FolderPredictor: Reducing the Cost of Reaching the Right Folder

XINLONG BAO and THOMAS G. DIETTERICH
Oregon State University

Helping computer users rapidly locate files in their folder hierarchies is a practical research problem involving both intelligent systems and user interface design. This paper reports on FolderPredictor, a software system that can reduce the cost of locating files in hierarchical folders. FolderPredictor applies a cost-sensitive prediction algorithm to the user's previous file access information to predict the next folder that will be accessed. Experimental results show that, on average, FolderPredictor reduces the number of clicks spent on locating a file by 50%. Several variations of the cost-sensitive prediction algorithm are discussed. An experimental study shows that the best algorithm among them is a mixture of the most recently used (MRU) folder and the cost-sensitive predictions. Furthermore, FolderPredictor does not require users to adapt to a new interface, but rather meshes with the existing interface for opening files on the Windows platform.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*User-centered design*; I.2.6 [**Artificial Intelligence**]: Learning—*Knowledge acquisition*

General Terms: Design, Human Factors

Additional Key Words and Phrases: Activities, directories, folders, intelligent user interfaces, intelligent systems, prediction, recommendation, shortcuts, tasks, user interface

1. INTRODUCTION

Computer users organize their electronic files into folder hierarchies in their file systems. But unfortunately, with the ever-increasing numbers of files, folder hierarchies on today's computers have become large and complex [Boardman and Sasse 2004]. With large numbers of files and potentially complex folder hierarchies, lo-

This research was supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010 and by DARPA, through Air Force Research Lab Contract No. FA8750-07-D-0185/0004. Additional support was provided by Intel Corporation and by the National Science Foundation under grant IIS-0133994.

Authors' address: Xinlong Bao, Google Inc., 6425 Penn Ave. #700, Pittsburgh, PA 15206, and Thomas G. Dietterich, School of Electrical Engineering and Computer Science, Oregon State University, 1148 Kelley Engineering Center, Corvallis, OR 97331-5501. email: Xinlong.Bao@gmail.com, tgd@eecs.oregonstate.edu.

cating the desired file is becoming an increasingly time-consuming operation. Some previous investigations have shown that computer users spend substantial time and effort in just finding files [Barreau and Nardi 1995; Jul and Furnas 1995; Ko et al. 2005]. Thus, designing intelligent user interfaces that can help users quickly locate desired files has emerged as an important research topic.

Previous research on helping users find files has focused on building Personal Information Management (PIM) systems in which documents are organized by their properties [Dourish et al. 2000; Dumais et al. 2003]. These properties include both system properties, such as name, path and content of the document, and user-defined properties that reflect the user's view of the document. In these systems, users can search for files by their properties using search engines. Although these search engines can be effective in helping the user locate files, previous user studies have indicated that instead of using keyword search, most users still like to navigate in the folder hierarchy with small, local steps using their contextual knowledge as a guide, even when they know exactly what they were looking for in advance [Barreau and Nardi 1995; Jones et al. 2005; Jul and Furnas 1995; Teevan et al. 2004].

In this paper, we try to address the file-locating problem from another perspective, using a system that we call the *FolderPredictor*. The main idea of FolderPredictor is that if we have observations of a user's previous file access behavior, we can recommend one or more folders directly to the user at the moment he/she needs to locate a file. These recommended folders are predictions — the result of running simple statistical prediction algorithms on the user's previously observed interactions with files.

Ideally we want to identify when the user has just started to look for a file and provide a shortcut to likely choices for the folder containing that file. In today's graphical user interfaces, there are several user actions that strongly indicate the user is or will be initiating a search for a file. These include the display of a file open/save dialog box or the opening of a file/folder browsing application such as Windows Explorer. Our approach intervenes both cases.

First, FolderPredictor presents predicted folders by changing the default folder of the open/save file dialog that is displayed to computer users from within an application. It also provides shortcuts to secondary recommendations, in case the top recommendation is not correct. Second, FolderPredictor presents predicted folders as buttons inside a Windows Explorer toolbar. Users can easily jump to the predicted folders by clicking these buttons. An important advantage of FolderPredictor is that it is easier for users to adapt to since it fits into existing UIs and does not require users to adapt to new interfaces/interactions.

The paper is organized as follows. In the next section, we introduce the TaskTracer system that FolderPredictor is built upon. Section 3 presents the user interfaces of FolderPredictor and the instrumentation for presenting predictions in the open/save file dialog box and the Windows Explorer. Section 4 describes how folder predictions are made, together with the cost-sensitive prediction algorithm and several variations of it. Section 5 reports the experimental results from two user studies: the first one establishes that FolderPredictor reduces the user's cost for locating files; the second one compares the performance of different variations of the basic prediction algorithm. Section 6 discusses some related work in email

classification. Finally, we conclude the paper in section 7.

2. THE TASKTRACER SYSTEM

FolderPredictor is built upon our TaskTracer system [Dragunov et al. 2005; Stumpf et al. 2005] — a software system designed to help multi-tasking computer users with interruption recovery and knowledge reuse. In this section, we briefly introduce the multi-tasking hypothesis and the data collection architecture of the TaskTracer system.

2.1 Multi-tasking Hypothesis

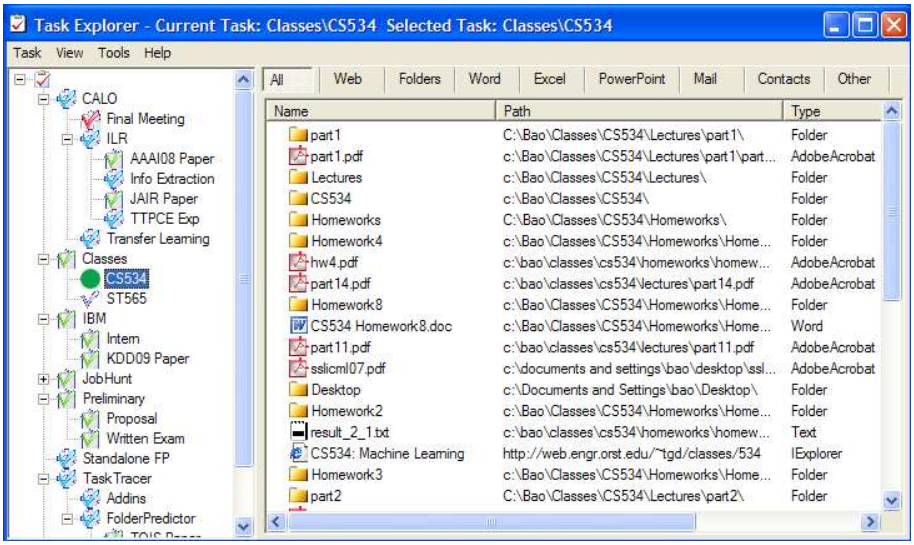
FolderPredictor monitors user activity to make predictions to optimize the user experience. Previous research in intelligent "agents" has explored this kind of approach [Horvitz et al. 1998; Maes 1994; Rhodes 2003]. However, one of the challenges faced by these intelligent agents is that users are highly multi-tasking, and they are likely to need access to different files depending on the exact task they are performing. For example, a professor may have a meeting with a student for one hour and then switch to writing a grant proposal. The notes file for the student meeting and the files for the grant proposal are normally stored in different folders. After the professor switches to working on the grant proposal, a naive agent might assume that the student notes are still the most likely choice for prediction, because they were the most recently accessed. This prediction would fail, because the agent is not taking into consideration the context of the user's current task.

