responds in interactive time. Also, on average, the expected information gain of BIGFileFast was 84.7% that of BIGFileOpt.

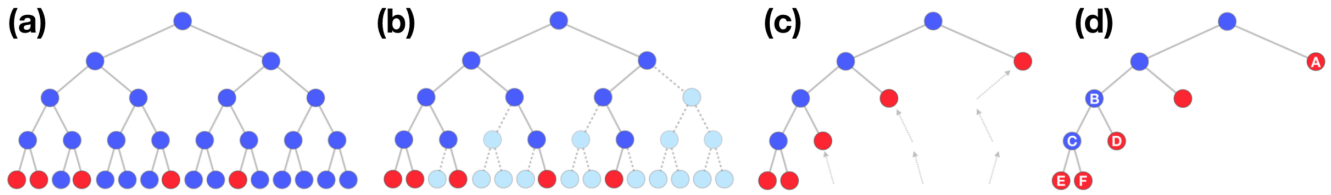

Figure 2. BIGFileFast with a binary tree: (a) Find the $n = 5$ most probable targets (red nodes); (b) Find their parents up to the root (dark blue nodes) and prune all the non-parent nodes (light blue nodes); (c) recursively replace the parent of a node by its child if it is the only child; (d) if the expected information gain of (A,B) is greater than that of (A,C), prune branch C and move directly to the next branch (A,D), skipping E and F.

PILOT STUDY

We conducted a pilot study to capture real users' file structures and understand their file navigation practices, informing our simulations (Study 1) and experiment (Study 2). We wanted to see if and how the structures and practices reported in the literature [4, 26] have changed. We recruited 15 participants from our institution, including faculty members, post-docs and students, all in technical areas. 13 were MacOS users, 2 were Windows users. We wanted to know the depth and breadth of their file systems, their navigation strategies, their preferred view for retrieving files, and the problems they run into.

Participants filled out a questionnaire, ran a script to get summary data of their file structures on their primary computer, and then reflected on their own file retrieval behavior. To run the script, participants identified the folder or folders that contain(s) the files that they navigate routinely with the Finder or File Explorer, such as the Documents and Desktop folders, but not the Music folder. The script counts the number of files and folders and returns a table with the file structure information and a graph visualizing the hierarchy (Fig. 3).

File Structures. Our findings differ somewhat from previous studies that found that people's file hierarchies tend to be shallow and broad (small depth and large branching factor), and have small and well defined folders [4, 26]. We found the average depth to be 7.7 ($\min = 5, \max = 10, \sigma = 1.18$), and interviews with participants confirmed that they do regularly navigate to deeper levels to access a file or folder. The average branching factor was 5.62 ($\min = 2.8, \max = 10.7, \sigma = 1.95$) and the average folder size 8.2 ($\min = 3.8, \max = 14.6, \sigma = 3.37$), which are relatively smaller than the findings in [4, 26], which found an average branching factor of 10 and an average folder size of 11. We also found that, in general, relatively fewer folders and files are nested at deep levels, suggesting that people do not build extremely complex file structures.

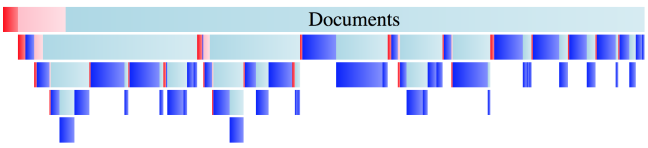

Figure 3. Visualization of P12's Documents folder, a 5-level tree. Each folder is represented by four bars: number of subfolders (branching factor) in red; number of files (folder size) in dark blue; total number of folders in subtree in pink; total number of files in subtree in pale blue.

Navigation Strategy. Ten participants reported that they first use Finder (or equivalent file navigator) to locate a file. Five reported using Spotlight (or equivalent search tool) first. This supports previous studies [3, 17] reporting that people usually navigate the hierarchical structure to locate a folder or a file, and use search as a "last resort" if navigation does not work.

Most-used View. Six participants preferred List view, four preferred Column view, three preferred both List view and Column view and regularly switch between them. Only two participants preferred Icon view, although many participants mentioned that they use Icon view to preview images.

Navigation Difficulty. Participants reported two main challenges when retrieving a file: (1) files/folders with repeated names, and (2) having partial knowledge about the location or the name of the target. This echoes previous studies [26, 33] reporting that people face retrieval difficulties stemming from semantic ambiguity. Hence, contextual information is crucial for a successful file retrieval. We used the findings from this pilot study to inform the designs of the simulations and experiment described next.

STUDY 1: SIMULATIONS

We ran simulations to investigate BIGFileFast's performance in estimating the target in a given hierarchical structure. We wanted to know how well the algorithm performs with respect to the following factors:

1. *Depth and width*: Both previous studies [21, 26] and our pilot study show that users have different file structures. Combining the results from [21, 26] and our pilot study, we used DEPTH = $\{4, 6, 8, 10, 12\}$ and BRANCHING FACTOR = $\{2, 4, 6, 8\}$ for the simulations.
2. *Initial distribution*: We did not log participants' use of their file system in our pilot study, but previous work indicates that file system use approximately follows a Zipf distribution [15]. To simulate different types of use history, we used two DISTRIBUTIONS: $Z(s = 1)$ and $Z(s = 2)$. The latter is a more skewed distribution describing cases where users focus primarily on a small set of targets.
3. *Size of target set*: Both [21, 26] and our pilot study suggest that users have different numbers of files and folders in their file system. Therefore, we used different target set sizes to see how BIGFileFast would perform. In our simulations, TARGET SET SIZE = $\{10, 100, 1000\}$.

