IEICE TRANS. INF. & SYST., VOL.E97–D, NO.4 APRIL 2014

PAPER Special Section on Data Engineering and Information Management

File and Task Abstraction in Task Workflow Patterns for File Recommendation Using File-Access Log**

Qiang SONG[†*a)], **Takayuki KAWABATA**[††b)], *Nonmembers*, **Fumiaki ITOH**[††c)], **Yousuke WATANABE**[†d)], *and* **Haruo YOKOTA**[†e)], *Members*

SUMMARY The numbers of files in file systems have increased dramatically in recent years. Office workers spend much time and effort searching for the documents required for their jobs. To reduce these costs, we propose a new method for recommending files and operations on them. Existing technologies for recommendation, such as collaborative filtering, suffer from two problems. First, they can only work with documents that have been accessed in the past, so that they cannot recommend when only newly generated documents are inputted. Second, they cannot easily handle sequences involving similar or differently ordered elements because of the strict matching used in the access sequences. To solve these problems, such minor variations should be ignored. In our proposed method, we introduce the concepts of abstract files as groups of similar files used for a similar purpose, abstract tasks as groups of similar tasks, and frequent abstract workflows grouped from similar workflows, which are sequences of abstract tasks. In experiments using real file-access logs, we confirmed that our proposed method could extract workflow patterns with longer sequences and higher support-count values, which are more suitable as recommendations. In addition, the F-measure for the recommendation results was improved significantly, from 0.301 to 0.598, compared with a method that did not use the concepts of abstract tasks and abstract workflows.

key words: *file recommendation, file abstraction, abstract task, abstract workflow, log analysis*

1. Introduction

In recent years, the numbers of files in file systems used for business have grown dramatically in step with the recent information explosion. As a consequence of this increase, office workers now spend much time and effort searching for files that include documents and data required for their business. For example, a white paper [1] reports that about 25% of working time in offices is occupied by "search" tasks. It is therefore important to find methods for reducing the time wasted in ineffective and unproductive searching. Also, in office, the searching targets are not limited to documents, the abstract working processes which we call them abstract workflows in this paper are also being searched by users.

Collaborative filtering [2] is well known as an algorithm for making recommendations. It predicts future actions or items based on information about the past actions or items of other users with similar behavior. The collaborative-filtering approach can also be applied to file recommendation. However, there are problems with the original form of collaborative filtering. It only works with files accessed in the past. In other words, the recommendation systems based on collaborative filtering do not work well on newly generated files. Moreover, collaborative filtering does not handle well sequences that include elements that are similar but not identical, or that have a different ordering of elements, because the algorithm uses strict matching of elements in access sequences.

To solve these problems, we have proposed a method for recommending files and types of operations on them, such as open file and copy file, that supports creative business processes as an operational tool [3]. In this paper, we add a more detailed description of our proposed method, more experiments and a more specific discussion on experimental results.

In our method, we consider the abstract workflows for a user. These are different from the ordinary workflows described by users, being generated by mining the file-access histories of the users. To recommend files and operations on them to a user, our method reserves the extracted workflows in a database, and retrieves patterns that match the current partial workflow of the user. To obtain good recommendations, it is important to ignore small variations in the matching process. In this paper, we introduce for the first time the concepts of abstract files, abstract tasks, and frequent abstract workflows. Abstract files are groups of similar files used for similar purposes, abstract tasks are groups of similar tasks, and frequent abstract workflows are groups of similar workflows, in which sequences of abstract tasks appear frequently.

The differences between collaborative filtering and the method that we propose in this paper are that we add the concept of tasks to weaken the constraint of strict matching for access sequences and we apply abstraction to files, tasks, and workflows to handle newly generated files. Our method first extracts the abstract tasks that are collections of files and operations on them, and then extracts working patterns as abstract workflows between the abstract tasks. After extracting this information from logs, our method then

Manuscript received July 4, 2013.
Manuscript revised October 29, 2013.

[†]The authors are with Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
[††]The authors are with the Canon Inc., Tokyo, 146–8501 Japan.
[*]Presently, with NTT DATA Corporation.

[**]A major part of this paper was presented at 24st International Conference on Database and Expert Systems Applications, Prague, Czech Republic.

a) E-mail: soukyou@de.cs.titech.ac.jp
b) E-mail: kawabata.takayuki@canon.co.jp
c) E-mail: ito.fumiaki@canon.co.jp
d) E-mail: watanabe@de.cs.titech.ac.jp
e) E-mail: yokota@cs.titech.ac.jp

DOI: 10.1587/transinf.E97.D.634

identifies the abstract workflow that best matches the current file-usage pattern of the user. This abstraction increases the number of patterns that match flexible user behavior and any newly generated files. Finally, our method recommends a combination of files and operation types.