Previous work on intelligent agents or assistants has attempted to produce more context-aware predictions by analyzing the content of the documents that the user is currently accessing to generate an estimated keyword profile of the user's "task" [Budzik and Hammond 2000]. This profile is then employed as a query to locate web pages with similar keyword profiles. This approach is limited because a) it cannot recommend resources that do not have text associated with them, b) there are substantial ambiguities in text, resulting in significant uncertainty in mapping from text profiles to user tasks, and c) the active window may not contain sufficient text. In the last case, the agent would be forced to scan recently-accessed documents to generate text profiles — documents that may have been accessed as part of a previous task.

In our TaskTracer system, three core hypotheses are made in order to characterize and utilize the user's multi-tasking behavior:

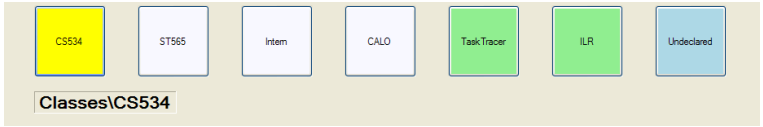
(1) All information workers break their work into discrete units to which they can give names — we call these *tasks*.
(2) At any moment in time, the user is working on only one task.
(3) Knowledge of the user's current task will substantially reduce the uncertainty in predicting what a user is trying to do.

Users define new tasks in the TaskTracer system via an application called the TaskExplorer. Users can then indicate their current task through three mechanisms, which are shown in Figure 1. Users can switch to the TaskExplorer (Figure 1(a)) and select the task from their defined hierarchy of tasks, or they can set the task by clicking on a taskbar widget that appears as a drop-down box at the



(a) TaskExplorer

(b) TaskSelector



(c) HotTask

Fig. 1. User interfaces for defining tasks and declaring current task in the TaskTracer system.

bottom-right corner of the screen. This taskbar widget is called the TaskSelector (Figure 1(b)), where the user can switch tasks by manually selecting another task from the drop-down menu or by typing in the textbox (with auto-completion). The third mechanism is through a tool called HotTask (Figure 1(c)). It shows a pop-up menu of tasks that are most-recently worked on, to provide quick access to them. The user can switch tasks by pressing Ctrl + ~ and stop at the desired task, just like switching windows by pressing Alt + Tab in Windows OS.

We also have a system called TaskPredictor, which can automatically detect task switches by observing the activity of the user [Shen et al. 2009]. If a probable task switch is detected, the user can be notified, actively or peripherally. Alternatively

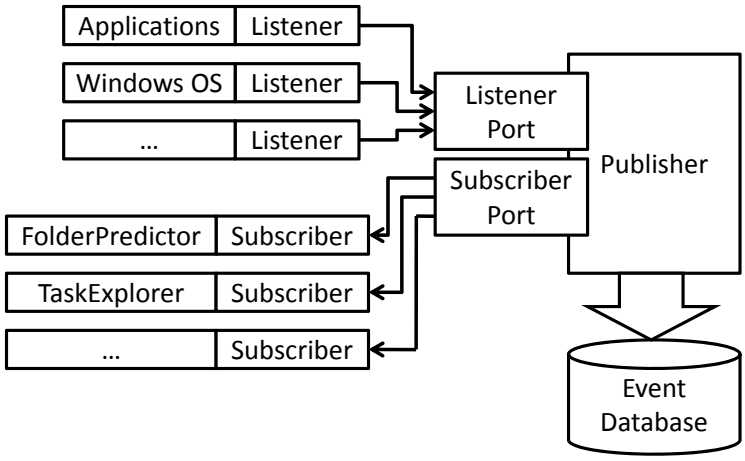


Fig. 2. Publisher-Subscriber architecture in the TaskTracer system.

the task can be automatically changed if task prediction confidence is high enough.

2.2 Data Collection Architecture

The TaskTracer system employs an extensive data-collection framework to obtain detailed observations of user activities. It instruments a wide range of applications under the Windows XP operating system, including Microsoft Office (Word, Excel, PowerPoint and Outlook), Internet Explorer, Adobe Acrobat, GSView and Notepad. It also instruments some operating system components including the system clipboard and window focus switches.

Listener components are plug-ins into applications. They capture user interaction events and send them to the *Publisher* as *Events*, which are XML strings with pre-defined syntax and semantics. One event of interest is the TaskBegin event, which is sent to the Publisher when the user switches tasks. The Publisher stores these events in its database and also distributes them to *Subscriber* applications that need to react to events online. FolderPredictor is one of the subscriber applications of TaskTracer. This *Publisher-Subscriber Architecture* is shown in Figure 2.

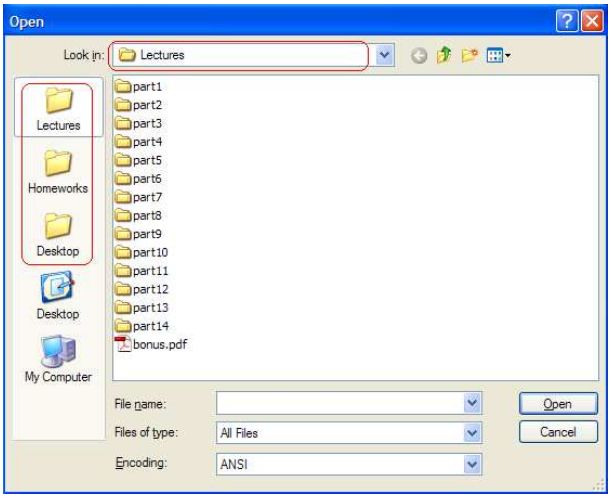
3. FOLDERPREDICTOR USER INTERFACES

A basic design principle of FolderPredictor is simplicity. This basic principle greatly influences the UI design — FolderPredictor actually has *no independent UI*.

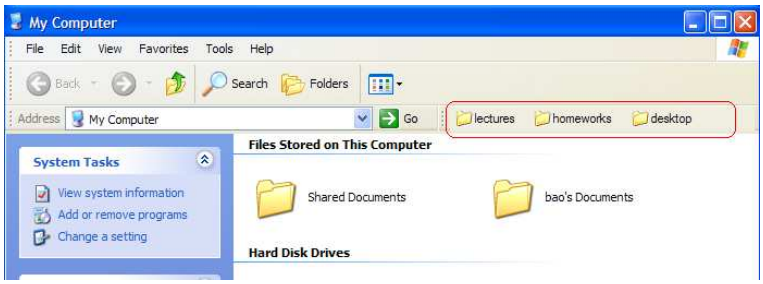
3.1 Display Predicted Folders in Existing User Interfaces

FolderPredictor provides three shortcuts to the likely choices for folders when the user has just started to look for a file (in particular, the display of a file open/save dialog box or the opening of a Windows Explorer window). FolderPredictor achieves this by modifying the existing user interfaces.

First, FolderPredictor enhances the well-known Windows Open/Save File Dialog boxes by adding its predicted folders. Figure 3(a) shows an example Open File Dialog box enhanced by FolderPredictor. Three predicted folders are shown as the



(a) Folder predictions in the Open/Save File Dialog box.



(b) Folder predictions in the Windows Explorer window.

Fig. 3. FolderPredictor user interfaces.

top three icons in the "places bar" on the left. The user can jump to any of them by clicking on the corresponding icon. The top predicted folder is also shown as the default folder of the dialog box so that the display will show this folder initially. Mousing over a folder icon will display the full path of that folder.