We compared BIGFileFast with AccessRank [14], a prediction algorithm that incorporates several elements to predict

what users will do next: Markov chain model [34], CRF (combined recency and frequency) [31], and time weighting. Fitchett & Cockburn [14] demonstrated that AccessRank outperforms existing prediction algorithms. We therefore use AccessRank as our baseline.

In the case of navigation-based file retrieval, AccessRank predicts the target by assuming that a subfolder is likely to be selected if its parent folder is selected, captured by the Markov chain model. Similarly, BIGFileFast also assumes that the target is within the subtree of the current folder, and renormalizes the probabilities at each step. The key differences between AccessRank and BIGFileFast are as follows:

- AccessRank assigns a score to all folders and files while BIGFileFast only considers the set of potential targets.
- AccessRank updates the score of an item (file or folder) once it has been clicked while BIGFileFast updates the probability of all potential targets after each user input.
- AccessRank identifies the N items with highest scores while BIGFileFast identifies N items that provide the maximally informative view.
- AccessRank has a parameter δ to control the stability of the prediction list; BIGFileFast does not.

We generated a number of symmetric hierarchical structures crossing DEPTH with BRANCHING FACTOR = 2 and BRANCHING FACTOR with DEPTH = 4. When needed, extra targets were added at the deepest level so that there would be 100 and 1000 targets respectively. Depending on the target set sizes, we constructed a series of selections following the Zipf distributions. We randomized the mapping between the Zipf distribution and the targets, as well as the order of the selections.

We logged the number of steps needed to locate the target, the information gain and the accuracy rate for both algorithms. Note that we consider the folders on the path to the final target to be partially correct. For example, if the target is at level L_2 but the shortcut is only correct up to the folder at level $L_1 < L_2$, we consider the accuracy rate to be L_1/L_2, no matter how many steps it takes to get to the target level L_2.

We used $\{\alpha = 0.8, \delta = 0.5\}$ for AccessRank as in [16] and $\{p = 2, \lambda = 0.1\}$ in CRF for both AccessRank and BIGFileFast. We also assumed 100% correct user behavior for all simulations, i.e. that users would be as efficient as possible, always selecting an item from the adaptive area if it would get them to the target in fewer steps. Each condition [DEPTH × BRANCHING FACTOR × TARGET SET SIZE × DISTRIBUTION] was run 100 times, and the average taken.

Simulation Results

Fig. 4 shows the number of steps and the accuracy rate for the two algorithms using a $Z(s = 1)$ distribution. The results for $Z(s = 2)$ distribution are very similar; both BIGFileFast and AccessRank performed slightly better than they did with the $Z(s = 1)$ distribution. This is intuitive since both algorithms are based on frequency and recency of the file system use, and $Z(s = 2)$ focuses on a small set of very frequent items. In information-theoretic terms, the computer starts with more knowledge (less uncertainty) about the user's goal.

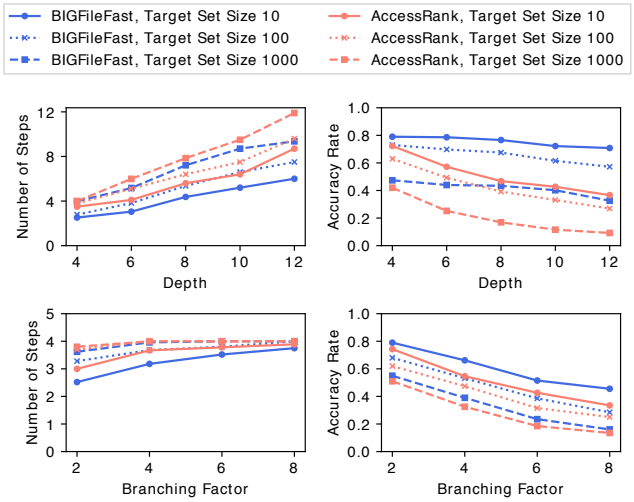

Figure 4. Simulations of BIGFileFast vs. AccessRank for a Zipf distribution ($s = 1$) and three target set sizes (10, 100, 1000). The plots show the number of steps (left, lower is better) and accuracy rates (right, higher is better) as a function of depth (top) and branching factor (bottom).

In general, BIGFileFast takes fewer steps to locate the target than AccessRank (Fig. 4, top left). In particular, the deeper the target is located, the better BIGFileFast performs: when DEPTH = 4, BIGFileFast averages 3.1 steps vs. 3.8 for AccessRank; when DEPTH = 12, BIGFileFast averages 7.6 steps vs. 10.1 for AccessRank. Increasing depth decreases the accuracy rate for both BIGFileFast and AccessRank, but the effect is more pronounced for AccessRank as shown in Fig. 4, top right: when DEPTH = 4, BIGFileFast is 66.5% accurate on average vs. 62.4% for AccessRank; when DEPTH = 12, BIGFileFast is 53.5% accurate vs. 24.2% for AccessRank.

This can be explained by the fact that AccessRank assigns a score to all files and folders; hence, the more levels that are traversed to get to a target, the more folders that are not targets themselves (but are on the way to the targets) have their scores increased. Another factor is that AccessRank takes user input into account for the next retrieval, but not within a retrieval task. If a node is shown but is not chosen, it will show up again at the next step as a prediction if it has a relatively high score and the final target is in the same parent folder. This results in wasting a prediction slot and not gaining information from the user. By comparison, BIGFileFast considers each user input within a retrieval, and since it assumes correct user behavior, if a node is shown and not chosen, all potential targets inside that node will be assigned a probability of 0. Therefore, the whole branch starting from that node will be discarded, i.e. it will not show up in the prediction slots at the next step.