In this paper, we describe an evaluation of the proposed method using real file-access logs. The evaluation results indicate that our method extracts workflow patterns with longer sequences and higher support-count values, which are more suitable as recommendations. Here support-count means the occurrence times of workflow patterns. Moreover, the results demonstrate that using the concepts of abstract tasks and abstract workflows improves the quality of the recommendations.

The remainder of this paper is organized as follows. In Sect. 2, we describe our goal and approach. We then outline our proposed method in Sect. 3. Details of the process of our proposed method are described in Sects. 4 and 5. Section 6 reports on the experimental evaluation. Related work is presented in Sect. 7. Section 8 offers conclusions and proposals for future work.

2. Goal and Approach

Figure 1 illustrates an example of the type of workflow pattern we are investigating. We assume that user *C* creates a new file "Ordering Instruction for Company Z." The goal is to predict the files required, for recommendation to user *C*.

In our approach, we search for similar patterns of operation in logs and make recommendations based on them. For example, the logs contain a history of user *A* accessing the files "Ordering Instruction for Company X" → "Example of an Estimate" → "Information of Product X." Similarly, user *B* accessed files "Ordering Instruction for Company Y" → "Information of Product Y" → "Example of an Estimate." In these two patterns, different files were accessed in different orders, but it is considered that users *A* and *B* were doing essentially the same work. Therefore, we expect to extract an abstract pattern of work, namely "ordering instruction" → "example estimate file" or "product information file." Using this abstract workflow pattern, it becomes possible to recommend files such as "Example of an estimate" or "product information file" to user *C*.

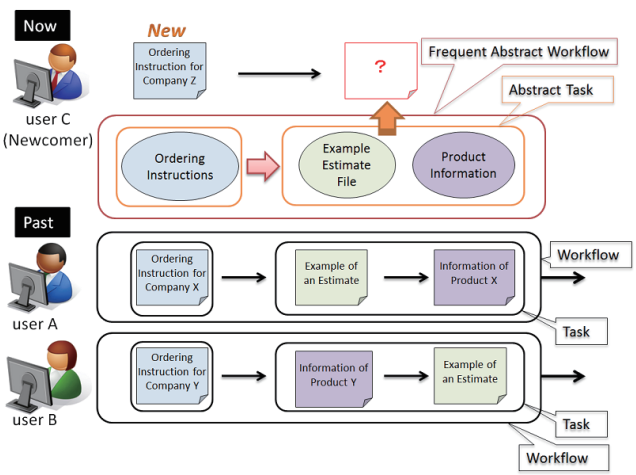

Fig. 1 Assumed workflow pattern.

3. Outline of Proposed Method

3.1 Overall View of the System

Figure 2 shows an overview of our method. The process flow comprises two parts, namely the offline and online parts, with the online part comprising two modules, namely the monitoring and recommendation modules. While users access files located on a file server, file-access histories are stored in a log file. Our method extracts abstract tasks and workflows from the log file, and reserves them in a database. The method then monitors the current file accesses by a user, and searches for workflows in the database that match the access patterns of that user to infer the user's current workflow pattern. Based on the inferred workflow, the method recommends those files, together with operations on them.

The processing procedures are described in detail in Sects. 4 and 5.

3.2 Workflow Model

In general, the orders of sequences of individual operations will vary, even if the outlines of the workflows are the same. To ignore such subtle differences in operational order in similar workflows, we introduce the concept of an abstract task as a group of similar combinations of file and operation, and an abstract workflow as a sequence of abstract tasks.

(1) **[Task]:**

To extract tasks, we first determine a parameter, Time Gap Between Tasks (TGBT). If there are no file operations by a user during an interval longer than TGBT, we infer that the previous task had finished before the interval and the next task has started after it. Our method scans the separate log of each user to find blank intervals longer than TGBT

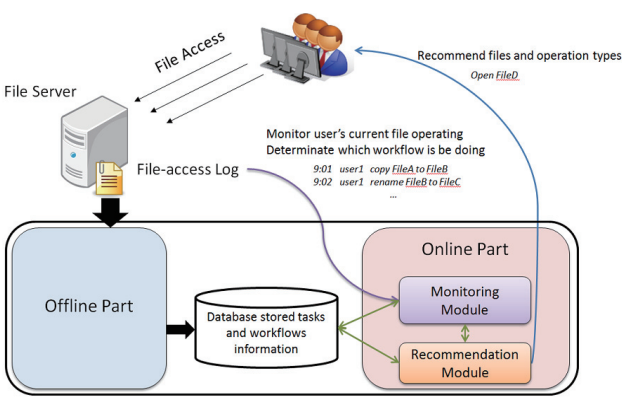

Fig. 2 Overall view of the system.

between log records. Each subsequence between gaps in the log is then identified as a *task*. A task is the basic unit in a working process's workflow, and the ordering of file operations within it is ignored. For example, from the log below, we set TGBT = 3 minutes.