There are five slots in the places bar. By default, Microsoft Windows places five system folders (including "My Computer" and "Desktop") there. Informal questioning of Windows users revealed that several of these shortcuts were not commonly used. Thus, we felt it was safe to replace some of them by predicted folders. By default, FolderPredictor uses three slots for predicted folders and leaves two system folders. This behavior can be configured by the users if they want to see more system folders in the places bar.

Second, FolderPredictor adds an explorer toolbar to the Window Explorer window as shown in Figure 3(b). Three predicted folders are accessible to the user while browsing inside the Windows Explorer. Clicking on a folder icon will navigate the current Windows Explorer window directly to that folder. Mousing over a folder icon will display the full path of that folder.

These two interfaces hook into the native Windows environment and carry no

overhead for the user to learn additional interactions. An important advantage of FolderPredictor is that it reduces user cost while introducing only minor changes to the existing user interfaces.

3.2 Implementing the Instrumentation

Microsoft Office applications employ a file dialog from a custom library, while most other Windows applications use the file dialog from the common Win32 dialog library provided by the operating system. Thus, FolderPredictor needs three separate sets of instrumentation: the common Win32 file dialog, the Microsoft Office file dialog and the Windows Explorer toolbar.

3.2.1 *Instrumentation for the common Win32 file dialog.* In order to modify the places bar in the common Win32 dialog, FolderPredictor creates five entries named "Place0" to "Place4" under the registry key "HKCU\Software\Microsoft\Windows \CurrentVersion\Policies\ComDlg32\Placesbar\". These five entries correspond to the five slots, and their values can be set to the paths of the predicted folders or to the system-independent numeric IDs (CSIDL) of the system folders.

Modifying the default folder is much more difficult. There is no documented support for this feature from Microsoft. FolderPredictor modifies the default folder by injecting an add-in (a .DLL file written in C with embedded assembly language) to all applications when they are started. This add-in intercepts Win32 API calls invoked by the injected application. Common Win32 applications show the Open/Save file dialog by invoking API calls named "GetOpenFileName" or "GetSaveFileName" with the default folder passed as a parameter. Thus, the add-in can modify the default folder by intercepting the API call, changing the parameter, and then passing it on. A detailed introduction of API interception technology can be found in Pietrek [1995].

By intercepting the above two API calls, we can also get the original default folder of the file dialog and the folder returned by the file dialog. This information was used in the evaluation of FolderPredictor, which is presented in Section 5.

3.2.2 *Instrumentation for the Microsoft Office file dialog.* As with the common Win32 file dialog, the places bar in the Microsoft Office file dialog can be modified by manipulating registry keys. The pertinent registry key is "HKCU\Software\Microsoft\Office*VersionNumber*\Common\Open Find\Places\", inside which *VersionNumber* should be replaced by the version number of Microsoft Office software installed on the computer — for example, "11.0" for Office 2003 and "12.0" for Office 2007.

Microsoft Office uses a file dialog from a custom library, and the API calls in this library are not exposed. Therefore, API interception technology cannot be used to modify the default folder of Microsoft Office file dialog. FolderPredictor hooks into the Microsoft Office file dialog by loading add-ins created by Visual Studio Tools for Office (VSTO) into Microsoft Office applications. Code in these add-ins, written in C#.NET, is invoked when the file dialog is called. When invoked, the code changes the default folder of the file dialog to the predicted folder and then shows the dialog box.

3.2.3 *Instrumentation for the Windows Explorer toolbar.* FolderPredictor's toolbar in the Windows Explorer is a customized COM add-in registered as an explorer toolbar by adding its GUID under the registry key "HKLM\Software\Microsoft\Internet Explorer\Toolbar\". This add-in implements the IObjectWithSite interface to initialize and dispose itself. It subscribes to the TaskTracer events that contain FolderPredictor's predictions and updates the three folder buttons accordingly.

4. FOLDER PREDICTION ALGORITHMS

The main goal of FolderPredictor is to reduce the cost of locating files. In this paper, we assume that the user navigates in the folder hierarchy using a mouse, and the number of "clicks" necessary to reach the destination folder is an appropriate measure of the cost to the user. One "click" can lead the user from the currently selected folder to its parent folder or to one of its sub-folders. Our folder prediction algorithms seek to reduce the number of clicks needed to reach the user's destination folder from the predicted folders. We further assume that the system knows about the current user task (from the user interfaces shown in Figure 1), and predict three folders to put into the file dialog boxes and Window Explorer toolbar as in Figure 3.

In this section, we first introduce our approach for collecting possible target folders and assigning weights to them. Then we discuss the idea of including ancestor folders as candidate folders for predictions. After that, we introduce our cost-sensitive prediction algorithm, followed by several variations of it.

4.1 Collecting Possible Target Folders with Weights

Our approach assumes that computer users, for the most part, separate files for different tasks into different folders. We further assume that, for the most part, the working set of folders for a task is relatively small. Given such assumptions, the paths of the files that the user has accessed when working on a task provide useful information for making folder predictions for future accesses for that task. And the prediction algorithms that will be introduced later in this section tend to make task-specific folder predictions — different predictions will be made when the user is working on different tasks. For example, imagine that, during one day, a student opens and saves files under the folders "C:\Classes\CS534\Homeworks\" and "C:\Classes\CS534\Presentations\" when he is working on a task named *CS534*. The next day, when he returns to working on the *CS534* task, it should be useful to recommend these two folders or even the parent folder "C:\Classes\CS534\".

FolderPredictor generates its predictions by applying a simple statistical prediction method to a stream of observed file open/save events. Each of these events includes the path of the folder containing the file that was opened or saved. For each task, FolderPredictor maintains statistics for each folder — how many times the user opened files from it or saved files to it.

A statistical approach to making folder predictions is important for two reasons: 1) more-frequently-accessed folders should be more probable predictions and 2) users sometimes access the wrong files, or forget to specify that they have changed task. In the second case, the misleading events will add to the statistics. If observed accesses to a particular folder are really just noise, then we are unlikely to observe repeated future accesses to that folder over time. Thus these folders should have

relatively low access frequencies. We can use this information to filter out these folders from recommendations.

Another factor that should be considered is recency. Our hypothesis is that a user is more likely to need to access recently-accessed folders. For example, a programmer working on a big project may need to access many different folders of source code, working on them one by one, but accessing each folder multiple times before moving on to the next folder. Thus, the folders with older access times should be quickly excluded from the predictions when the programmer begins to work on source code under new folders. To achieve this, we have incorporated a recency weighting mechanism into FolderPredictor. Instead of keeping a simple count of how many times a folder is accessed, we assign a recency weight w_i to each folder f_i. All weights are initially zero. When a file in folder f_i is opened or saved while working on a task, the weights of all the folders that have been accessed on that task are multiplied by a real number α between 0 and 1, and w_i is then increased by 1. α is called the discount factor. Multiplying by the discount factor exponentially decays the weights of folders that have not been accessed recently. When $\alpha = 0$, only the most recently-accessed folder will have a nonzero weight, and it will always be predicted. When $\alpha = 1$, no recency information is considered, and weights are not decayed. Experiments show that a discount factor in the range [0.7, 0.9] performs the best. Another benefit of recency weighting is that folders erroneously identified as relevant due to noisy data are excluded from consideration quickly, because their weights decrease exponentially.