Increasing the branching factor negatively affects both BIGFileFast and AccessRank (Fig. 4, bottom). The accuracy rate of BIGFileFast drops from 66.5% to 30.1% while the accuracy of AccessRank drops from 62.4% to 24.3%. This is not surprising as there is not much information from the user input for a wide but shallow (DEPTH = 4) hierarchy. Increasing target set size also negatively affects the performance of both BIGFileFast and AccessRank.

Averaged across all simulations, BIGFileFast is 15.5% more accurate and takes 23.1% fewer steps than AccessRank. The results can be summarized as follows: The deeper the target is located, the better BIGFileFast is than AccessRank; Both increased target set size and branching factor negatively affect the performance of both BIGFileFast and AccessRank; and BIGFileFast performs better on a deep hierarchy than on a broad hierarchy. We next compare *BIGFile* (which uses BIGFileFast) with a split interface using AccessRank in an experiment with real users.

STUDY 2: EXPERIMENT

We conducted an experiment to investigate the effectiveness of *BIGFile* with users. Our goal was to replicate and extend the methodology used by Fitchett et al. [16]. We used their implementation of the algorithm with the exception of one improvement which is noted below. We also used their hierarchical structure, which is a 3-level semantically organized hierarchy.

Since we learned from our pilot study that people do navigate to deeper levels, we extended their structure to 6 levels using the branching factors and folder sizes from Bergman [4]: 10, 5, and 4 folders, and 11, 8 and 7 files at levels 4, 5 and 6 respectively. Example targets for level 3 include 'Dog' with the path "Animals > Mammals > Dog" and 'Darwin' with the path "People > Inventors/Scientists > Darwin". Example targets for level 6 include 'Hawaii' with the path "Geography > Islands > Tropical > Touristic > Large > Hawaii", and 'Brie' with the path "Food > Dairy > Cheese > France > Creamy > Brie". As in [16], only the folders containing the final target are populated. In total, the hierarchy (available as supplemental material) contains 958 folders and 1068 files, of which 30 files are chosen as targets for each level-3 and level-6 condition.

Method

We used a [3×2] within-subject design with 3 INTERFACE conditions: *BIGFile* with the BIGFileFast algorithm, *ARFile*, a split interface using AccessRank for prediction, and a standard *Finder* as baseline; and 2 target LEVELS: 3 and 6.

We made a slight modification to AccessRank in order to make *ARFile* as effective as possible for users. In AccessRank, each folder and file is assigned a score. If users constantly go to the same item (file or folder), the algorithm's set of top predictions might include both the item and its parent folder. Since we are showing the full paths to the predicted items (not just the items themselves), this would result in an overlap between the shortcuts. Therefore we only show the deepest path if one shortcut is a prefix of another.

Elsweiler et al. [9] introduced the notion of Folder Uncertainty Ratio (FUR), which was later used by Fitchett & Cockburn [15] to illustrate users' uncertainty when navigating to files. If users are uncertain that they are going down the correct path, they are likely to select incorrect folders by mistake. In [15], users were found to be almost 94% accurate, while the other 6% of time, they might click on the wrong folder. Thus, we set the rate of correct user input to 94% and divided the remaining 6% among the other user inputs. These rates were used in the user behavior function in BIGFileFast

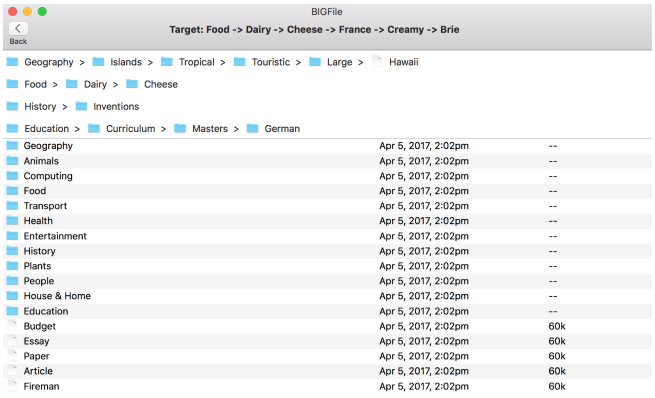

Figure 5. *BIGFile* experiment condition: The stimulus (full path to the target) is first presented in a modal window (not shown), and the participant must click "start" to begin the trial. The stimulus is also displayed at the top of the *BIGFile* browser throughout the trial. The image is cropped to save space: 11 additional files were visible below 'Fireman'.

and for calculating information gain in *ARFile* and in *Finder*. Furthermore, as in our simulations, we used $\{\alpha = 0.8, \delta = 0.5\}$ for AccessRank as in [16] and $\{p = 2, \lambda = 0.1\}$ for CRF for both AccessRank and BIGFileFast. A list view was used for the static part in all interface conditions because it was preferred in our pilot study.

Participants

Eighteen participants (7 women), aged 21 to 39 (mean = 28.5, σ = 5.1), all right-handed and with normal or corrected-to-normal vision, volunteered to participate in the experiment. Ten were MacOS users, eight were Windows users but were familiar with list view.

Apparatus

The experiment was conducted on a Macbook Pro with a 2.7 GHz processor, 8 GB RAM with resolution of 2560×1600. The file browser window was 880 × 631 pixels, as in [16]. One row on the list view takes 20 pixels. The software was implemented in Swift 3.0.

