Discovering Frequent Work Procedures From Resource Connections

Jianqiang Shen, Erin Fitzhenry, Thomas G. Dietterich
1148 Kelley Engineering Center, School of EECS, Oregon State University
Corvallis, OR 97331, U.S.A.
{shenj, fitzheer, tgd}@eecs.oregonstate.edu

ABSTRACT

Intelligent desktop assistants could provide more help for users if they could learn models of the users' workflows. However, discovering desktop workflows is difficult because they unfold over extended periods of time (days or weeks) and they are interleaved with many other workflows because of user multi-tasking. This paper describes an approach to discovering desktop workflows based on rich instrumentation of information flow actions such as copy/paste, SaveAs, file copy, attach file to email message, and save attachment. These actions allow us to construct a graph whose nodes are files, email messages, and web pages and whose edges are these information flow actions. A class of workflows that we call *work procedures* can be discovered by applying graph mining algorithms to find frequent subgraphs. This paper describes an algorithm for mining frequent closed connected subgraphs and then describes the results of applying this method to data collected from a group of real users.

ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*Graphical user interfaces*; I.2.1 Artificial Intelligence: Applications and Expert Systems—*Office automation*

Author Keywords

Intelligent interfaces, workflow, provenance, data mining, automated assistance, resource management

General Terms

Design, Human Factors, Experimentation.

INTRODUCTION

Knowledge workers frequently change tasks, either by choice or through interruptions [22]. With an increased number of tasks and task switches, it becomes more and more difficult for knowledge workers to keep track of the state of each of their activities. Many attempts have been made to develop effective ToDo managers, but these impose additional overhead on desktop work and they do not capture many of the user's activities [3]. One way to reduce this overhead and increase the coverage of a ToDo manager would be if the computer could automatically recognize which activities were being executed and track their status in an intelligent ToDo manager.

In our work on the TaskTracer system [7, 31], we took an initial step in this direction by applying machine learning methods to predict the current activity of the user based on the resources (files, web pages, email messages) that the user accesses. This approach requires that the user explicitly name each of his/her activities, which imposes additional overhead and is more appropriate for larger chunks of activity such as long-lived projects. An additional shortcoming of this approach is that it lacks an understanding of the *state* of the activity, so it can't tell whether a task is completed or ongoing, nor can it provide useful reminders to the user about actions that still need to be performed to complete the task. So while the TaskTracer approach is very valuable for helping the user recover from interruptions of long-duration activities, it does not provide any support for managing to-do lists.

A significant portion of the desktop activity of knowledge workers involves executing workflow procedures. These can be formally prescribed procedures (e.g., submitting requests for travel authorization) or informal, idiosyncratic procedures (e.g., keeping track of fantasy football results). Suppose that the computer had models of these workflows. With such models, it would be able to detect when a new workflow instance was initiated, track the current state of execution of each workflow instance, and determine when the execution was complete. This would allow it to automatically populate and maintain an intelligent ToDo list for the knowledge worker.

If the workflow models were sufficiently precise, they could also provide the basis for partial automation of workflows. For example, consider the simple workflow "provide comments on a document". In this workflow, a document arrives as an email attachment. The user saves the document into an appropriate folder, edits it (optionally saving it under a new name), and then attaches it to a reply email. A smart desktop assistant might be able to recognize the start of the workflow by analyzing the initial email message. In which case, it could place an entry on the ToDo list. When the user was ready to work on this action item, the assistant could provide a one-click way of opening the email message, saving the attachment, and opening the file. Later, after the file had been edited and saved, it could provide one-click automation for opening the email message, initiating a reply, and attaching the file to the reply message. These simple automation steps could avoid the need for the user to find the old email message, remember where the file was saved, and so on. We hypothesize that such support could avoid the need to resort to desktop search tools to remember and re-access such "lost" items.

While this vision of an intelligent workflow assistant is very attractive, it has one critical flaw: how can the computer acquire detailed models of desktop workflows? Some researchers have addressed this problem by providing easy ways for users to define the workflow procedures themselves [20]. We propose instead to automatically discover the workflows by observing the user's desktop activity and detecting repeated sequences of actions. This is very challenging for a number of reasons. First, these workflows take place over extended periods of time (days, weeks, months). Second, they are interleaved with thousands of irrelevant actions that either do not belong to any workflow or belong to other workflows. Indeed, for a workflow such as "Review conference paper", many instances of that workflow will be executing simultaneously. How can we detect and untangle these interleaved instances?

In this paper, we present an approach based on capturing information flow actions (also called provenance links [27, 30, 12]). We define a *work procedure* as a directed graph whose nodes are resources (files, email messages, web pages) and whose arcs are actions such as SaveAs, copy/paste, save email attachment, attach file to email, upload file to web page, download file from web page, and so on. With sufficient instrumentation of desktop applications, we can capture the complete set of these provenance links. This allows us to capture the user's desktop behavior as a large graph of resources and provenance links. Each work procedure instance is a connected subgraph in this graph. To discover these instances, we extend an existing subgraph mining algorithm to find frequently occurring subgraphs.

The information flow graph solves the problem of temporally-extended interleaved work procedures. The provenance links in the graph are the same regardless of whether the work procedure was executed in a single day or over many months. And the topology of the graph does not depend on how multiple work procedure instances were interleaved. This leads to a powerful savings in computation. Consider two work procedure instances each involving 8 actions. There exist $\binom{16}{8} = 12,870$ possible interleavings of these actions, but there is only one information flow graph.