- Record 1: 12:00 [Open File A]
- Record 2: 12:01 [Open File B]
- Record 3: 12:10 [Open File C]
- Record 4: 13:00 [Open File D]

Three tasks will be extracted from these four records.

- Task 1: {[Open File A], [Open File B]}
- Task 2: {[Open File C]}
- Task 3: {[Open File D]}

(2) **[Abstract Task]:**

Next, in order to ignore small variations in between tasks, we consider abstract tasks. An *abstract task* is a set of combinations of file and operation derived by clustering similar tasks. First, we calculate the degree of similarity between files. After calculating the similarity, files with high similarity are grouped in a cluster. In the above example, we assume that Cluster 1 contains File A and File C while Cluster 2 contains File B, Cluster 3 contains File D. We then cluster similar tasks into groups as abstract tasks. Note that abstract tasks are formed from file-clusters, not the individual files. In terms of the abstract task, files and operational orders with small differences are treated as the same work. In the above example, three abstract tasks would be derived as follows.

- Abstract Task 1: [Open File-Cluster 1], [Open File-Cluster 2]
- Abstract Task 2: [Open File-Cluster 1]
- Abstract Task 3: [Open File-Cluster 3]

(3) **[Abstract Workflow]:**

Next, we define abstract workflows. An *abstract workflow* is a sequence of abstract tasks. Similarly to the process extracting tasks, we first determine a parameter, Time Gap Between Workflows (TGBW). If there are no file operations during an interval longer than TGBW, we infer that the previous workflow had finished before the interval and the next workflow has started after it. In typical working patterns, TGBW is much longer than TGBT. For the above example, we set TGBW = 30 minutes and can extract two abstract workflows that are sequences of the abstract tasks.

- Abstract Workflow 1: [Abstract Task 1] → [Abstract Task 2]
- Abstract Workflow 2: [Abstract Task 3]

(4) **[Frequent Abstract Workflow]:**

Because large numbers of abstract workflows can be derived from the file-access logs for the full range of office work, it is useful to reduce the number of target abstract workflows by extracting patterns that occur frequently. We call these frequent patterns of abstract workflow *frequent abstract workflows*. In this paper, we adopt sequential-pattern-mining technique BIDE+ [4] to derive the frequent abstract workflows.

4. Offline Part

The aim of this part is to extract abstract tasks and frequent abstract workflows from file-access logs. As shown in Fig. 3, there are mainly 3 steps in offline part. Details of each step of the offline part are described in the following subsections.

4.1 Extraction of Tasks and Workflows

We first partition the log file in terms of user ID. Then as mentioned in Sect. 3.2, we cut out tasks by parameter TGBT and workflows by parameter TGBW. In Fig. 3, We partition the log into six tasks and three workflows.

4.2 Abstraction of Tasks and Workflows

4.2.1 Calculation of Similarity between Files

To treat files with small differences as the same work, we cluster similar files together. Here, cause we want to group files used for the same purposes together, how to define the similarity is important. Simply using the contents of the files does not fit for our purpose. For example, there are two ordering instruction files for different companies. The contents vary a lot, because these two files are for different companies. It is hard to group them together by contents information. But the files are both ordering instruction files, in our method we want to cluster them together. To do this, the copy relationship and the similarity in filenames are useful.

(1) [Copy-relation Similarity]:

It is reasonable to assume that files that have been copied from the same template file will have a strong commonality

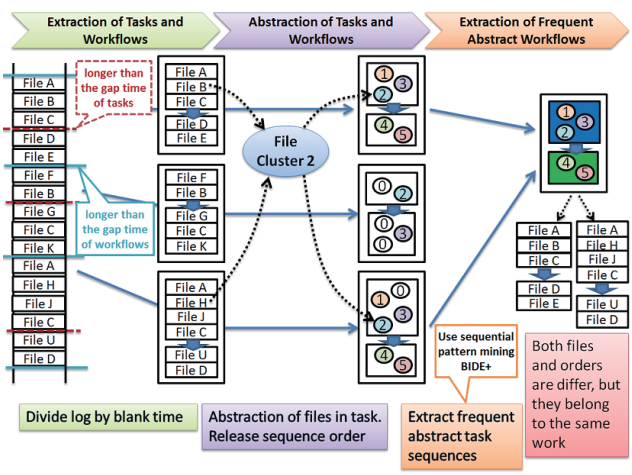

Fig. 3 Offline part.

of purpose. The equation for calculating the copy-relation similarity is

$$simCopy(file_a, file_b) = \begin{cases} \gamma^{steps} & (if\ copy\ relationship\ from\ file_a\ to\ file_b\ exists) \\ 0 & (otherwise). \end{cases} \quad (1)$$

We assume that $file_a$ and $file_b$ have a copy relationship between them. Here, $steps$ is the number of transition steps from $file_a$ to $file_b$. γ is the attenuation coefficient. In our experiment, we set γ as 0.9. For example, files "A," "B," and "C" have the copy relation A → B → C. Files "A" and "B" have a parent–child relationship. The similarity between A and B is 0.9. Files "A" and "C" have a grandparent–grandchild relationship. The similarity between "A" and "C" is attenuated to 0.81.