Procedure

The experiment consisted of two parts: practice, where participants familiarized themselves with the split interface using a training file hierarchy, and retrieval, where participants completed a series of file retrievals following a stimulus, which was presented as a path to the final target, e.g. "Food > Dairy > Cheese > France > Creamy > Brie", as shown in Fig.5.

During the retrieval phase, participants always started with level 3, and then level 6 using the same interface. At each level, they completed two sessions. Session 1 consisted of 20 file retrievals, which comprised 10 different target files following a near-Zipf distribution (frequencies 5, 3, 2, 2, 2, 2, 1, 1, 1, 1), as in [16]. Unlike [16], where the experiment started with a uniform probability distribution, we started with the above-mentioned Zipf distribution so that the item that was assigned a certain frequency would appear the corresponding number of times. For instance, if an item was assigned a frequency of 5, it would appear as the target stimulus 5 times during the session.

The mapping between frequency distribution and targets was counterbalanced across all participants and all conditions.

Each trial started by displaying the stimulus inside a popup window hiding the file browser. Participants were instructed to take as much time as they needed to understand the stimulus. When they were ready, they hit a start button to initiate the trial, at which point the content appeared inside the file browser (in both the adaptive and static parts, for the two conditions with split interfaces) and they were instructed to retrieve the file as fast and accurately as possible. When the popup window disappeared, the stimulus was shown in the toolbar at the top of the file browser, as in Fig.5. When the participant successfully clicked the target, a popup window appeared with the stimulus for the next trial. If they clicked a wrong target, a popup window let them know that they had made an error and asked them to try again. After clicking a folder or a file, the score for this item was updated in *ARFile*. Similarly, after each user input, the probability of each potential target being the actual target was updated, and after each retrieval, the initial distribution for the potential targets was updated in *BIGFile*.

Session 2 repeated Session 1 with the same initial distribution and randomized selection order. The goal was to see whether and how participants would use the split interfaces once they were more familiar with the file hierarchy and had some expectations about the targets, which is more representative of real use. Participants could take a break between sessions and between interface conditions.

For each level, we categorized the 30 targets into 3 non-overlapping groups of 10. To reduce learning effects stemming from familiarity with the hierarchy, within each group, the targets came from different top-level folders for level 3, and from different second-level folders for level 6. The order of interface and group of targets were counterbalanced using Latin Square across all participants. Thus, the target group, the order in which each target group is seen, the ordering of targets within a group, and the order in which each interface is seen all serve as control variables in our experiment.

After Session 2, for each interface, participants completed the NASA Task Load Index (TLX) worksheets [23] and provided comments on the interface. After all 3 conditions, we asked participants for their preferences among the three interfaces. The experiment lasted about 90 minutes.

Data Collection

For each trial, the program collects the task completion time (TCT), the number of steps a participant takes to locate the target (the number of items clicked, including the final target), the amount of time spent at each step, the uncertainty the computer has about the final target, the calculated shortcuts, the participant's input at each step, and the information gain after each input. We collected 3 INTERFACE × 2 LEVEL × 2 SESSION (20 Selections each) × 18 Participants = 4320 trials.

RESULTS

For our analyses, we first removed 60 outliers (about 1.3%) in which TCT was larger than 3 standard deviations from the mean. We verified that outliers were randomly distributed across participants, interfaces and conditions. We also checked for outliers for all our other dependent variables, but none were found. Note that the results are the same if we include the outliers in the analyses. Except where noted, we ran a repeated-measures INTERFACE × LEVEL × SESSION factorial ANOVA on our dependent measures[2].

Factors	*df, den*	*F*	*p*
INTERFACE	2, 34	452.47	< 0.0001
LEVEL	1, 17	895.61	< 0.0001
SESSION	1, 34	32.12	< 0.0001
INTERFACE × LEVEL	2, 34	211.89	< 0.0001
LEVEL × SESSION	1, 17	14.69	$= 0.0242$

Table 1. Significant effects in the full-factorial ANOVA on TCT.

Task Completion Time and Step Time

Table 1 shows the results of a repeated measures ANOVA on TCT. All main effects are significant, as well as two interaction effects: INTERFACE × LEVEL and LEVEL × SESSION.

On average, *BIGFile* is 39.3% faster than *ARFile*, and 59.0% faster than *Finder*, across all levels and sessions. The significant interaction effect between INTERFACE and LEVEL is shown in Fig. 6 (a) and (b). A post-hoc Tukey HSD test reveals that all differences are significant: *BIGFile* is 44.5% faster than *ARFile* and 63.8% faster than *Finder* at level 6, while *BIGFile* is 27.8% faster than *ARFile* and 47.6% faster than *Finder* at level 3. These findings are consistent with our simulation results: the deeper the target is located, the better BIGFileFast is compared to AccessRank.

The repeated measures ANOVA on step time (the time of a single step) shows only two significant main effects: INTERFACE ($F_{2,34} = 114.48, p < 0.0001$) and LEVEL ($F_{1,17} = 142.27, p <$

[2]All analyses are performed with SAS JMP, using the REML procedure to account for repeated measures.

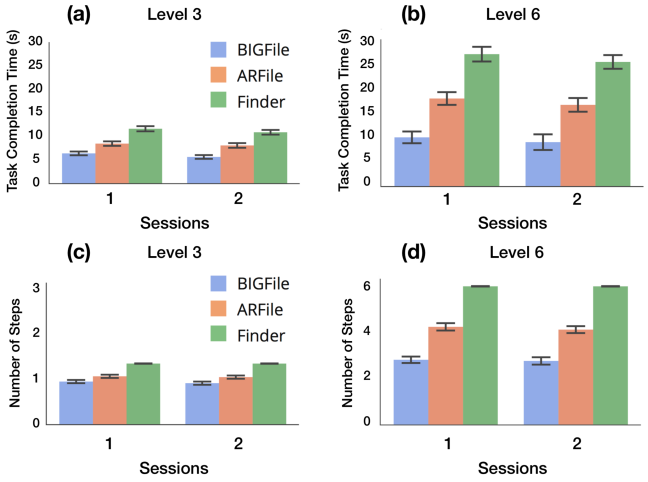

Figure 6. Task Completion Time (a, b) and number of steps (c, d) for the 3 interfaces, in 2 sessions, at levels 3 & 6, with 95% confidence intervals.