To acquire the information flow graph, we employed the TaskTracer system [7, 31]. TaskTracer already tracks every resource (file, folder, web page, email message, email contact) accessed by the user, so these provide the nodes in the graph. To acquire the provenance links, we extended TaskTracer to capture several kinds of provenance links (File copy, File rename, Copy/Paste, SaveAs, Save Email Attachment, Attach to Email, Download from web page, Upload to web page). This does not capture all of the relevant provenance links, so for this paper we performed additional automatic analysis of the user's resources to identify additional links as described below.

The remainder of the paper is structured as follows. The next section reviews related work. Then, we motivate the work procedure discovery problem with some typical work procedure cases and discuss its potential usage. Next, we provide a brief overview of the activity management system used for our research and present a user interface for displaying the provenance information. We describe our methods for building the information graph, for finding frequent generalized closed patterns, and for assigning appropriate action sequences. The last section presents experimental results on the accuracy of the approach.

RELATED WORK

Several other researchers have worked on problems similar to the one we address in this paper.

Business workflow mining. Traditionally, a *workflow* is defined as a partial or total automation of a business process in which a collection of activities must be executed by humans or machines according to certain procedural rules [1, 23, 14, 10]. Workflow management systems help to execute, monitor and manage work process flow and execution. They (partially) automate the definition, creation, execution, and management of work processes through the use of software. Their transaction logs record information of each executed process. The activity of using computer software to examine theses records and deriving various structural data results is called workflow mining [10].

Our work procedure discovery problem can be thought as a special case of the workflow mining problem with two differences: (a) the traditional workflow mining problem tries to find a model expressing the business process of an organization and the execution of a model usually involves multiple people, while we focus on individual knowledge workers and the discovered model is executed by a single user; (b) in the traditional workflow mining problem, the instances of the workflows are assumed to already be identified, whereas in our work we must discover the instances first.

Motif discovery. Another related line of work is motif discovery in biological sequence data and continuous time-series data [21, 5, 32, 15, 24, 25]. A *motif* is a frequently-appearing pattern. A typical motif discovery approach first applies a sliding window to divide the temporal data into a set of subsequences, and then builds a similarity matrix among these subsequences. This matrix is then employed to select seed motif occurrences and locate additional motif occurrences. Motif discovery is useful for various time-series data mining tasks such as mining association rules in time-series data, building time series classification, anomaly/interestingness detection, etc [5].

A motif occurrence corresponds to a subsequence of the time-series data. Every event point between the start point and the end point belongs to that motif occurrence. This is very different from the work procedures that we seek to discover, which are interleaved and intermixed with unrelated (irrelevant) actions.

Desktop activity modeling. To build intelligent personal assistants to help users organize their work, people have tried to automatically extract understandable descriptions of a desktop user's activities [9, 26, 2, 11]. Such descriptions can help researchers understand users' behavior and design smarter intelligent agents. However, unlike our work procedures, these descriptions do not support (semi-) automated execution of the discovered activities. Instead, each activity is usually represented as a set of objects (people, documents) or as a cluster of action sequences. Many of these approaches assume prior knowledge about the number of activities and their duration.

Email activity management. Email activity management systems help knowledge workers manage their daily work life by tracking email exchanges. The work by Bellotti, et al. [3], supports the manual population of an activity template from emails. Dredze, et al. [8] classify email messages into activities by analyzing their content. The work of Kushmerick, et al. [18] is quite similar to our work in that they also attempt to discover workflows and track the status of those workflows. However, they focus on e-commerce transactions where the email messages involve unique identifiers (e.g., order numbers; ebay item numbers), which simplies the problem.

WORK PROCEDURE DISCOVERY PROBLEM

Definitions

We now formally define the work procedure discovery problem.

DEFINITION 1. *A resource is a data object such as a file, email message, or web page.*

In our modified TaskTracer system, each resource is assigned a unique identifier the first time it is accessed. There are several subtleties involved in handling ephemeral resources such as email messages that are currently being composed or MS Office documents that have not yet been saved ("Document1", "Presentation2").

DEFINITION 2. *An information flow action (also called a provenance link) is a primitive or composite desktop action that directly links two resources.*

We directly instrumented the following information flow actions:

- Copy/paste. This involves instrumenting both the Copy event (to capture the resource from which the information is copied) and the Paste event (to capture the resource into which the information is pasted).
- Attach file to email. In order to capture the name of the file on disk, we added our own button to the MS Outlook user interface.
- Save email attachment. In order to capture the name of the saved file on disk, we added our own button to the MS Outlook user interface.
- Download file from web page. We wrote our own download manager as a plugin to Internet Explorer.
- Upload file to web page.
- Copy or Rename file in Windows Explorer.

DEFINITION 3. *An* information flow graph *is a labeled directed graph where the nodes are resources and the arcs are information flow actions. If there are multiple information flow actions linking two nodes, the graph contains a single arc labeled with the set of corresponding information flow actions.*

The information flow graph is the raw graph of resources and actions constructed from observing the user's desktop activity. Note that there may be multiple actions connecting two resources. For example, suppose the user performs a SaveAs and then accidentally deletes part of a document. He might copy/paste that part from the previous version. Another simple example is that there are often multiple copy/paste actions from one resource to another.

After the basic information flow graph is constructed, it is augmented with additional arcs representing chain relations:

DEFINITION 4. *A* resource-action chain *is a sequence of resources joined by actions of a single type.*

For example, consider a chain of files created by a sequence of SaveAs actions: F1 was saved as F2 which was saved as F3 which was saved as F4. We can collapse this into a single step F1 $\rightarrow$ F4 for action "SaveAs…", where "SaveAs…" is a resource-chain action. A slightly more general example is a chain of email messages: user A sends message M1 to user B, user B replies with message M2 to user A, and then user A replies with message M3 to user B. We can collapse this into a single step M1 $\rightarrow$ … M3 for the resource chain action "EmailSend…". More formally, for the actions "SaveAs", "EmailSend", and "EmailReceive", we define additional resource chain actions "SaveAs…", "EmailSend…", and "EmailReceive…". Two resources linked by one of these special actions denote a resource-action chain linked by the appropriate primitive actions.