(2) [Filename Similarity]:

Another similarity between files is obtained from their filenames. It is reasonable to assume that if files have similar filenames and extensions, the degree of similarity between them will be high. For example, it can be estimated that files such as "$ordering\_instruction\_for\_company\_X.docx$" and "$ordering\_instruction\_for\_company\_Y.docx$" have a similar commonality of purpose.

We can treat filenames as sequences or sets, as the smallest units are characters or words. We extract words from filename by morphological analysis. If we treat filenames as sequences, as the metrics for measuring the difference between two sequences, we can by finding the Longest Common Subsequence (LCS) [5] or calculating the Levenshtein Distance [6]. If we treat filenames as sets, we can compare two filenames by finding same 2-grams on character level or use Dice's Coefficient [7]. By comparing these methods, we found that dividing filename into words ahead, then compare filename sequences using the LCS is the most stable way. The formula we use for calculating the file similarity using filename is

$$simFilename(file_a, file_b) = \frac{len(LCS(file_a, file_b))}{min(len(file_a), len(file_b))} * \delta. \quad (2)$$

Here, $len(file_X)$ means the number of words in filename of $file_X$, $min(len(file_a), len(file_b))$ represents the length of the shorter of $file_a$ and $file_b$, and $LCS(file_a, file_b)$ represents the LCS in the filenames for $file_a$ and $file_b$. If the extensions of $file_a$ and $file_b$ are different, we lower their similarity by a multiplying parameter δ. In our experiments, we set $\delta = 0.9$.

Finally, the overall degree of similarity between $file_a$ and $file_b$ is given by a linear sum of these two kinds of similarities, namely

$$sim(file_a, file_b) = \alpha * simCopy(file_a, file_b) + \beta * simFilename(file_a, file_b). \quad (3)$$

α and β are weight parameters, From the experiments described in Sect. 6, we set $\alpha = 2$ and $\beta = 1$.

4.2.2 Abstraction of Files

Using the similarity between files, we apply agglomerative hierarchical clustering [8] to the files. Abstraction of files means replacing files, which appeared in the access log, with the corresponding clusters.

4.2.3 Calculation of Similarity between Tasks

As described in Sect. 3.2 (1), by comparing each blank time between two continuous file operations with parameter TGBT, we separate the log into tasks. So each task represents a set of file operations. Tasks are the basic units in a working process's workflow. The mathematical definition of tasks is as follows. $AccessLog$ is the sequence of $file_x$.

$AccessLog = [file_1, file_2, \ldots, file_n]$.
$task_i = [file_j, file_{j+1}, \ldots, file_m], (1 \le j, m \le n)$,
where $AT(file_{k+1}) - AT(file_k) < TGBT$,
$(j \le k, k < m)$,
and $AT(file_j) - AT(file_{j-1}) \ge TGBT$,
and $AT(file_{m+1}) - AT(file_m) \ge TGBT$.
Here, $AT(file_x)$ means the access time stamp of $file_x$.

We calculate the similarity between tasks by using the similarity between files. The similarity between tasks is a metric representing the degree of matching of two tasks in terms of file operations. It is large when two tasks have numerous abstract file operations in common. Here, we use Dice's Coefficient [7] to calculate the degree of similarity. The formula for the similarity between $task_a$ and $task_b$ is

$$sim(task_a, task_b) = \frac{2|task_a \cap task_b|}{|task_a| + |task_b|}. \quad (4)$$

4.2.4 Abstraction of Tasks and Workflows

As described in Sect. 3.2 (3), by comparing each blank time between two continuous file operations with parameter TGBW, we separate the log into workflows. We set TGBW much longer than TGBT, so each workflow represents a sequence of tasks. The mathematical definition of workflows is as follows. $TaskSequence$ is the sequence of $task_x$.

$TaskSequence = [task_1, task_2, \ldots, task_n]$.
$workflow_i = [task_j, task_{j+1}, \ldots, task_m], (1 \le j, m \le n)$,
where $AT(FF(task_{k+1})) - AT(LF(task_k)) < TGBW$,
$(j \le k, k < m)$,
and $AT(FF(task_j)) - AT(LF(task_{j-1})) \ge TGBW$,
and $AT(FF(task_{m+1})) - AT(LF(task_m)) \ge TGBW$.
Here, $AT(file_x)$ means the access time stamp of $file_x$. $FF(task_x)$ means the first file in $task_x$. $LF(task_x)$ means the last file in $task_x$.