Factors	*df, den*	*F*	*p*
INTERFACE	2, 34	3178.13	< 0.0001
LEVEL	1, 17	7267.81	< 0.0001
INTERFACE × LEVEL	2, 34	739.30	< 0.0001

Table 2. Full-factorial ANOVA on the number of steps required to locate the target. Only significant effects are shown.

0.0001). A post-hoc Tukey HSD test indicates that *BIGFile* averages 3.29s per step, which is significantly faster than *ARFile* (3.78s), which is significantly faster than *Finder* (4.05s). In terms of levels, the average step time is 3.35s for level 3 vs. 3.94s for level 6, despite the fact that there are fewer files and folders at levels 5 and 6. Although not significant, the average step time is 3.78s in Session 1 vs. 3.52s in Session 2. The LEVEL × SESSION interaction indicates that the difference in performance between Session 1 and Session 2 was generally smaller at Level 6 than at Level 3, probably because Level 3 trials provided some training for Level 6 trials.

Number of Steps and Information Gain

Table 2 shows the results of a repeated measures ANOVA on the number of steps (user inputs) required to locate the target. Both INTERFACE, LEVEL and their interaction significantly affect the number of steps. A post-hoc Tukey HSD shows the following significant differences: for level 3, *BIGFile* takes 2.09 steps to locate the target, compared with *ARFile*'s 2.36 steps and *Finder*'s 3.08 steps; for level 6, *BIGFile* takes 2.8 steps vs. *ARFile*'s 4.22 and *Finder*'s 6.05, as shown in Fig. 6 (c) and (d). The interaction effect indicates that the impact of the algorithm is more pronounced at level 6 than level 3. This also reflects what we learned from the simulations: *BIGFile* has a greater impact on a deep hierarchy.

Regarding the average information gain, *BIGFile* gains 1.27 bits per user input on average, while *ARFile* gains 0.94 bits and *Finder* gains 0.69 bits. A typical plot of the 3 interfaces under the same condition at level 6 is shown in Fig. 7. We see that *BIGFile* gains more information from each user input, therefore the uncertainty drops to zero much faster at each step than with *ARFile* and *Finder*. By providing shortcuts, *ARFile* also gains more information from each user input than *Finder*.

Accuracy Rate and Characterization of Use

We use the same definition of accuracy rate as in the simulations. Across all factors, *BIGFile* is 68.4% accurate vs. 52.1% for *ARFile*. At level 3, *BIGFile* is 74.8% accurate vs. 61.2% for *ARFile*. At level 6, *BIGFile* is 62.8% accurate vs. 42.6% for *ARFile*. As in our simulations, a deeper hierarchy affects AccessRank more dramatically than BIGFileFast.

We also logged the use of the adaptive area with *BIGFile* and *ARFile*. We find that with *BIGFile*, the final target was present 64.7% of the time and the parent folder was present 89.3% of the time. In *ARFile*, the final target was present 48.5% of the time and the parent folder was present 76.4% of the time. When the final target was present in a shortcut, the participant selected it 99.2% of the time. When the target was absent but on of its parent folders was present in a shortcut, the participant selected it 96.8% of the time.

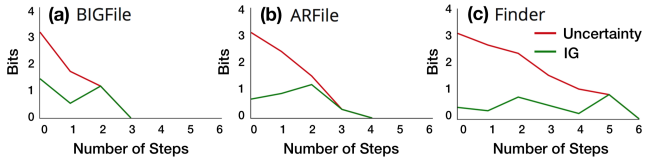

Figure 7. Uncertainty and information gain after each step in (a) BIGFile, (b) ARFile and (c) Finder.

Subjective Feedback

Subjective responses in all categories of the NASA TLX worksheets favored the split interfaces over *Finder* (Fig. 8). We note significant effects between *BIGFile* and *Finder* as well as between *ARFile* and *Finder* in terms of mental demand, physical demand, performance, effort and frustration. In the *Finder* conditions, four participants asked "*where is the section above [adaptive area]? Can you make it come back?*" and mentioned that "*this is really slow...*". However we found no significant differences between *BIGFile* and *ARFile*.

In terms of overall preference, participants ranked the interfaces (1 to 3). Both *BIGFile* and *ARFile* were preferred by participants over Finder, but there was no difference in preference between *BIGFile* and *ARFile*. When asked about their preference, 11 participants mentioned that they did not see any difference between *BIGFile* and *ARFile* and thought those interface conditions were just a different set of test words.

Three participants asked if they could search and whether the results in the adaptive area would correspond to their queries. Five participants mentioned that they would like to see "it" on their own file systems: "*I'm curious to see how it'd work on my own Finder. I have so many files everywhere with super long names... Wonder if this one will still work*". Another participant asked "*Can you reorder the list somehow as you like? Can you change the number [of shortcuts]?*" All these comments provide opportunities to further improve *BIGFile*.