DEFINITION 5. *A* work procedure $\mathcal{WP} = \langle \mathcal{R}, \mathcal{A} \rangle$ *is a labeled directed graph where $\mathcal{R}$ is the set of resources involved in $\mathcal{WP}$ and $\mathcal{A}$ is the set of actions (including resource-chain actions) performed in $\mathcal{WP}$. Each resource $r \in \mathcal{R}$ corresponds to a node. Each action $a \in \mathcal{A}$ corresponds to a directed edge in the graph, with the target resource as the end node. The label of a node is the resource type of that node (document, web page) and the label of an edge is the type of the action (copy/paste, attach file).*

A work procedure is an abstract graph in which the specific identity of the resources and information flow actions is replaced only by their types. Unlike in the information flow

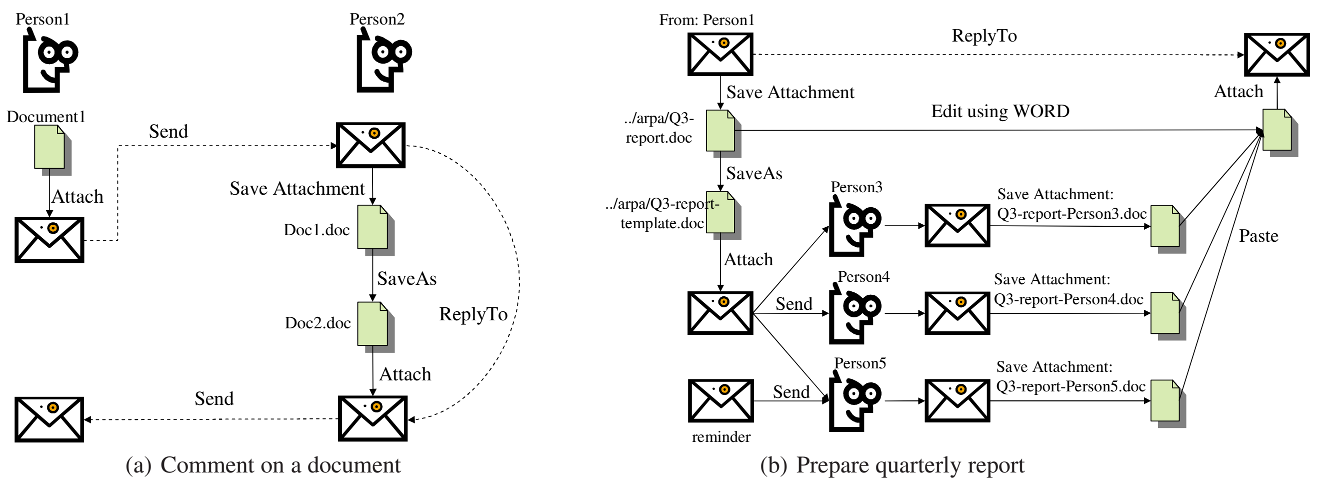


Figure 1. Two typical work procedure examples.

graph, two nodes in a work procedure can be joined by only a single edge.

Note that a work procedure typically does not contain all of the information needed to automate the procedure. For example, editing a document involves more than just performing a SaveAs, and uploading a file typically also involves logging in to a web page, filling out a form, and so on.

DEFINITION 6. *A* work procedure instance *consists of a pair of a work procedure and a subgraph of an information-flow graph joined by a mapping that maps each node and edge in the work procedure into a node or edge of the same type in the information-flow graph. If an edge is labeled with a resource-chain action, then it can be mapped to a chain of 0 or more resources in the information-flow graph.*

Given an information-flow graph and a work procedure, there may be several different work procedure instances in the information-flow graph. These instances may even share nodes and edges.

DEFINITION 7. *The* work procedure discovery problem *is the following: Given an information-flow graph, discover a set of work procedures to cover as much of the graph as possible and that will generalize to additional information-flow graphs.*