Based on the degree of similarity between tasks calculated in Sect. 4.2.3, we group tasks with a high degree of similarity together in a cluster as abstract tasks. We set two

thresholds to filter out small clusters and unnecessary items inside clusters. First, only when a cluster contains more than Minimum Number of Tasks (MNT) tasks, it will be treated as an abstract task. Second, for each file inside a cluster, only when its occurrences ratio is more than the Minimum Emergence Ratio (MER), this file will be included in the abstract task. For example, we assume that there are 3 tasks grouped into the same cluster. If a file was used only in 2 tasks, this file's occurrences ratio for this cluster is 0.67.

In our experiments, we set the parameters as MNT = 2 and MER = 0.5. For example, if tasks "a, b, c, d," "a, b, d," and "a, b" are grouped into the same cluster, not all of the files "a, b, c, d" would be put into the abstract task. We would remove items whose occurrences ratio in tasks is less than 0.5. In this case, only "a, b, d" would be put into the abstract task.

After the task abstraction, we simply replace each task in a workflow with the corresponding abstract task to obtain the abstract workflow.

4.3 Extraction of Frequent Abstract Workflows

To remove any infrequent abstract workflows created as described in Sect. 4.2.4, we extract those that appear frequently as frequent abstract workflows by parameter Minimum Support Threshold (MST) of the workflow sequence. Parameter MST means the minimum number of times a workflow appeared. Since percentages are used as Minimum Support sometimes, we denote it by MST to make clear the usage of integer numbers. In other words, we only accept workflows that appeared more frequently than MST, as frequent workflows. We use BIDE+ [4] as the sequential-pattern-mining algorithm. The reason for choosing BIDE+ is that it outputs only "closed sequential patterns," which are sequences containing no subsequences with the same support-count value.

5. Online Part

There are two modules in Online part. The aim of monitoring module is to find matching abstract tasks and frequent abstract workflows in database while monitors user's current operation. Recommendation module recommends files and operations on them (e.g. open, close) to the user.

5.1 Monitoring Module

The aim of this module is to infer the user's current task and workflow by monitoring the user's recent file-operations. As mentioned in Sect. 3.2, even when users are doing the same type of work, different files are usually being handled. For this reason, we abstract files being operated on currently by the user. We then estimate the abstract task and the frequent abstract workflow.

(1) **Abstraction of Files:**

We abstract files by the method explained in Sect. 4.2.2. If the user is accessing a file that has a corresponding file-cluster, we simply replace the being accessed file's filename in the log by the corresponding file cluster. However, in some cases, such as newly created files, some files do not own corresponding file-clusters. In such cases, we first replace the file being accessed by the most similar file that does have a corresponding file-cluster.

(2) **Estimation of Abstract Tasks:**

We estimate the current abstract task being operated on by the user from the abstract files being accessed. We group the abstract files being accessed into a task, and then compare this task with tasks in the database to find the most similar task. The comparison algorithm for abstract tasks is described in Sect. 4.2.3.

(3) **Estimation of Frequent Abstract Workflows:**

After estimation of the abstract task, we estimate the frequent abstract workflow. Because there are several abstract workflows that contain the estimated abstract task, we score each frequent abstract workflow in the database according to these three criteria listed below. We call the workflow being operated on by the user *workflow being accessed*, and a workflow in the database *workflow in database*.

- Degree of matching between *workflow being accessed* and *workflow in database*
- Frequent occurrence score for *workflow in database*
- Number of possible tasks in *workflow in database* for recommendation

Degree of matching between *workflow being accessed* **and** *workflow in database***:** A higher score is set if the degree of matching is high between two workflows. The score is higher if the matching subsequence is longer.

Frequent occurrence score for *workflow in database***:** A higher score is set if there are many occurrences of the frequent abstract workflow. A high frequent occurrence score means that the workflow pattern has been reused many times in the organization. We record this information while extracting workflows from the log.

Number of possible tasks in *workflow in database* **for recommendation:** The number of possible elements for recommendation in the workflow refers to the number of file operations that have not yet been undertaken by the user. When this number is high, because there is more information available for recommendation, a higher score is set.

In Fig. 4, frequent abstract workflow "1, 2, 3" → "4, 5" was estimated from the abstract task "1, 2, 3."

5.2 Recommendation Module

The aim of this module is to recommend files and operations based on the estimated abstract task and frequent abstract workflow. If we know about the user's current abstract task and frequent abstract workflow, we can identify and recommend subsequent abstract tasks. However, a frequent abstract workflow is a sequence of abstract tasks, while an ab-

Table 1 Information about extracted workflows.