DISCUSSIONS AND LIMITATIONS

We have seen that *BIGFile* is an effective technique, saving time and steps to retrieve a file in a hierarchical structure. We now discuss the benefits of split adaptive interfaces for file retrieval, provide a deeper comparison of BIGFileFast and AccessRank and outline some limitations of this work.

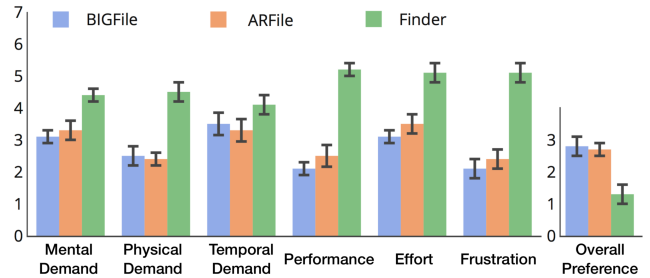

Figure 8. NASA TLX scores (from mental demand to frustration, lower is better) and overall preference (higher is better).

Split File Interface

To the best of our knowledge, *BIGFile* is the first attempt at introducing split adaptive interface to hierarchical file retrieval. Regardless of the underlying algorithm, BIGFileFast or AccessRank, the split interface outperformed the traditional Finder with unanimous preference from participants. This has not always been the case with adaptive interfaces, even split ones. For example, Gajos & Chauncey [18] have demonstrated systematic individual differences in the use of adaptive features, correlated with users' personality traits. Hence this approach does not benefit all users equally.

In our case, the preference might be due to the nature of the task: retrieving a file in a 3-level or 6-level hierarchy is much more difficult than selecting menu items, which is the task used in most split adaptive interface studies. Therefore, split adaptive interfaces may be more beneficial for difficult tasks where users need to "work hard" to reach their goal.

One possible issue with split interfaces is screen real estate. The more shortcuts are shown in the adaptive area, the better the underlying algorithm will work. But more shortcuts use more space and may result in higher cognitive demand and more occurrences of scrolling. Future work should therefore study the effects of the number of shortcuts on performance, preference and cognitive load.

Comparisons with AccessRank

Even though BIGFileFast can locate the final target more accurately than AccessRank in our simulations and experiment, unlike AccessRank, it does not account for repeated user behavior and repetitive access at the same time of day or day of the week. It also does not have a parameter to control the stability of estimated shortcuts across successive steps. These features are likely to benefit users in real settings. Future work should study their effect in *BIGFile*. AccessRank also needs to be compared with BIGFileFast in more realistic settings.

We were surprised that users did not express a preference between *BIGFile* and *ARFile*, attributing the differences to the set of targets rather than the underlying algorithm. This may be due to the fact that interface differences are more obvious to users than the inner workings of a system. Indeed, Fitchett et al. [16] found *Icon Highlights* and *Search Directed Navigation* to be more effective than *Hover Menus*, even though the latter predicts targets several levels down the hierarchy. In that respect, our split interface is an alternative to *Hover Menus* that shows to be effective for both BIGFileFast and AccessRank. Further work should therefore tease apart the respective roles of the interface and the prediction algorithm in file retrieval tasks.

Limitations

Despite *BIGFile*'s strong performance benefits, we want to emphasize some limitations of our experiment.

File Hierarchies: Our pilot study was designed to inform the design of our simulations and experiment in terms of the depth and width of the hierarchy we should evaluate. Even though we combined the results from our pilot study with those in the literature, it is still possible that the hierarchies we used are not fully representative. A larger scale study is needed to capture user file structures and retrieval practices.

Potential Target Set Size: We learned from the simulations that BIGFileFast performs much better on a 10-item potential target set than on a 1000-item potential target set. The latter is more realistic since users have thousands files and folders in their file systems. Larger target sets should therefore be tested to produce more robust findings.

Task Instruction: The task was initiated by showing a full path to the final target, which allowed users to compare the paths shown in the adaptive area with the instruction. In real life, recall of either the full path or the name of the final target is imperfect. Therefore, it is important to study how *BIGFile* performs in a more realistic setting, where navigation is combined with exploration.

CONCLUSION AND FUTURE WORK

We presented *BIGFile*, a fast navigation-based file retrieval technique where the computer is trying to gain information from the user by providing shortcuts that may help access the target faster. These shortcuts are presented in a split adaptive area of a file retrieval interface and include the estimated files or folders selected by our computationally efficient algorithm BIGFileFast, which together with the items in the current folder, maximize the expected information gain from the next user input. The interface includes the paths to the estimated items so that contextual information is provided to identify them. Users can use any shortcut in the adaptive area or simply navigate the hierarchy as usual.

We first ran a pilot study to better understand users' file structures and retrieval practices. We ran simulations demonstrating the effectiveness and accuracy of BIGFileFast compared to the AccessRank prediction algorithm in various hierarchical structures. We also ran an experiment comparing *BIGFile* with *ARFile*, a split interface using AccessRank, and with a Finder-like list view as baseline. *BIGFile* was up to 44% faster than *ARFile* and 64% faster than *Finder*, and users unanimously preferred the split interfaces.

Future work includes improving *BIGFile* by adding a stability parameter and potentially repeated user behavior. We also plan to evaluate *BIGFile* in a longitudinal study, and explore applications of the BIGFileFast algorithm to other areas.

ACKNOWLEDGMENTS

We thank Stephen Fitchett and Andy Cockburn for providing their data set and source code for AccessRank, the reviewers and our colleague Julien Gori for their helpful comments, and the participants in our studies. This research was partially funded by Labex DigiCosme (ANR-11-LABEX-0045-DIGICOSME), operated by the French Agence Nationale de la Recherche (ANR) as part of the program "Investissement d'Avenir" Idex Paris-Saclay (ANR-11-IDEX-0003-02), by European Research Council (ERC) grants n° 695464 "ONE: Unified Principles of Interaction", and n° 321135 "CREATIV: Creating Co-Adaptive Human-Computer Partnerships", and by NSERC RGPIN-2017-04549 "Highly Personalized User Interfaces".