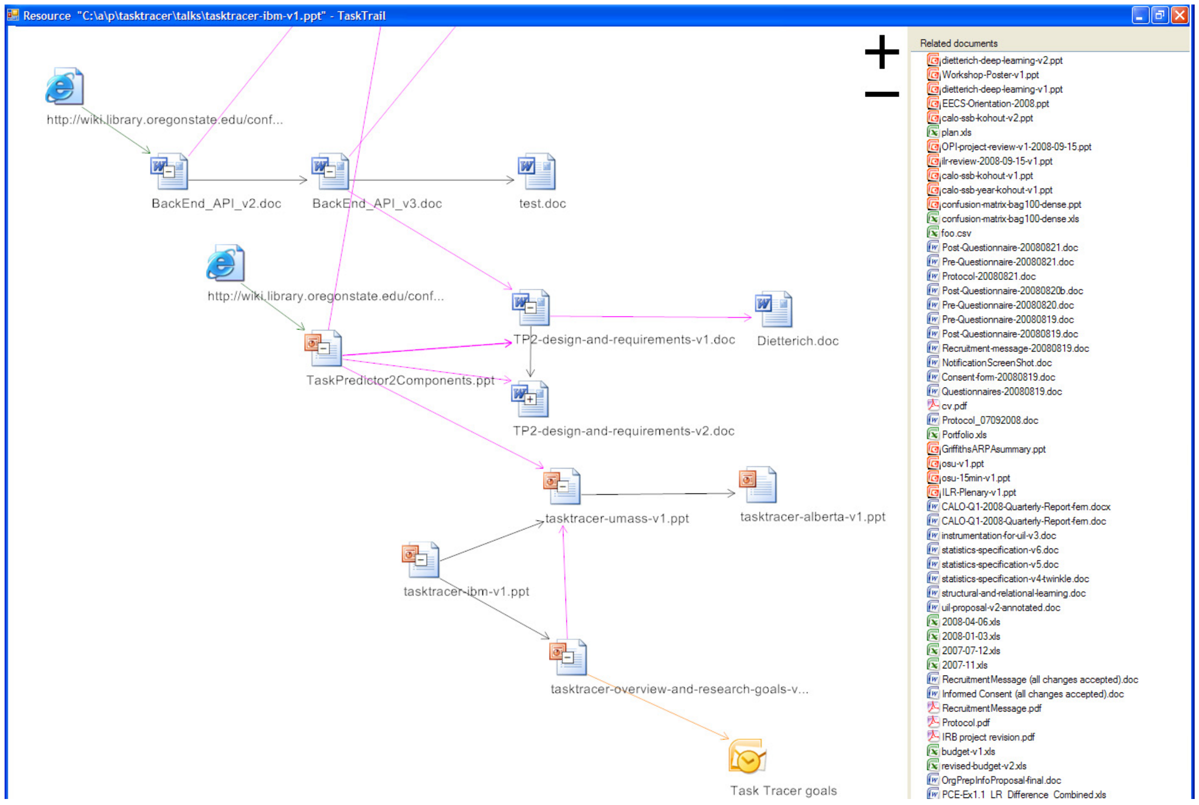
Examples of Work Procedures

A simple work procedure *Provide Comments on a Document* is presented in Figure 1(a). Person2 receives an email message from Person1 requesting comments on an attached Word file. Person2 saves the attachment in a folder associated with the relevant project. He opens the document and uses SaveAs to save it with a new name. After editing the file, Person2 replies to the original email message and attaches the edited file. If the system can recognize instances of this work procedure, it can automatically create a ToDo item when the email arrives and remind Person2 about the progress. After detecting that Person2 has finished editing the document, it can offer to send it back to Person1.

A more complex work procedure *Prepare Quarterly Report* is presented in Figure 1(b). Person2 receives an email from Person1 requesting the quarterly report with an attached Word file and possibly a deadline. Person2 saves the attachment into the relevant folder. He opens the file and does some editing. Let us suppose that Person2 needs to obtain information from three additional people (Person3, Person4, and Person5) to complete the report. He composes an email message to these people requesting their contributions and attaching this file as the template. As replies arrive, he saves their attached files in the same folder. As the deadline approaches, he sends a reminder to those people who have not yet sent in their contributions. When the deadline arrives, he opens the original file and all of the various contributions and he copies/pastes their materials into the file. Finally he replies to the original email from Person1 with the attached report file. If the system could recognize instances of this work procedure, it could automatically create a ToDo item when the email arrives. Later, it could save the attachment and open it in Word. On request, it could create an outgoing email message and attach the template. As the replies come in, it could track them and save the attachments in the right folder. It could also offer to send reminders to the people who have not yet responded. It could also offer to open the template file and all of the contribution files to help the user edit the report. Finally, it could offer to compose a reply email to Person1 and attach the final version of the file.

In this paper, we are interested in work procedures that can provide benefit to the user either through automatic creation and state tracking or through (partial) automation. Hence, we make the following assumptions:

- A work procedure is a (weakly) connected graph. We assume that we can observe (or reconstruct) enough information flow actions so that each work procedure is a weakly connected graph. This means that every node can be reached from every other node if we ignore the directions of the information flow links.
- A work procedure involves at least $k = 3$ resources. Au-


Resource "C:\a\p\tasktracer\talks\tasktracer-ibm-v1.ppt" - TaskTrail
+
−
http://wiki.library.oregonstate.edu/conf...
BackEnd_API_v2.doc
BackEnd_API_v3.doc
test.doc
http://wiki.library.oregonstate.edu/conf...
TP2-design-and-requirements-v1.doc
Dietterich.doc
TaskPredictor2Components.ppt
TP2-design-and-requirements-v2.doc
tasktracer-umass-v1.ppt
tasktracer-alberta-v1.ppt
tasktracer-ibm-v1.ppt
tasktracer-overview-and-research-goals-v...
Task Tracer goals

Related documents

dietterich-deep-learning-v2.ppt
Workshop-Poster-v1.ppt
dietterich-deep-learning-v1.ppt
EECS-Orientation-2008.ppt
calo-ssb-kohout-v2.ppt
plan.xls
OPI-project-review-v1-2008-09-15.ppt
ilr-review-2008-09-15-v1.ppt
calo-ssb-kohout-v1.ppt
calo-ssb-year-kohout-v1.ppt
confusion-matrix-bag100-dense.ppt
confusion-matrix-bag100-dense.xls
foo.csv
Post-Questionnaire-20080821.doc
Pre-Questionnaire-20080821.doc
Protocol-20080821.doc
Post-Questionnaire-20080820b.doc
Pre-Questionnaire-20080820.doc
Pre-Questionnaire-20080819.doc
Post-Questionnaire-20080819.doc
Recruitment-message-20080819.doc
NotificationScreenShot.doc
Consent-form-20080819.doc
Questionnaires-20080819.doc
cv.pdf
Protocol_07092008.doc
Portfolio.xls
GriffithsARPAsummary.ppt
osu-v1.ppt
osu-15min-v1.ppt
ILR-Plenary-v1.ppt
CALO-Q1-2008-Quarterly-Report-fem.docx
CALO-Q1-2008-Quarterly-Report-fem.doc
instrumentation-for-uil-v3.doc
statistics-specification-v6.doc
statistics-specification-v5.doc
statistics-specification-v4-twinkle.doc
structural-and-relational-learning.doc
uil-proposal-v2-annotated.doc
2008-04-06.xls
2008-01-03.xls
2007-07-12.xls
2007-11.xls
RecruitmentMessage (all changes accepted).doc
Informed Consent (all changes accepted).doc
RecruitmentMessage.pdf
Protocol.pdf
IRB project revision.pdf
budget-v1.xls
revised-budget-v2.xls
OrgPrepInfoProposal-final.doc
PCE-Ex1.1 LR Difference Combined.xls

Figure 2. Screenshot of TaskTrail displaying a real-case provenance connection.

tomatic tracking of work procedures is likely to be more useful if there are many resources involved, because users have more difficulty keeping track of the state of more complex procedures.