In the implementation of FolderPredictor, we apply an incremental update method to maintain the folder weights. The first time FolderPredictor is run on a computer, historical TaskTracer data, if available, are used to build the initial list of folders and associated weights for each task. This information is stored in FolderPredictor's private database. Then FolderPredictor incrementally updates this database as new open or save events arrive, until the FolderPredictor is shut down. The next time FolderPredictor is started, it updates its database using only the events that have been added since the last time FolderPredictor was shut down. This incremental update method helps FolderPredictor keep its data up-to-date without any perceivable delay in prediction time or startup.

4.2 Predicting Ancestor Folders Is Better Sometimes

After we have collected a list of folders with weights assigned to them, the folder prediction problem seems trivial: we could just recommend the folder having the highest weight. However, while that might maximize the chance that we pick the best possible folder, it may not minimize the average cost in clicks. Recall that our goal here is to minimize the number of clicks to reach the target folder. To do this, we may want to sometimes recommend an ancestor folder. We will motivate the need for this decision by an example. Suppose a student has a folder hierarchy as shown in the tree structure in Figure 4. His files are stored in the bottom level folders, such as "Part3" and "Hw2". When he worked on the *CS534* task, he accessed almost all of the bottom level folders with approximately similar counts. In this circumstance, predicting a bottom level folder will have a high probability of being wrong, because all the bottom level folders are equally probable. This will cost the student several more "clicks" to reach his destination folder — first to go

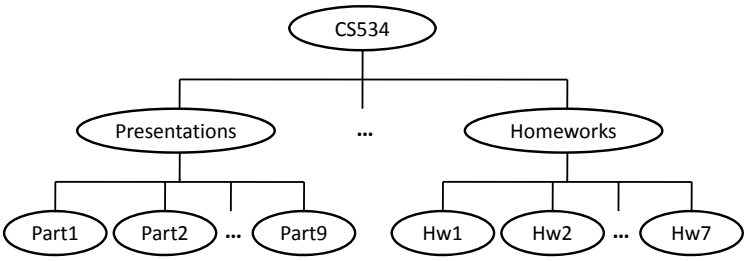


Fig. 4. An example folder hierarchy. Each node of this tree denotes a folder. Child nodes are sub-folders of their parent node.

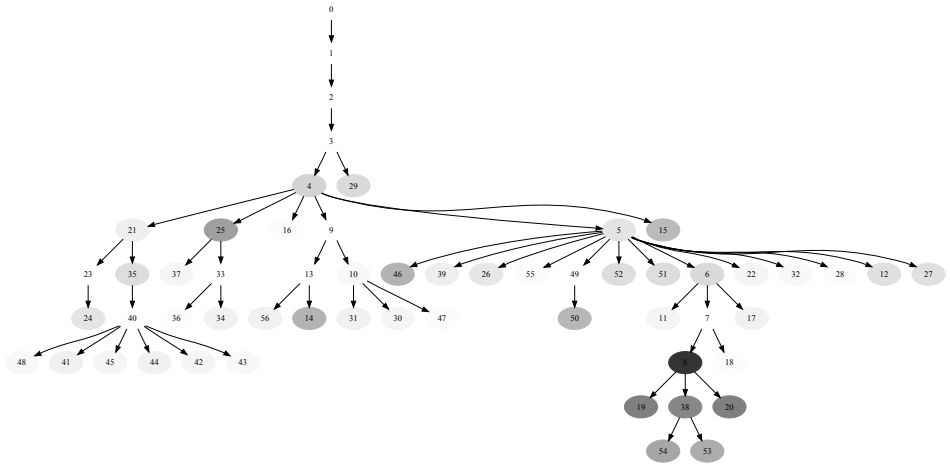


Fig. 5. Folder tree from a real user. Each node of this tree denotes a folder. Darker nodes denote the folders that have been accessed more often. The number on each node is the identification number of the folder. The root node marked with "0" denotes "My Computer".

up to an ancestor folder and then go back down to the correct child. On the other hand, predicting a higher level folder, such as "Presentations" or "Homeworks" or even "CS534", may not be a perfect hit, but it may reduce the cost in half — the student only needs to go downward to reach his destination folder.

Figure 5 shows the folder tree of a real user who has been using FolderPredictor for approximately a month. The folders shown here are only the ones that were accessed by this user during this period and their ancestors. The darkness of a node in this tree is proportional to how frequently the corresponding folder was accessed. There are quite a few cases where predicting the parent folder is better than predicting a child folder. One example is folder #5. It has 13 child folders, most of which were accessed. Another example is folder #40. It wasn't directly accessed by the user, but it has 6 child folders that were accessed by the user with approximately equal frequency.

Furthermore, we believe that incorrectly predicting a leaf node will be on average more frustrating for users than picking an ancestor node that is an incomplete path

to the desired folder. For example, in Figure 4, if the user is going to access the leaf folder "Part1" and FolderPredictor predicts the parent folder "Presentations" and initializes the file open dialog box with it, the user will be able to see the folder "Part1" when the dialog box is displayed and thus easily jump to it. On the other hand, if the dialog box is initialized with a leaf node such as "Part2", it will be cognitively harder for the user to recognize that "Part2" is a sibling of the target folder "Part1", because there is no visual cue of this.

4.3 Cost-Sensitive Prediction Algorithm

Based on the ideas presented above, we developed the following Cost-Sensitive Prediction (CSP) algorithm shown in Figure 6.

Input: A finite set $\mathsf{F} = \{f_1, f_2, \ldots, f_m\}$, where f_i is a folder with a positive weight w_i, $1 \le i \le m$.
Output: Three recommended folders a_1, a_2 and a_3(descending in predicted preference).

```
1  for i ← 1 to m do        /* normalize weights to get probability distribution */
2  |  p_i = w_i / Σ_{j=1}^{m} w_j;
3  end
4  H = ∅;
5  for i ← 1 to m do /* construct the candidate set by adding ancestor folders */
6  |  H_i ⇐ all ancestors of f_i, including f_i itself;
7  |  H = Union (H, H_i);
8  end
9  forall folder h in H do
10 |  forall folder f in F do
11 |  |  dis[h, f] = TreeDistance (h, f);
   |  |                                  /* tree distance from h to f, in clicks */
12 |  end
13 end
14 {a_1, a_2, a_3} =
   arg min_{a_1,a_2,a_3} Σ_{i=1}^{m} p_i × min{dis[a_1, f_i], dis[a_2, f_i] + 1.0, dis[a_3, f_i] + 1.0};
```

Fig. 6. Cost-Sensitive Prediction (CSP) Algorithm.

The goal of the CSP algorithm is to find three folders from the candidate set that jointly minimize the expected number of clicks required to reach the user's target folder. The candidate set `H` here consists of the folders that the user has accessed before (set `F`) and their ancestor folders. The probability distribution of the user's target folder is estimated by normalizing the weights of the folders in `F`, which are collected as described in Section 4.1.

In line #14 of the CSP algorithm, the expected cost (in the number of clicks) of reaching a folder f_i from a prediction $\{a_1, a_2, a_3\}$ is computed as

$$\min\{dis[a_1, f_i], dis[a_2, f_i] + 1.0, dis[a_3, f_i] + 1.0\},$$

because of the following facts:

(1) a_1 will be set as the default folder of the open/save file dialog. This means that the user will be in folder a_1 with no extra "click" required. Therefore, $dis[a_1, f_i]$ is the cost if the user navigates to f_i from a_1.