REFERENCES

1. Deborah Barreau and Bonnie A. Nardi. 1995. Finding and Reminding: File Organization from the Desktop. *SIGCHI Bull.* 27, 3 (July 1995), 39–43. DOI: http://dx.doi.org/10.1145/221296.221307
2. Ofer Bergman, Ruth Beyth-Marom, and Rafi Nachmias. 2006. The Project Fragmentation Problem in Personal Information Management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, New York, NY, USA, 271–274. DOI: http://dx.doi.org/10.1145/1124772.1124813
3. Ofer Bergman, Ruth Beyth-Marom, Rafi Nachmias, Noa Gradovitch, and Steve Whittaker. 2008. Improved Search Engines and Navigation Preference in Personal Information Management. *ACM Trans. Inf. Syst.* 26, 4, Article 20 (Oct. 2008), 24 pages. DOI: http://dx.doi.org/10.1145/1402256.1402259
4. Ofer Bergman, Steve Whittaker, Mark Sanderson, Rafi Nachmias, and Anand Ramamoorthy. 2010. The Effect of Folder Structure on Personal File Navigation. *J. Am. Soc. Inf. Sci. Technol.* 61, 12 (Dec. 2010), 2426–2441. DOI: http://dx.doi.org/10.1002/asi.v61:12
5. Ofer Bergman, Steve Whittaker, Mark Sanderson, Rafi Nachmias, and Anand Ramamoorthy. 2012. How Do We Find Personal Files?: The Effect of OS, Presentation & Depth on File Navigation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2977–2980. DOI: http://dx.doi.org/10.1145/2207676.2208707
6. Richard Boardman and M. Angela Sasse. 2004. "Stuff Goes into the Computer and Doesn'T Come out": A Cross-tool Study of Personal Information Management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 583–590. DOI: http://dx.doi.org/10.1145/985692.985766
7. David Carmel, Liane Lewin-Eytan, Alex Libov, Yoelle Maarek, and Ariel Raviv. 2017. Promoting Relevant Results in Time-Ranked Mail Search. In *Proceedings of the 26th International Conference on World Wide Web (WWW '17)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 1551–1559. DOI: http://dx.doi.org/10.1145/3038912.3052659
8. Edward Cutrell, Susan T. Dumais, and Jaime Teevan. 2006. Searching to Eliminate Personal Information Management. *Commun. ACM* 49, 1 (Jan. 2006), 58–64. DOI:http://dx.doi.org/10.1145/1107458.1107492
9. David Elsweiler, Morgan Harvey, and Martin Hacker. 2011. Understanding Re-finding Behavior in Naturalistic Email Interaction Logs. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '11)*. ACM, New York, NY, USA, 35–44. DOI: http://dx.doi.org/10.1145/2009916.2009925
10. Leah Findlater and Joanna McGrenere. 2004. A Comparison of Static, Adaptive, and Adaptable Menus. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 89–96. DOI: http://dx.doi.org/10.1145/985692.985704
11. Leah Findlater and Joanna McGrenere. 2008. Impact of Screen Size on Performance, Awareness, and User Satisfaction with Adaptive Graphical User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1247–1256. DOI: http://dx.doi.org/10.1145/1357054.1357249
12. Leah Findlater and Joanna McGrenere. 2010. Beyond Performance: Feature Awareness in Personalized Interfaces. *Int. J. Hum.-Comput. Stud.* 68, 3 (March 2010), 121–137. DOI: http://dx.doi.org/10.1016/j.ijhcs.2009.10.002
13. Leah Findlater, Karyn Moffatt, Joanna McGrenere, and Jessica Dawson. 2009. Ephemeral Adaptation: The Use of Gradual Onset to Improve Menu Selection Performance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1655–1664. DOI: http://dx.doi.org/10.1145/1518701.1518956
14. Stephen Fitchett and Andy Cockburn. 2012. AccessRank: Predicting What Users Will Do Next. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2239–2242. DOI: http://dx.doi.org/10.1145/2207676.2208380
15. Stephen Fitchett and Andy Cockburn. 2015. An empirical characterisation of file retrieval. *International Journal of Human-Computer Studies* 74, Supplement C (2015), 1 – 13. DOI:http://dx.doi.org/10.1016/j.ijhcs.2014.10.002
16. Stephen Fitchett, Andy Cockburn, and Carl Gutwin. 2013. Improving Navigation-based File Retrieval. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2329–2338. DOI: http://dx.doi.org/10.1145/2470654.2481323
17. Stephen Fitchett, Andy Cockburn, and Carl Gutwin. 2014. Finder Highlights: Field Evaluation and Design of an Augmented File Browser. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 3685–3694. DOI: http://dx.doi.org/10.1145/2556288.2557014
18. Krzysztof Z. Gajos and Krysta Chauncey. 2017. The Influence of Personality Traits and Cognitive Load on the Use of Adaptive User Interfaces. In *Proceedings of the 22Nd International Conference on Intelligent User Interfaces (IUI '17)*. ACM, New York, NY, USA, 301–306. DOI: http://dx.doi.org/10.1145/3025171.3025192