- A work procedure involves resources other than emails. A procedure only consisting of emails is not our focus. First, there are already very good email threading tools that can help users track their work status [19]. We believe it is also more difficult for the user to track work procedures involving different kinds of resources, because this requires working with multiple application programs.

BUILDING THE INFORMATION FLOW GRAPH

We build the information flow graph by utilizing the provenance information collected by TaskTracer and augmenting it with links inferred by analyzing the contents of resources.

The TaskTracer System and Provenance Information

User's activities applied to resources are captured by the TaskTracer system [7]. TaskTracer is an intelligent activity management system that allows users to organize and retrieve information based on their activities. TaskTracer collects various time-stamped user interactions (such as file new, open, save, text selection, copy/paste, windows focus, web navigation, email read/send, etc.) in Microsoft Office (Word, Excel, PowerPoint, Outlook), text and pdf files, Internet Explorer, and the Windows operating system. Each interaction generates events which are stored in a database. TaskTracer associates with each activity the set of resources accessed when performing that activity. It leverages this data and machine learning technology [31] to configure the desktop to assist users in organizing and re-finding information.

Recent research [4] has shown that common search criteria, such as file modification time and even the document title, are usually remembered inaccurately. There are other resource attributes that are more easily remembered by users and could be helpful in re-finding documents. One such attribute is a resource's relationship to other resources [13, 4]. Inspired by this and other work, we extended TaskTracer to capture a variety of provenance actions. These are also published as events within TaskTracer and recorded in the event database and in a provenance database. We have found that even without any further analysis, provenance data can provide answers to user queries that cannot be obtained in any other way. For example, a user might want to know such things as "Did I ever email this Word file to Bob?", "Did I save this email attachment, and if so, where did I put it?", or "What Powerpoint presentations did I copy slides from to produce this presentation?" We have modified the Windows and Outlook user interfaces so that the user can right-click on a file or email message and request a "provenance graph" to be displayed. *TaskTrail* is the TaskTracer component that displays these graphs and allows users to interact with them.

A real case using TaskTrail is shown in Figure 2. The user could not find the PowerPoint document he prepared for Alberta. But he remembered that this document was created from a previous presentation "tasktracer-ibm-v1.ppt" and he knew where that presentation was located. So he right-clicked "tasktracer-ibm-v1.ppt" and chose "Show TaskTrail" from the context menu. He instantly found what he was searching in the provenance graph. The left panel of TaskTrail visualizes the entry point resource and resources related via provenance (moving from left to right). The right panel lists all resources contained in the current provenance relationship. The user can change the display resolution, hover the mouse over nodes and links to get detailed information, and double-click on nodes to open resources.

Inferring Implicit Links

Some important provenance relations are not currently captured by TaskTracer because the corresponding instrumentation has not been implemented. Hence, for the data analyzed in this study, we analyzed the contents of various resources to reconstruct additional information flow relationships:

- **Email threading**: if email message X is a reply or forward of message Y, we want to create a "reply/forward" link from Y to X. An email thread is a group of messages that are related by such links. By capturing email threads, we can connect many resources which seemed unrelated before. Those threading relationships can be accurately recovered by analyzing the subject lines and the RFC-822 header fields of email messages (such as *Message-ID*, *In-Reply-To*, *References* field, etc).
- **Document conversion**: many users prefer to convert doc files into pdf files in order to send them as email attachments. We would like to create a "PDFPrint" link from this doc file to this pdf file. To capture such connection, we extract the text contents of each pdf file and compare it to the contents of doc files. If it matches a doc file, we create a link from this doc file to the pdf file.
- **URL click-through**: instead of attaching a file, the user sometimes provides a URL in the email message and asks the recipient to download the file. If the user clicks on a URL in an email and opens a website or downloads a file, we wish to create a "Click-through/SaveAttachment" link from this email to the website or the file. For each navigation or download event, we check if the previous window focus is the email client. If it is and the opened email message contains that URL, we create a "Click-through/SaveAttachment" link.
- **Email reference**: some email messages contain information about specific documents. For example, the user might receive an email regarding his paper submission and asking for a response. There should be a "Reference" link from this document to the email message. Such email usually contains the paper title in the email body. We use a pre-trained key value extractor [6] and "false positive" regular expressions [19] to extract key words such as document titles in email messages. We define the first 200 words of a document to be the abstract of that document. If any key word extracted from the email message appears in the abstract, there is a potential connection. If an email matches multiple documents (this could happen when the user edits and saves the document with several versions), we only create one link from the document which has the closest time distance to the email.

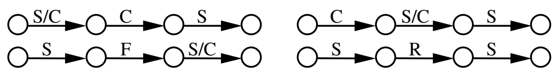


Figure 3. Four occurrences of a frequent pattern. S: SaveAs, C: Copy&Paste, F: File Copy, R: File Rename.