	Proposed method	Comparative method
Number of records in log	7358	
Number of files	1421	
Number of file-clusters	645	-
Average size of file-clusters	2.203	-
Number of tasks	306	-
Average sequence length for tasks	2.680	-
Number of workflows	416	1375
Average sequence length for workflows	3.236	2.834
Average support-count value for workflows	5.370	3.864

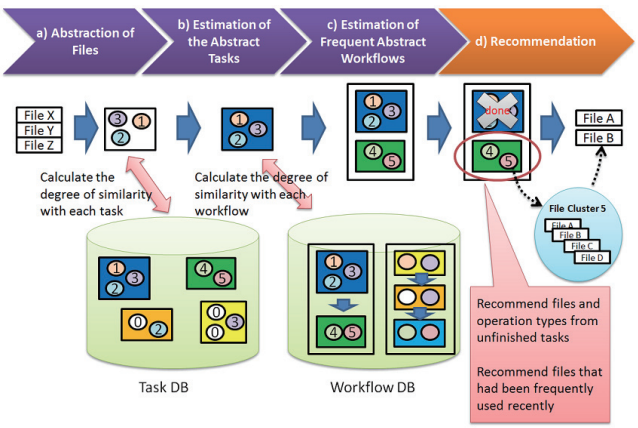

Fig. 4 Online part.

stract task is a set of pairs of operations and file-clusters. Each file-cluster contains several files. The problem is how to rank the recommendation candidates. Therefore, the proposed method extracts the last access time stamp of each file, ranking files belonging to the same file-cluster by most recent access time stamp.

6. Experiments

The goal of our experiment is to investigate the effectiveness of the features of our proposed concept, namely the abstraction of tasks and workflows. Before that, we tune parameters to investigate their influence on the recommendation results.

6.1 Comparative Method

Because we want to investigate the effectiveness of the features of our proposed concept, we set up a method that did not use abstract tasks for comparison. The comparative method simply extracts sequences of file operations as workflows and calculates those that are frequently used for recommendation. More specifically, the comparative method uses the same TGBW parameter as the proposed method to identify sequences of file operations from the log. The comparative method also applies the same sequential-pattern-mining algorithm BIDE+ to extract frequent workflows. In the recommendation algorithm, because the comparative method does not use the concept of a task, it matches directly the current workflow sequence with frequent workflows in the database for workflow estimation, and then recommends file operations that have not yet been used in this workflow.

6.2 Data

The experimental data came from actual file-access logs provided by a commercial organization. There are 22 users in our log data. The record term is about 8 months and 9917 records and 1750 files are recorded in the log data. The usernames, access time stamps, file paths, and operation type are recorded in the time series. Examples of operation types are "checkout," "checkin," "update," and "view."

We split the log in a ratio of 70% to 30%. The first 70% of the log was for learning tasks and workflows and the other 30% was for evaluation. Table 1 shows the log information about the first 70% part log. We divided the log for evaluation in terms of user IDs, using the same TGBW parameter to obtain workflows as used by the proposed method and created 151 test cases.

6.3 Evaluation Scheme

Each test case is a workflow sequence. For each test case, the n records (in the experiment, we set $n = 2$) from the beginning were input into our recommendation system to acquire a set of recommendation results. The remaining records in the test case were used as a correct answer set (the "examinees set"). By matching the examinees set and the results set, we obtained values for Precision, Recall, and F-measure.

A comparison between the proposed method and the comparative method was then made using the average values for all test cases.

6.4 Parameter Tuning

We investigate how these parameters influence the recommendation result and to find the optimal values for them. The basic approach to parameter tuning was to adjust the target parameter while keeping other parameters fixed. The value giving the highest average F-measure for all of the 151 test cases is then the optimal parameter value. We tune all parameters in our proposed method, and found parameters α, β, γ and δ are not sensitive while $TGBT$, $TGBW$ and MST are sensitive. Here we describe how these sensitive

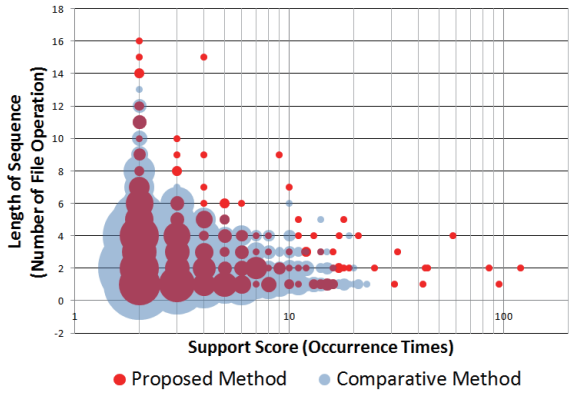

Fig. 5 Comparison of workflow characteristics.

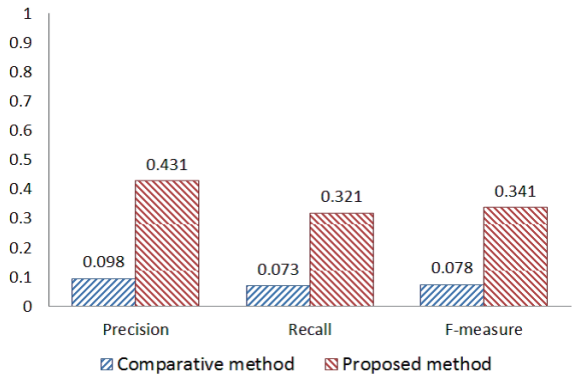

Fig. 6 Comparison of precision, recall, and F-measure for all test cases.

parameters influence the recommendation result.