(2) a_2 and a_3 will be shown as shortcuts to the corresponding folders in the "places bar" on the left side of the open/save file dialog box (see Figure 3). The user must execute one "click" on the places bar if he wants to navigate to f_i from a_2 or a_3. Therefore, an extra cost 1.0 is added. (In fact, this doesn't affect which three folders are chosen, but it will order them to make sure a_1 is the best folder to predict.)

(3) We assume the user knows which folder to go to and how to get there by the smallest number of clicks. Therefore, the cost of a prediction $\{a_1, a_2, a_3\}$ is the minimum of $dis[a_1, f_i]$, $dis[a_2, f_i] + 1.0$, and $dis[a_3, f_i] + 1.0$.

FolderPredictor runs this algorithm whenever the user switches task and whenever the folder weights are updated. The three predicted folders are then displayed in the file dialog boxes and the Windows Explorer.

4.4 Variations of the CSP Algorithm

During our initial deployment of FolderPredictor, we observed that, although FolderPredictor largely reduces the cost of reaching the right folder, the approach of predicting ancestor folders that lead to multiple possible folders tends to make the prediction less "perfect": the top predicted folder sometimes is close to but not exactly the same as the user's target folder. This observation inspired us to consider several variations of the CSP algorithm that may increase the chance of making a perfect prediction.

The first variation is to make FolderPredictor more "aggressive": choosing the most probable folder (the folder with the highest weight among the folders that the user has accessed before) as the top prediction (a_1). The other two predicted folders are choosen by running the CSP algorithm with the top prediction fixed. That is, line #14 of the CSP algorithm in Figure 6 is changed to be

$$\{a_2, a_3\} = \arg \min_{\{a_2,a_3\}} \sum_{i=1}^{m} p_i \times \min\{dis[a_1, f_i], dis[a_2, f_i] + 1.0, dis[a_3, f_i] + 1.0\}.$$

By doing this, the top predicted folder will tend to be a "leaf" folder that contains the files that the user is most-likely to access. Therefore, it may increase the chance that FolderPredictor makes a perfect prediction, but once the top predicted folder is wrong, it may cost more clicks for the user to reach the target folder. For example, in the scenario shown in Figure 4, the first prediction of this "aggressive" CSP algorithm will be the leaf folder with the highest weight, while the first prediction of the original CSP algorithm is probably one of the parent folders.

The second variation is to make use of the most recently-used (MRU) folder. Microsoft Windows usually initialize the file open/save dialog boxes with the MRU folder. This approach works fairly well and in our experiments presented in the next section, it perfectly hits the target folder about 53% of the time. Therefore, it's worth combining it into our predictions — let the MRU folder be the top prediction (a_1) and choose the other two predicted folders by running the CSP algorithm with the top prediction fixed.

There are two different types of MRU folder: the system-wide MRU folder and the application-specific MRU folder. The application-specific MRU folder is the last folder accessed by the user using the same application. For example, the

application-specific MRU folder for Microsoft Word is the folder containing the file that was most recently opened or saved by Microsoft Word. We store one application-specific MRU folder for each application. On the other hand, the system-wide MRU folder is the last folder accessed by the user, no matter which application was used. It is hard to say which type of MRU folder is better, so we implemented both and compared them in our experiments.

The third variation is slightly different from the second one. A potential problem with always predicting the MRU folder as the top prediction is that when there is a task switch, the MRU folder may be a bad choice. Therefore, we want to apply the original CSP algorithm to predict for the first file access within a task episode (period between two task switches), and apply the second variation described above to predict for the remainder of the task episode. A "task episode" for task T is the segment of time in which the user is working on task T. It starts with a "TaskBegin" message, when the user indicates he/she is starting to work on task T, and it ends with another "TaskBegin" message when the user switches to a different task.

Table I summarizes the folder prediction algorithms we have described above.

Table I. Summary of the folder prediction algorithms.

Name	Description
CSP	Cost-Sensitive Prediction algorithm described in Section 4.3.
ACSP	Use the most-probable folder as top prediction and run CSP to get the other two folders.
SMCSP	Use the system-wide MRU folder as top prediction and run CSP to get the other two folders.
SMCSP2	Use CSP for the first file access within a task episode, and use SMCSP for the remainder of the task episode.
AMCSP	Use the application-specific MRU folder as top prediction and run CSP to get the other two folders.
AMCSP2	Use CSP for the first file access of each application within a task episode, and use AMCSP for the remainder of the task episode.

5. EXPERIMENTAL EVALUATIONS

To measure the effectiveness of FolderPredictor and compare the performance of different folder prediction algorithms, we have conducted two user studies. The goal of the first user study was to measure how many clicks FolderPredictor with the CSP algorithm saves, by comparing FolderPredictor's predictions with *Windows Default* (without folder predictions). The goal of the second user study was to compare the performance of the various folder prediction algorithms listed in Table I. One thing worth mentioning is that, although FolderPredictor needs to know the user's current task, the task switches specified by the users are noisy by nature. We do not ask the study participants to clean them up, and simply make folder predictions using the noisy current task labels.

5.1 Comparing the CSP Algorithm with Windows Default

The first user study was conducted at Oregon State University. The participants were two professors and two graduate students. The TaskTracer system (with FolderPredictor) was deployed on the computers that the participants performed

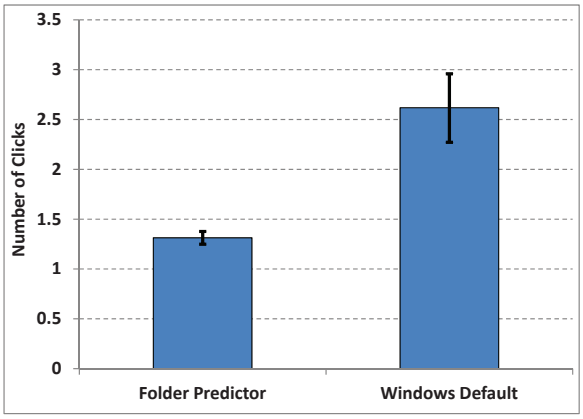


Fig. 7. Average cost of FolderPredictor and Windows Default, in number of clicks required to reach the target folder for each file open/save.

their daily tasks on. Every participant ran the TaskTracer system for a fairly long time (from 4 months to 1 year). The algorithm employed in this version of FolderPredictor was the CSP algorithm. The discount factor α was set to 0.85.

At the end of this user study, we collected four data sets as shown in Table II. Each data set is a list of predictions that FolderPredictor made for a user, ordered by time. Each prediction is marked with the name of the task that was active at the moment this prediction was made. The **Set Size** is the number of predictions contained in the data set.

Table II. Information about the data sets collected in the first user study. Set size is the number of Open and SaveAs events where a prediction is made and its effectiveness is measured.

#	User Type	Data Collection Time	Set Size
1	Professor	12 months	1748
2	Professor	4 months	506
3	Graduate Student	7 months	577
4	Graduate Student	6 months	397

5.1.1 *Average Cost.* Figure 7 compares the average cost (in "clicks") of the FolderPredictor and the Windows Default on all four data sets. The cost of the Windows Default is the distance between the original default folder of the file dialog and the destination folder. In other words, the cost of Windows Default is the user cost when FolderPredictor is not running. The figure also shows 95% confidence intervals for the costs.

Statistical significance testing shows that FolderPredictor surely reduces the user cost of locating files (P-value $= 1.51 \times 10^{-29}$ using an ANOVA F-test). On average, the cost is reduced by 49.9% when using FolderPredictor with the CSP algorithm.