MINING FREQUENT WORK PROCEDURES

Given information flow graphs of desktop users, we can infer work procedures by identifying recurring patterns within these graphs. For computational reasons, we do this in two steps. First, we ignore the labels on the links in the graph and search for frequently-occurring subgraphs. Then, we choose action types for each edge in the discovered subgraphs by analyzing all instances of those subgraphs to choose the most frequently occurring combinations of action types.

Mining Closed Frequent Relations

Our goal is to find closed weakly connected patterns in the information flow graph that are frequent (i.e., have at least s instances). A directed graph G is weakly connected if replacing all of its directed edges with undirected edges will produce a connected (undirected) graph. The frequency of a pattern graph P is the number of instances of P in the information flow graph G. A pattern graph P is frequent if its frequency is greater than or equal to a specified minimum support threshold s. A frequent pattern P is closed if there is no graph P' such that P is a subgraph of P' and they have the same frequency. In this sense, closed patterns are locally "maximal" patterns that cannot grow any further without losing some instances.

Graph pattern mining [16, 33, 29] follows the Apriori principle: the support of a subgraph is always no less than the support of any of its super-graphs. Thus we can first consider frequent small graphs, and then check whether larger graphs that contain them are still frequent. There are two general approaches to efficient graph pattern mining [33]: Apriori-based approaches extend the Apriori-based candidate generation-and-test approach while pattern-growth approaches grow patterns from a single graph directly. Pattern-growth approaches are usually more efficient, since they avoid some costly operations such as joining two subgraphs into a larger graph. In this paper, we extend the GASTON algorithm [29]. Recall that we ignore patterns that consist only of email messages and patterns that involve fewer than k resources. The remaining closed patterns correspond to candidate work procedures, except that their arcs are not labeled.

Searching For Frequent Activity Paths

The next step is to assign action type labels to the arcs. In principle, we could consider all possible action type assignments and score them to see how many work procedure instances they have. We would then retain only those assignments that had at least the minimum number of instances. However, there are combinatorially many possible assignments. Simply picking the most frequent individual action type for each edge does not necessarily find patterns with many instances. Figure 3 shows an example. If we pick the most frequent activity separately for each edge, then the labels S, C, and S would be the result. However, pattern (S, C, S) only appears once. Instead, the most frequent assignment should be (C, C, S), which appears twice.

Algorithm 1 Search For Frequent Action Label Assignments

```
Require: $P$: frequent pattern, $O$: its occurrences in $D$
 1: $A = \{\langle \emptyset, O \rangle\}$
 2: for each edge $e = u \rightarrow v$ in $P$ do
 3:     $A' = \emptyset$
 4:     for each label $a$ appearing between $u$ and $v$ in $O$ do
 5:         for each set $\langle Acts, Occ \rangle \in A$ do
 6:             Search for occurrence set
                    $O' = \{o | Acts \wedge (a \in e) \in o, o \in Occ\}$
 7:             if $|O'| \geq$ the support threshold then
 8:                 $A' = A' \cup \{\langle Acts \wedge (e = a), O' \rangle\}$
 9:             end if
10:         end for
11:     end for
12:     if $A' = \emptyset$ then
13:         return fail
14:     else
15:         $A = A'$
16:     end if
17: end for
18: return all activity paths in $A$
```

We employ a dynamic programming approach, as described in Algorithm 1, to search for action assignments that appear no less than the minimum support threshold in the information flow graph. The complexity of this algorithm is $O(|P| \cdot m \cdot s)$ where $|P|$ is the number of edges in pattern P, m is the maximal number of labels that edges of P have, and s is the largest size of P's frequent action subsets given different numbers of actions. Since m and s are usually small, this algorithm can efficiently find the frequent action label assignments. Note that there is a chance that no assignment of action types to the arcs in P can produce a sufficiently frequent work procedure, in which case we discard P.

EXPERIMENTAL RESULTS

To evaluate our approach, we conducted experiments based on data from real users. In December, 2007, the TaskTracer system was deployed on Windows machines at SRI International as part of a 3-day data collection exercise. Four staff members participated in an exercise of knowledge work in which they submitted and reviewed papers for a conference, filed travel documents, and prepared quarterly reports. These activities were interleaved with other things (e.g., reading online newspapers). We combined the actions identified through desktop instrumentation with the additional actions described above to produce information flow graphs for each user. We manually analyzed the data and identified all of the work procedures that the users executed. We identified 24 instances of work procedures involving 118 resources and 106 actions.

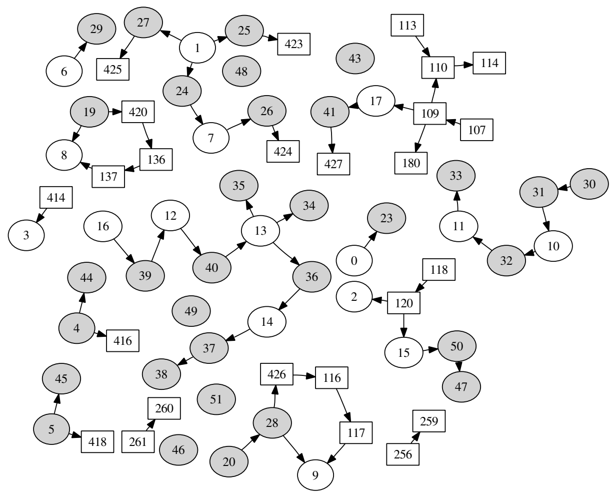


Figure 4. Part of a user's resource relationship graph. Shadowed circles are received emails. Non-shadowed circles are sent-out emails. Boxes are non-email resources.