Parameter $TGBT$ influences abstract tasks directly. The optimal TGBT value was 150 seconds. We obtained 2301 tasks for TGBT = 150. When TGBT was very large (900 seconds), we obtained only 217 large tasks. If an abstract task is too large, some unrelated file operations will inevitably be grouped in the same task, thereby reducing the Precision. For the $TGBW$ parameter, we tested from 10 minutes to 6 hours and found the optimal range was found to be 30 minutes–1.5 hours. Too large a $TGBW$ value will connect unrelated workflows into an excessively long workflow and reduce the Precision. On the other hand, too small a $TGBW$ value will split up a genuine workflow, thereby reducing the Recall value. The optimal MST value is 2. $MST = 1$ would imply that we ignore the process of extracting frequent abstract workflows, with all abstract workflows being used directly (as the frequent abstract workflows) in our proposed method. We found that, for $MST = 1$, the recommendation results were very poor in comparison to those for $MST = 2$. The reason is that there are many infrequent abstract workflows that are unsuitable for direct recommendation. Conversely, too high an MST value will eliminate too many abstract workflows, making some test cases unable to return recommendation results.

6.5 Comparison between the Proposed Method and the Comparative Method

Table 1 shows the information about tasks and workflows extracted from the logs when using these parameter values. We compared the workflows extracted by the proposed method with the comparative method; we found that the number of workflows for the proposed method was less than for the comparative method, but the average sequence length and average support-count values for the proposed method were greater. This is because similar workflows were settled on as the one frequent abstract workflow in the proposed method.

Furthermore, we can show the differences in workflows by using Fig. 5. In this bubble chart, the horizontal axis represents the occurrence value (support-count value) for workflows, the vertical axis represents the number of file operations in workflow sequences (sequence length), and the size of the bubbles represents the number of workflows. Note that comparative method's bubbles are more concentrated in the lower left corner than proposed method's bubbles, which means that the workflows extracted by the comparative method have smaller support-count values and shorter sequence lengths. Workflows with a small support-count value (a score of 2, for example) are more contingent and tend not to be reusable working patterns. In addition, workflows with small sequence lengths are undesirable because of the small amount of information available for the recommendation. Our proposed method can extract workflows with higher support-count values and greater sequence lengths than the comparative method. In other words, the workflows extracted by our proposed method have higher support-count values and are more suitable for making recommendations, thus, improving the recommendation accuracy greatly.

We now describe the recommendation results for the evaluation experiment that used test cases created from the remaining 30% of the log. Figure 6 shows the average Precision, Recall, and F-measure for all 151 test cases. Note that the proposed method performs much better than the comparative method for all metrics. In particular, the F-measure score was improved from 0.078 to 0.341. The reason for this large difference is that the numbers of test cases that can be recommended are overwhelmingly different. As shown in Fig. 7, because the comparative method can only recommend files that have been used in the past, only 25.8% of the test cases (39 out of 151) returned a result. On the other hand, because of our proposed method's abstract file operations, the proposed method can recommend files from similar file operations undertaken in the past. This enables more files to be available for recommendation. About 57.0% of the test cases (86 out of 151) returned results.

For the rest 43.0% test cases, our method did not give any recommendation. It is impossible and not necessary to make recommendation for all file accesses. If there is no pattern of meaningful workflows for the current file access, it is not appropriate to give any recommendation for it. In other

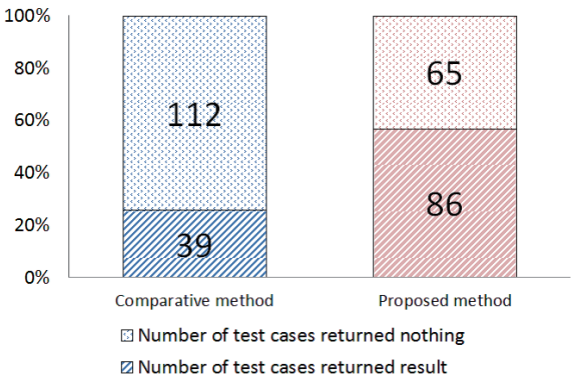

Fig. 7 Number of test cases that output results.