5.1.2 *Distribution of Costs.* Figure 8 shows a histogram of the number of clicks required to reach the target folder under the Windows Default and the FolderPredictor. We can see from the figure that:

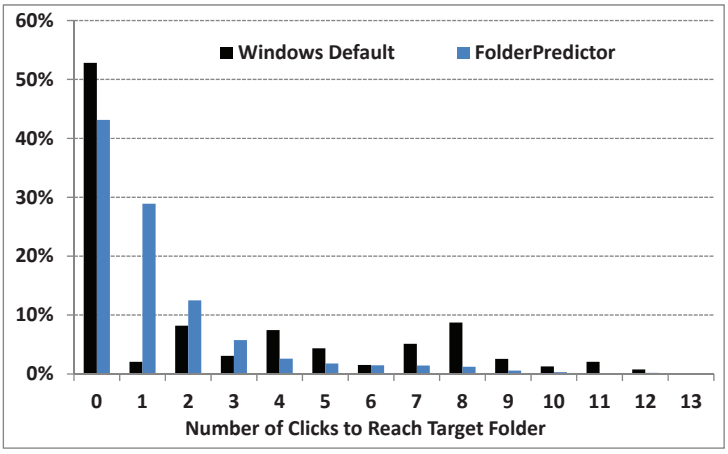


Fig. 8. Histogram of the number of clicks required to reach the target folder: Windows Default v.s. FolderPredictor.

(1) About 90% of the FolderPredictor's costs are less than or equal to three clicks. Only a small fraction of the FolderPredictor's costs are very high.
(2) Although about half of the Windows Default's costs are zero (meaning that the user is in the right folder at the moment the file dialog box is opened), about 40% of the Windows Default's costs are above three.

This means that FolderPredictor not only reduces the overall average cost of locating files, but also decreases the probability of encountering very high costs in locating files.

It is interesting to see that Windows Default actually gets the default folder perfectly correct more than FolderPredictor. This most likely happens because Windows typically picks a leaf folder (i.e., the most recently-used folder) as the default folder. FolderPredictor sometimes plays it safe and picks an ancestor folder that is more likely to be close to multiple possible folders and less likely to be perfect. Thus we see a large number of cases relative to the Windows Default where FolderPredictor is one, two, or three clicks away. In fact, this discovery inspired us to come up with algorithm variations described in Section 4.4.

5.1.3 *Learning Curve.* Intelligent systems usually perform better as more training data is acquired. In FolderPredictor's case, the training data are the user's opens and saves per task. For each open/save, FolderPredictor makes a prediction and uses the user's actual destination folder to update itself. Therefore, the cost of the folders recommended by FolderPredictor should decrease as more predictions are made for a task. To evaluate this, we computed the learning curve for FolderPredictor shown in Figure 9.

In the figure, the X-axis is the number of predictions within one task aggregated into ranges of width 10, and the Y-axis is the average cost of one prediction within this range. For example, the first point of the curve shows the average cost of all predictions between (and including) the 1st and 10th predictions of all tasks. The figure also shows 95% confidence intervals for the average costs.

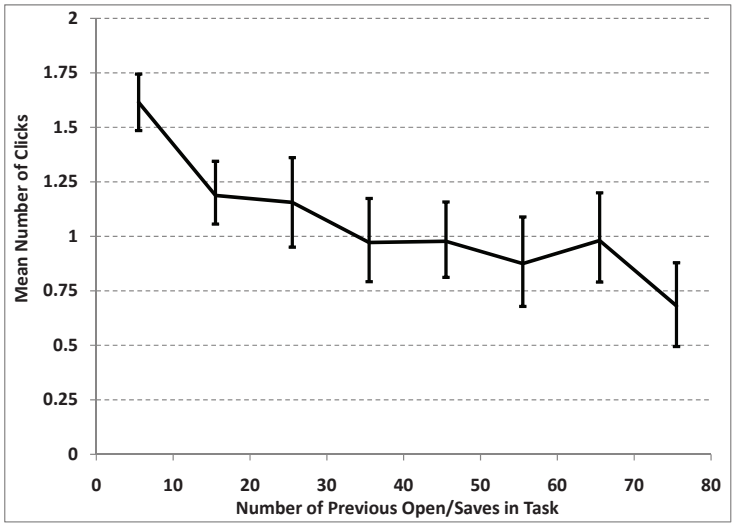


Fig. 9. Learning curve of FolderPredictor. The average number of clicks per file open/save decreases as FolderPredictor sees more file accesses with a task.

We want to point out that FolderPredictor does not need a separate training phrase, it benefits the user from the beginning. For a new task, the first prediction is the same as the Windows Default, because FolderPredictor knows nothing about this task yet. But as the user opens/saves files while working on this task, FolderPredictor predicts and learns simultaneously. The curve shows that the cost decreases as more predictions made. The average cost decreases from 1.6 (first 10 predictions) to 0.7 (71st to 80th predictions).

5.2 Comparing Different Folder Prediction Algorithms

The second user study was conducted at Intel Corporation. The participants were knowledge workers (managers and administrative assistants) who perform their daily work on the computers. During a 4-week period, a software system called Smart Desktop (Beta 3), which is a commercialized version of TaskTracer, was running on their computers. Smart Desktop incorporates a version of the FolderPredictor but, more importantly, it collects all of the information needed to evaluate alternative folder prediction algorithms. At the end of the study, we ran an analysis program on the data to evaluate the cost of each folder prediction algorithm by simulating the algorithm at each File Open/SaveAs event in time order.

Seven participants completed the study. One of them is excluded from the analysis because the number of file opens/saves is too low (< 20) in his data. Table III shows the information about the data sets collected from the remaining 6 participants. The set size is the number of File Open/SaveAs events (i.e., the number of evaluation points) in the data set.

5.2.1 *Average Cost of Different Algorithms.* Figure 10 compares the average cost (in "clicks") of different folder prediction algorithms over all study participants. The figure also shows 95% confidence intervals for the costs.

Table III. Information about the data sets collected in the second user study. Set size is the number of Open and SaveAs events where a prediction is made and its effectiveness is measured.

#	Data Collection Time	Set Size
s1	4 weeks	138
s2	4 weeks	99
s3	4 weeks	395
s4	4 weeks	146
s5	4 weeks	102
s6	4 weeks	352

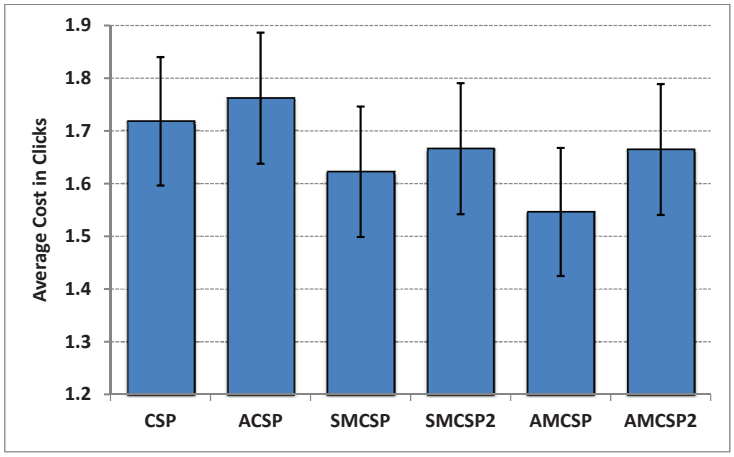


Fig. 10. Average cost of different folder prediction algorithms over all Intel participants, in number of clicks required to reach the target folder for each file open/save.