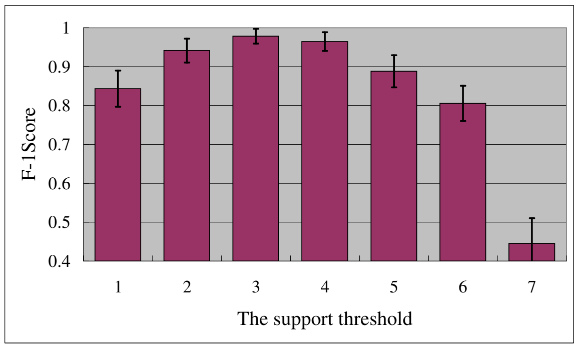


Figure 5. F1 score as a function of the minimum support threshold, with 95% confidence intervals.

We applied our learning algorithm to these information flow graphs to discover work procedures. We evaluate the results by measuring the extent to which the discovered work procedures match the manually-identified instances without including false steps or omitting steps. Performance is measured by *precision* and *recall*. Precision is defined as the number of relevant resources and actions in the set of discovered occurrences divided by the total number of resources and actions in the set of occurrences, and Recall is defined as the number of relevant resources and actions in the set of discovered occurrences divided by the total number of existing relevant resources and actions (which should have been retrieved into the set of occurrences). We evaluate the algorithms with the F1 score [17] which is 2*precision*recall /(precision+recall).

A portion of one user's resource relationship graph is shown in Figure 4. It contains some interesting work procedures. For example, Email28 from User2 is the start of a work procedure asking User1 to comment on User2's paper and Email19 from User3 is the start of a work procedure asking User1 to make contribution to a quarterly report.

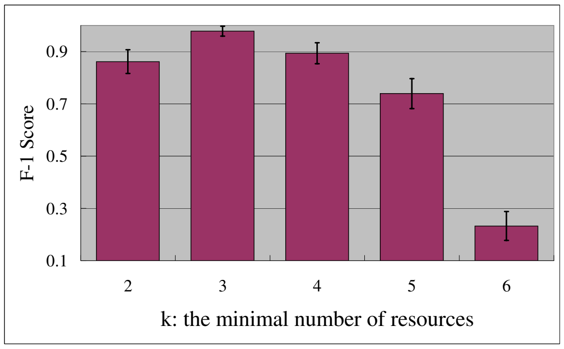


Figure 6. F1 score as a function of the minimum number of resources involved in work procedures, with 95% confidence intervals.

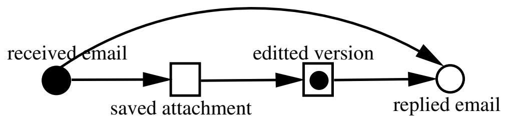


Figure 7. The discovered most frequent work procedure: edit and return Document. The box with a dot inside means zero to multiple "SaveAs" resources.

Figure 5 plots F1 as a function of the minimum support threshold of the discovery algorithm. When the threshold is small, the algorithm falsely treats many irrelevant resources and actions as work procedures. This produces high recall but very low precision. As the threshold increases, the number of false work procedures decreases, since such false procedures don't happen very often in the data. The best trade-off between precision and recall is achieved when the threshold is 3. Our approach successfully discovers most work procedures without including many false resources or actions.

Figure 6 plots F1 as a function of the minimum number k of resources that must be present in the work procedure. When k is small, the algorithm falsely treats some random patterns as work procedures. This produces low precision. Most work procedures involve fewer than 6 resources. Thus the recall is very low when k becomes too large. The best performance is achieved when we require that a work procedure involves at least 3 resources.

The most frequent work procedure we discovered was "edit and return a document", as shown in Figure 7. In this procedure, the user receives an email with an attachment, asking him to edit this attachment and return it. So he saves that attachment and begins editing it. He sometimes uses SaveAs to save the edited document with a new name. After editing the file, he replies to the original email message and attaches the edited file. This general work procedure includes at least two cases: "comment on another person's paper" and "contribute to CALO quarterly report".

CONCLUSION AND FUTURE WORK

This paper investigates how to discover work procedures from user desktop activity. A work procedure is defined as a directed labeled graph, with nodes corresponding to resources involved and edges corresponding to actions executed. To discover work procedures frequently executed, we first build an information flow graph of resources based on the provenance information captured by TaskTracer (and manually extended by analyzing resource content). We then applied a two-phase algorithm to find frequently-occurring work procedures in these graphs. Experimental results show that our approach can accurately discover almost all of the known work procedure instances without introducing many false work procedures.

The focus of this paper is to discover work procedures frequently executed by users. The procedures that we discover do not contain non-deterministic choices or conditionals (except for the resource-action chains). The procedures also do not capture partial order constraints on the steps. One can imagine learning models that include "choice" branches that require the user to choose one of several paths to reach a goal (which would support optional steps and alternative ways of performing a task) and "parallel" branches that require the user to finish a set of parallel paths in order to reach a goal [23] (which would support partially-ordered actions). With these extensions, different executions of one procedure could lead to different observations or to observations occurring in different orders. With enough data, it should be possible to discover these more complex procedures.

Another goal of future work will be to convert the discovered work procedures into a form that supports efficient recognition and state estimation. One candidate representation is Logical Hidden Markov Models (logical HMMs) [28]. Logical HMMs can handle passing parameters (e.g., email addresses, resource ids) from one step to another in the work procedure. They have been shown to be effective in modeling users' actions [28]. With a well-defined logical HMM for each discovered work procedure, an intelligent assistant could monitor the user's behavior and determine the state of execution of each procedure instance. This would support a smart ToDo manager that could automatically populate the ToDo list and automatically detect when items were complete and could be removed from the ToDo list.

Acknowledgments

This work was supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. FA8750-07-D-0185/0004. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA, or the Air Force Research Lab. The authors thank the TaskTracer team for implementing the instrumentation, Xiaoli Fern, Akihiro Inokuchi and Xifeng Yan for kindly answering research questions, and Hung H. Bui, Julie Wong and other participants at SRI for providing desktop data.