19. Krzysztof Z. Gajos, Mary Czerwinski, Desney S. Tan, and Daniel S. Weld. 2006. Exploring the Design Space for Adaptive Graphical User Interfaces. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '06)*. ACM, New York, NY, USA, 201–208. DOI: http://dx.doi.org/10.1145/1133265.1133306

20. Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. 2008. Improving the Performance of Motor-impaired Users with Automatically-generated, Ability-based Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*. ACM, New York, NY, USA, 1257–1266. DOI: http://dx.doi.org/10.1145/1357054.1357250

21. Daniel J Gonçalves and Joaquim A Jorge. 2003. An empirical study of personal document spaces. In *DSV-IS*. Springer, 46–60. DOI: http://dx.doi.org/10.1007/978-3-540-39929-2_4

22. Saul Greenberg and Ian H Witten. 1993. Supporting command reuse: empirical foundations and principles. *International Journal of Man-Machine Studies* 39, 3 (1993), 353–390. DOI: http://dx.doi.org/10.1006/imms.1993.1065

23. Sandra G Hart and Lowell E Staveland. 1988. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. *Advances in psychology* 52 (1988), 139–183. DOI: http://dx.doi.org/10.1016/S0166-4115(08)62386-9

24. Sarah Henderson. 2005. Genre, task, topic and time: facets of personal digital document management. In *Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural*. ACM, 75–82. DOI: http://dx.doi.org/10.1145/1073943.1073957

25. Sarah Henderson. 2011. Document duplication: How users (struggle to) manage file copies and versions. *Proceedings of the Association for Information Science and Technology* 48, 1 (2011), 1–10. DOI: http://dx.doi.org/10.1002/meet.2011.14504801013

26. Sarah Henderson and Ananth Srinivasan. 2009. An empirical analysis of personal digital document structures. *Human Interface and the Management of Information. Designing Information Environments* (2009), 394–403. DOI:http://dx.doi.org/10.1007/978-3-642-02556-3_45

27. Brian Johnson and Ben Shneiderman. 1991. Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures. In *Proceedings of the 2Nd Conference on Visualization '91 (VIS '91)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 284–291. DOI: http://dx.doi.org/10.1109/VISUAL.1991.175815

28. William Jones, Ammy Jiranida Phuwanartnurak, Rajdeep Gill, and Harry Bruce. 2005. Don'T Take My Folders Away!: Organizing Personal Information to Get Ghings Done. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems (CHI EA '05)*. ACM, New York, NY, USA, 1505–1508. DOI: http://dx.doi.org/10.1145/1056808.1056952

29. John Lamping, Ramana Rao, and Peter Pirolli. 1995. A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '95)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 401–408. DOI: http://dx.doi.org/10.1145/223904.223956

30. Mark W Lansdale. 1988. The psychology of personal information management. *Applied ergonomics* 19, 1 (1988), 55–66. DOI: http://dx.doi.org/10.1016/0003-6870(88)90199-8

31. Donghee Lee, Jongmoo Choi, Jong-Hun Kim, Sam H Noh, Sang Lyul Min, Yookun Cho, and Chong Sang Kim. 1999. On the existence of a spectrum of policies that subsumes the least recently used (LRU) and least frequently used (LFU) policies. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 27. ACM, 134–143. DOI:http://dx.doi.org/10.1145/301453.301487

32. Wanyu Liu, Rafael Lucas D'Oliveira, Michel Beaudouin-Lafon, and Olivier Rioul. 2017. BIGnav: Bayesian Information Gain for Guiding Multiscale Navigation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 5869–5880. DOI: http://dx.doi.org/10.1145/3025453.3025524

33. Thomas W. Malone. 1983. How Do People Organize Their Desks?: Implications for the Design of Office Information Systems. *ACM Trans. Inf. Syst.* 1, 1 (Jan. 1983), 99–112. DOI: http://dx.doi.org/10.1145/357423.357430

34. Andrei Andreyevich Markov. 1960. The theory of algorithms. *Am. Math. Soc. Transl.* 15 (1960), 1–14.

35. J. Mitchell and B. Shneiderman. 1989. Dynamic Versus Static Menus: An Exploratory Comparison. *SIGCHI Bull.* 20, 4 (April 1989), 33–37. DOI: http://dx.doi.org/10.1145/67243.67247

36. Katharina Reinecke and Abraham Bernstein. 2011. Improving Performance, Perceived Usability, and Aesthetics with Culturally Adaptive User Interfaces. *ACM Trans. Comput.-Hum. Interact.* 18, 2, Article 8 (July 2011), 29 pages. DOI: http://dx.doi.org/10.1145/1970378.1970382

37. Andrew Sears and Ben Shneiderman. 1994. Split Menus: Effectively Using Selection Frequency to Organize Menus. *ACM Trans. Comput.-Hum. Interact.* 1, 1 (March 1994), 27–51. DOI: http://dx.doi.org/10.1145/174630.174632

38. Sandeep Tata, Alexandrin Popescul, Marc Najork, Mike Colagrosso, Julian Gibbons, Alan Green, Alexandre Mah, Michael Smith, Divanshu Garg, Cayden Meyer, and Reuben Kan. 2017. Quick Access: Building a Smart Experience for Google Drive. In *Proceedings of the 23rd*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17). ACM, New York, NY, USA, 1643–1651. DOI: http://dx.doi.org/10.1145/3097983.3098048

39. Steve Whittaker and Candace Sidner. 1996. Email Overload: Exploring Personal Information Management of Email. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '96)*. ACM, New York, NY, USA, 276–283. DOI: http://dx.doi.org/10.1145/238386.238530
40. GK Zipf. 1949. Human Behaviour and the Principle of Least-Effort, patterns. (1949).