There are several interesting observations from this figure:

(1) The ACSP algorithm performs worse than the original CSP algorithm. This means that predicting the most probable folder as top prediction actually increases the overall cost of the prediction.

(2) Algorithms using an MRU folder as the top prediction (SMCSP, SMCSP2, AMCSP, AMCSP2) have better performance than the original CSP algorithm.

(3) The SMCSP2/AMCSP2 algorithms are worse than the corresponding SMCSP/AMCSP algorithms. This is a little surprising, because we thought that the MRU folder would not be good for the first file access after a task switch. One possible explanation is that different tasks may share the same folders, especially for two tasks that are worked on sequentially in time. Another factor is that the users may not declare task switches accurately. Anecdotally, users report that they often don't realize that they have forgotten to declare a task switch until they perform an Open or SaveAs and notice that the folder predictions are from the "wrong" task.

(4) The AMCSP algorithm has the best performance among all the algorithms.

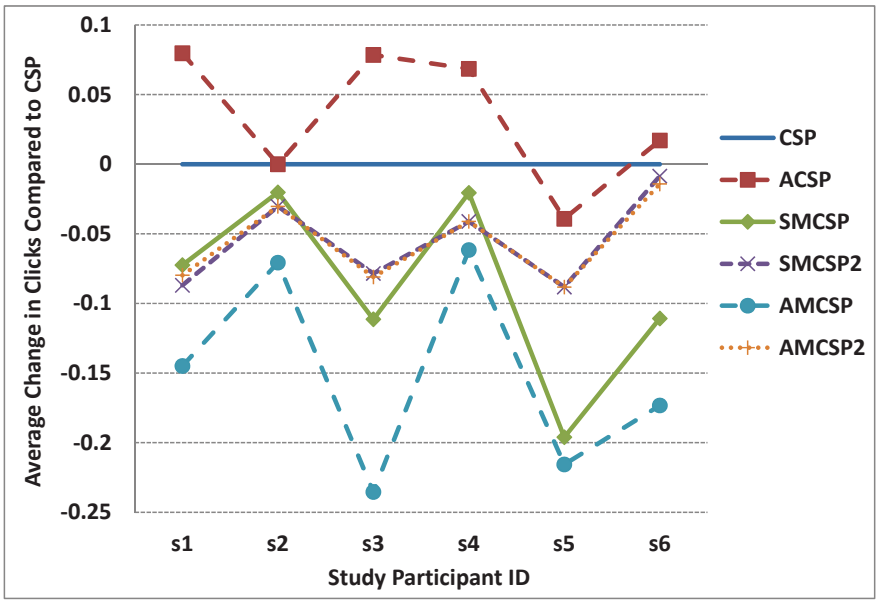


Fig. 11. Average cost of different folder prediction algorithms for each participant, in number of clicks required to reach the target folder for each file open/save.

The differences among the average costs of these algorithms are generally not statistically significant. However, the difference between the average cost of the AMCSP algorithm (the best one) and that of the original CSP algorithm is statistically significant with $p\text{-}value = 0.05$ assessed with a paired t-test. On average, the AMCSP algorithm further reduces the cost of FolderPredictor by 10%, compared with the original CSP algorithm.

5.2.2 *Comparison of Algorithms across Different Users.* Different users behave differently in folder organization and file access. We wanted to investigate how the folder prediction algorithms compare with each other for different participants. Figure 11 illustrates the average cost of the folder prediction algorithms across different participants. In this figure, the cost of the CSP algorithm is used as baseline, and the values shown are the change in the number of clicks compared to the CSP algorithm. The horizontal axis lists the ID's of our study participants.

The figure clearly shows that the AMCSP algorithm is consistently the best algorithm across all participants. Note that all of the other observations we discussed in Figure 10 are also valid for each study participant.

5.2.3 *Distribution of Costs.* To investigate why the AMCSP algorithm outperforms the CSP algorithm, we compared the distribution of costs of these two algorithms, as shown in Figure 12.

As illustrated in the figure, both algorithms have the desired property of reducing the probability of encountering very high costs in locating files. The main advantage of the AMCSP algorithm is that it makes more perfect predictions (zero clicks) than the CSP algorithm, which validates our hypothesis that using the MRU folder as the

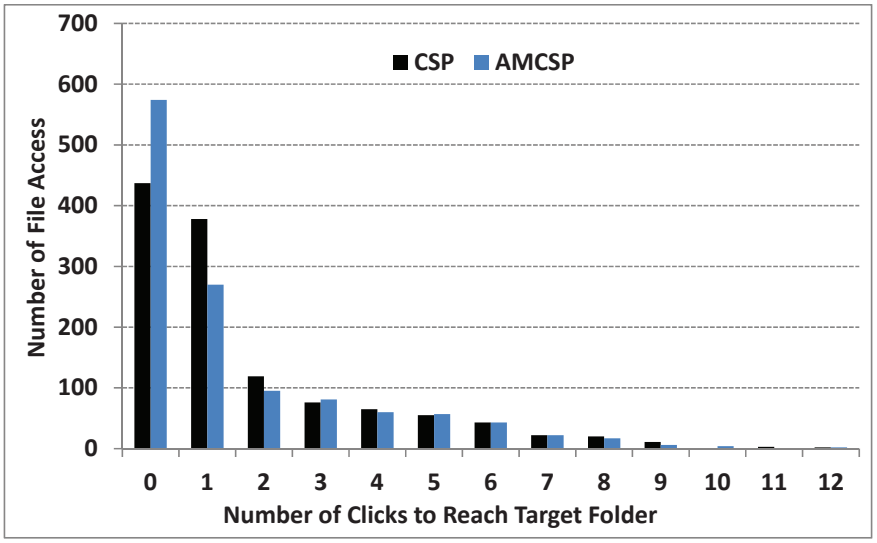


Fig. 12. Histogram of the number of clicks required to reach the target folder: CSP algorithm v.s. AMCSP algorithm.

top prediction can increase the chance of making perfect predictions. If we sum up the number of file accesses with cost of 0, 1 and 2 clicks, this value is approximately equal for the two algorithms. Therefore, the reason why the AMCSP algorithm outperforms the CSP algorithm is that the AMCSP algorithm replaces some of CSP's 1- and 2-click predictions with perfect predictions.

6. RELATED WORK

On the surface, the FolderPredictor approach is similar to some email classification systems that predict possible email folders based on the text of incoming email messages [Rennie 2000; Segal and Kephart 1999; 2000]. In particular, both MailCat and FolderPredictor present their top three predicted folders to the user [Segal and Kephart 1999]. However our approach is different. There are many different sources of information that one could use to predict the resource that the user wants to access, e.g., content similarity, activity patterns, link structures, etc. Our approach focuses on user activities (given task information), whereas many of the email classification systems focus on textual content. Building a text-based classification system for file folder prediction might be more challenging because:

(1) for most users, there are many more file folders than email folders, with much more complex hierarchy;
(2) there are many different types of files under file folders and it is hard to extract text from some of them;
(3) latency introduced by extracting and analyzing texts from files are not desired in a real time application like FolderPredictor;
(4) files with similar keywords are not necessarily in the same folder, or even close to each other (e.g., the class I taught last year versus the class I am teaching

this year).