REFERENCES

1. R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In *Proc. of Sixth International Conference on Extending Database Technology*, pp 469–483, 1998.

2. S. Amershi and C. Conati. Unsupervised and supervised machine learning in user modeling for intelligent learning environments. In *IUI-07*, pp 72–81.
3. V. Bellotti, N. Ducheneaut, M. Howard, and I. Smith. Taking email to task: the design and evaluation of a task management centered email tool. In *CHI-03*, pp 345 – 352, 2003.
4. T. Blanc-Brude and D. L. Scapin. What do people recall about their documents?: implications for desktop search tools. In *Proc. of IUI-07*, pp 102–111, 2007.
5. B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. In *KDD-03*, pp 493–498, 2003.
6. W. W. Cohen. *Minorthird: Methods for Identifying Names and Ontological Relations in Text using Heuristics for Inducing Regularities from Data*, 2004. Software available at http://minorthird.sourceforge.net.
7. A. N. Dragunov, T. G. Dietterich, K. Johnsrude, M. McLaughlin, L. Li, and J. L. Herlocker. Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *IUI-05*, pp 75–82, 2005.
8. M. Dredze, T. Lau, and N. Kushmerick. Automatically classifying emails into activities. In *Proc. of IUI-06*, pp 70 – 77, 2006.
9. M. El-Ramly, E. Stroulia, and P. Sorenson. From run-time behavior to usage scenarios: an interaction-pattern mining approach. In *Proc. of KDD-02*, pp 315–324, 2002.
10. C. A. Ellis, K.-H. Kim, and A. J. Rembert. Workflow mining: Definitions, techniques, and future directions. *Workflow Handbook*, pp 213–228, 2006.
11. X. Z. Fern, C. Komireddy, and M. Burnett. Mining interpretable human strategies: A case study. In *Proc. of ICDM-07*, pp 475–480, 2007.
12. J. Frew, D. Metzger, and P. Slaughter. Automatic capture and reconstruction of computational provenance. *Concurr. Comput. : Pract. Exper.*, 20(5):485–496, 2008.
13. D. Gonçalves and J. A. Jorge. Describing documents: what can users tell us? In *IUI-04*, pp 247–249, 2004.
14. G. Greco, A. Guzzo, G. Manco, and D. Sacca. Mining and reasoning on workflows. *IEEE Trans. on Knowl. and Data Eng.*, 17(4):519–534, 2005.
15. R. Hamid, S. Maddi, A. Johnson, A. Bobick, I. Essa, and C. Isbell. Unsupervised activity discovery and characterization from event-streams. In *UAI-05*, 2005.
16. A. Inokuchi, T. Washio, K. Nishimura, and H. Motoda. A fast algorithm for mining frequent connected graphs. Technical Report RT0448, IBM Research, 2002.
17. T. Joachims. *Learning to Classify Text Using Support Vector Machines*. Kluwer Academic Publishers, 2001.
18. N. Kushmerick and T. Lau. Automated email activity management: an unsupervised learning approach. In *Proc. of IUI-05*, pp 67 – 74, 2005.
19. D. Lam, S. L. Rohall, C. Schmandt, and M. K. Stern. Exploiting e-mail structure to improve summarization. In *Proc. of CSCW-02*, 2002.
20. G. Leshed, E. Haber, T. Matthews, and T. Lau. Coscripter: Automating and sharing how-to knowledge in the enterprise. In *CHI 2008*, 2008.
21. J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. In *the 2nd workshop on temporal data mining at the 8th SIGKDD*, pp 53–68, 2002.
22. G. Mark, V. M. Gonzalez, and J. Harris. No task left behind? examining the nature of fragmented work. In *Proc. of CHI-05*, pp 321 – 330, 2005.
23. A. K. A. D. Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Workflow mining: Current status and future directions. In *CoopIS/DOA/ODBASE, LNCS-2888*, pp 389–406, 2003.
24. D. Minnen, C. Isbell, I. Essa, and T. Starner. Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. In *Proc. of AAAI-07*, pp 615–620, 2007.
25. D. Minnen, T. Starner, I. Essa, and C. Isbell. Improving activity discovery with automatic neighborhood estimation. In *Proc. of IJCAI-07*, pp 6–12, 2007.
26. T. M. Mitchell, S. H. Wang, Y. Huang, and A. Cheyer. Extracting knowledge about users' activities from raw workstation contents. In *Proc.of AAAI-06*, 2006.
27. K.-K. Muniswamy-Reddy, D. A. Holland, U. Braun, and M. Seltzer. Provenance-aware storage systems. In *Proc. of the annual conference on USENIX-06*, 2006.
28. S. Natarajan, H. H. Bui, P. Tadepalli, K. Kersting, and W.-K. Wong. Logical hierarchical hidden markov models for modeling user activities. In *Proc. of ILP-08*, pp 192–209, 2008.
29. S. Nijssen and J. N. Kok. A quickstart in frequent structure mining can make a difference. In *Proc. of KDD-04*, pp 647–652, 2004.
30. S. Shah, C. A. N. Soules, G. R. Ganger, and B. D. Noble. Using provenance to aid in personal file search. In *Proc. of the USENIX-07*, 2007.
31. J. Shen, L. Li, and T. G. Dietterich. Real-time detection of task switches of desktop users. In *Proc. of IJCAI-07*, pp 2868–2873, 2007.
32. Y. Tanaka, K. Iwamoto, and K. Uehara. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Machine Learning*, 58(2-3):269–300, 2005.
33. X. Yan and J. Han. Closegraph: mining closed frequent graph patterns. In *KDD-03*, pp 286–295, 2003.