There are also some software tools that help users to quickly locate their files in the open/save file dialog boxes, e.g. Default Folder X [2009] for Mac and Direct Folders [2009] for Windows. These tools make the open/save file dialog boxes more configurable and comprehensive to the user. The user can put more shortcuts in the dialog, as well as define a default folder for each application. However, these tools cannot adjust the shortcuts and default folders automatically based on the context of the user's activities. On the other hand, our approach makes intelligent folder predictions based on the user activities within each task. Therefore, FolderPredictor can be a good complement to these tools.

7. DISCUSSION AND CONCLUSION

To reduce user cost for accessing files, we applied intelligent prediction methods to records of user activity to recommend file system folders. We described the FolderPredictor, which reduces the user's cost for locating files in folder hierarchies. FolderPredictor applies a cost-sensitive online prediction algorithm to the user's activities. Experimental results show that, on average, FolderPredictor reduces the cost of locating a file by 50%. Perhaps even more importantly, FolderPredictor practically eliminates cases in which a large number of clicks are required to reach the right folder. We also investigated several variations of the cost-sensitive prediction algorithm and presented experimental results showing that the best folder prediction combines an application-specific MRU folder with two folders computed by our cost-sensitive prediction algorithm. An advantage of FolderPredictor is that it does not require users to adapt to a new interface. Its predictions are presented directly in the open/save file dialogs. Users are able to easily adapt to exploit these predictions.

The folder prediction algorithms presented in this paper (CSP and its variants) are very simple yet effective. We have considered investigating more complex machine learning techniques, but there are two reasons that make us believe our current approach is more appropriate. First, simple, fast and easy-to-understand algorithms are critical to designing and deploying real-world applications, like FolderPredictor, that require instantaneous predictions. Second, our simple algorithms learn quickly and typically reduce the number of clicks to only 1 or even less than 1 (see the learning curve in Figure 9). This means there is not much room for improvement. Hence, we don't see any benefit to implementing more complex algorithms.

The results reported in this paper are likely an underestimate of the value of the FolderPredictor. One of the key assumptions of this paper is that the user always knows which folder they want to get to and where it is located — in such cases, we have demonstrated that FolderPredictor will get them there faster. The reality is that people have limited memory, and highly multitasking users often cannot maintain in their memory the locations of files for all of their tasks, particularly tasks that they have not worked on recently. Thus users may need many more clicks to "search" for the right folder. By default, Windows only remembers what was worked on most recently, regardless of task. FolderPredictor on the other hand remembers multiple folders used on each task, regardless of how long ago the task was last active. Thus FolderPredictor's recommendations can help remind the user

where files related to a task have been stored.

There are other psychological benefits of FolderPredictor that are harder to evaluate. For example, being placed consistently in the wrong folder can generate frustration, even if the click distance is not far. At this stage, all that we have is qualitative evidence of this. During the deployments of FolderPredictor, multiple participants reported becoming "addicted" to FolderPredictor — they were distressed when they had to use computers that did not have TaskTracer installed. They also reported that they did a better job of notifying TaskTracer of task switches in order to ensure that the FolderPredictor recommendations were appropriate for the current task.

One interesting future research direction is to create an "unsupervised" version of FolderPredictor. The FolderPredictor discussed in this paper is "supervised" — the user is asked to declare what task they are working on at each point in time. However, we find from user feedback that declaring the current task is the biggest overhead of the TaskTracer system. Therefore, we have started to investigate ways on making folder predictions without knowing what task the user is working on. Initial results look very promising — our "unsupervised" FolderPredictor shows performance close to the "supervised" version. The ultimate goal is to create standalone FolderPredictor software that runs silently in the background and provides folder predictions to the user without requiring any additional user interaction.

ACKNOWLEDGMENTS

The authors thank the participants of our two user studies for their willingness to use the TaskTracer and Smart Desktop software for extended periods of time. The authors also thank the many members of the TaskTracer research team with special thanks to Jon Herlocker, who founded the TaskTracer project and provided the initial vision. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA), through the Department of the Interior, NBC, Acquisition Services Division, under Contract No. NBCHD030010 and by DARPA, through Air Force Research Lab Contract No. FA8750-07-D-0185/0004. Additional support was provided by Intel Corporation and by the National Science Foundation under grant IIS-0133994.

REFERENCES

Barreau, D. K. and Nardi, B. 1995. Finding and reminding: File organization from the desktop. *ACM SIGCHI Bulletin 27,* 3, 39–43.

Boardman, R. and Sasse, M. A. 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*. 583–590.

Budzik, J. and Hammond, K. J. 2000. User interactions with everyday applications as context for just-in-time information access. In *Proceedings of the 4th International Conference on Intelligent User Interfaces*. ACM, 44–51.

Default Folder X. 2009. St. Clair Software. http://www.stclairsoft.com/defaultfolderx/

Direct Folders. 2009. Code Sector Inc. http://www.codesector.com/directfolders.asp

Dourish, P., Edwards, W. K., LaMarca, A., Lamping, J., Petersen, K., Salisbury, M., Terry, D. B., and Thornton, J. 2000. Extending document management systems with user-specific active properties. *ACM Transactions on Information Systems 18,* 2 (January), 140–170.

Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L., and Herlocker, J. L. 2005. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Procceedings of 9th International Conference on Intelligent User Interfaces*. 75–82.

Dumais, S., Cutrell, E., Cadiz, 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 the 26th Annual International ACM SIGIR conference on Research and Development in Informaion Retrieval*. 72–79.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. 1998. The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. 256–265.

Jones, W., Phuwanartnurak, A. J., Gill, R., and Bruce, H. 2005. Don't take my folders away! Organizing personal information to get things done. In *CHI-05 extended abstracts on Human factors in Computing Systems*. 1505–1508.

Jul, S. and Furnas, G. W. 1995. Navigation in electronic worlds: Workshop report. *ACM SIGCHI Bulletin 29,* 2, 44–49.

Ko, A., Aung, H. H., and Myers, B. 2005. Eliciting design requirements for maintenance-oriented ides: A detailed study of corrective and perfective maintenance tasks. In *Proceedings of the International Conference on Software Engineering*. 126–135.

Maes, P. 1994. Agents that reduce work and information overload. *Communications of the ACM 37,* 7, 30–40.

Pietrek, M. 1995. *Windows 95 System Programming Secrets*. IDG Books Worldwide, Inc., Foster City, CA 94404, USA.

Rennie, J. 2000. ifile: An application of machine learning to e-mail filtering. In *KDD 2000 Workshop on Text Mining*. Boston, MA.

Rhodes, B. 2003. Using physical context for just-in-time information retrieval. *IEEE Transactions on Computers 52,* 8, 1011–1014.

Segal, R. B. and Kephart, J. O. 1999. Mailcat: An intelligent assistant for organizing e-mail. In *Proceedings of the Third International Conference on Autonomous Agents*. ACM Press, 276–282.

Segal, R. B. and Kephart, J. O. 2000. Incremental learning in swiftfile. In *Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, 863–870.

Shen, J., Irvine, J., Bao, X., Goodman, M., Kolibaba, S., Tran, A., Carl, F., Kirschner, B., Stumpf, S., and Dietterich, T. G. 2009. Detecting and correcting user activity switches: algorithms and interfaces. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*. ACM, 117–126.

Stumpf, S., Bao, X., Dragunov, A., Dietterich, T. G., Herlocker, J., Johnsrude, K., Li, L., and Shen, J. 2005. Predicting user tasks: I know what you're doing! In *AAAI-05 Workshop on Human Comprehensible Machine Learning*.

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 the SIGCHI conference on Human factors in Computing Systems*. 583–590.

Received May 2010;