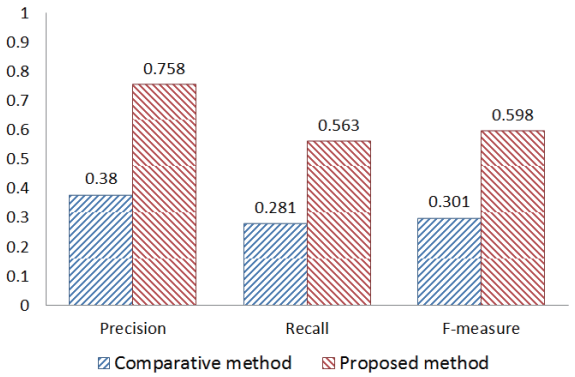

Fig. 8 Comparison of precision, recall, and f-measure for test cases that output results.

words, the proposed method finds out the general working process pattern and makes proper recommendation only in the case user doing some meaningful workflows. It means that we assume the documents are shared inside the organization and the recommend target user is executing some meaningful workflows. It is the applicable scope of our recommend method.

It is difficult to distinguish actual meaningful workflows in the test cases. However, the experiments demonstrate that 57.0% of file accesses in the test cases are chosen as the target of recommendation. It indicates that the file access logs used for the test cases are good examples for the applicable scope of our recommend method.

We should now focus on the test cases that returned results, when comparing the two methods. The experimental results are shown in Fig. 8, where the proposed method still performs considerably better than the comparative method. The reason for this is considered to be the quality of the workflows used in the recommendations. We therefore investigated the average support-count value for frequent abstract workflows that are used in the recommendations. The comparative method's average support-count value for frequent abstract workflows is 4.833, while the proposed method's value is 20.186. We can conclude that the proposed method's workflows have higher support-count values and more suitable for making recommendations, which will improve the recommendation accuracy greatly.

7. Related Work

There are a number of studies on extracting information from access logs such as Web-access and file-access logs to enable recommendations.

WRAPL [9] is for recommending Web pages using Web-access logs. WRAPL focuses on Web pages recommendation, not file recommendation. Even if we were to apply their method to file recommendations, their method does not make abstractions on files and there is no concept of tasks. It is difficult to obtain valid recommendation results. Okamoto et al. also proposed a Web-page recommendation method using Web-access logs [10]. By extracting patterns in combinations of multiple attributes of the accessed pages, their method can also recommend new Web pages. Although each Web page is abstracted in their study, there is no concept of abstract tasks.

Tanaka et al. proposed a personalized document recommendation system [11]. The system gives a recommendation based on these tendencies via collaborative filtering. In this system, documents viewed at the same time or within a short period of time are considered to be related to the same working unit for the same purpose. For this reason, they partition the access logs using the time gap between two records. In their experiments, the time gap was set to 30 minutes. We adopted this idea when extracting tasks from logs.

Another file recommendation system was proposed by Lai et al. [12]. Their work proposes recommendation methods based on the knowledge-flow (KF) model. KFs are similar to the workflows in our approach. There are two differences between their work and our proposed methods. First, the aims of file abstraction are different. In their work, each file is converted into a set of keywords inside it. Therefore, files with similar topics (keywords) are grouped together. As discussed in Sect. 4.2.1, it is not useful to group files for the same purpose of use together. On the other side, we group files not depending on topics, but the purpose of use, which is more suitable for extracting meaningful workflows. Second, in Lai et al., the definition of similarity between files is different from that in our method. In addition, their study groups similar files and then calculates KFs (workflows) directly, whereas our proposed method first groups similar files into tasks and then calculates the sequences of tasks as workflows. In other words, the definitions of workflows in their work and in our approach are different. By introducing the concept of abstract task, our method can extract patterns with difference in order.

Instead of file recommendations, Odagiri et al. proposed a method called FI [13] for file searching. The approach of the FI method is that files always being accessed simultaneously will have a high degree of similarity. This idea could be applied to improving our calculation algorithm for similarity between files in the future.

SUGOI [14] is for searching files using file access logs.

SUGOI finds related files using file-access logs. In their study, a task is defined as the file set containing related files in simultaneous use. However, their method does not perform abstraction on tasks, which is the main difference from our study. Their method considered the operations of Rename, Move, and Copy when calculating the degree of relevance between tasks, which is similar to our method for calculating the degree of similarity between files that have a copy relationship between them.

8. Conclusion and Future Work

In this paper, we propose a method to extract frequently used abstract-workflow patterns from the history, and recommends files and operations by monitoring the current workflow of the target user. There are two points in our proposed method. First, our proposed method is able to extract general patterns, which are more suitable for making recommendations by abstracting such files. Another point is that, the proposed method introduces abstract tasks to eliminate sequential relations inside tasks.

We have evaluated the proposed method using actual file-access logs. The results of these experiments demonstrate that our proposed method can extract workflow patterns with longer sequences and higher support-count values than a method that does not use the concepts of abstract tasks and workflows. Consequently, the F-measure of the recommendation results was improved significantly from 0.301 to 0.598.

In the future, we plan to consider a better algorithm for partitioning logs instead of simply using a fixed time. This might involve using information such as the frequency and type of operations.

References

[1] S. Feldman, J. Duhl, J.R. Marobella, and A. Crawford, "The hidden costs of information work," IDC WHITE PAPER, March 2005.
[2] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," IEEE Internet Comput., vol.7, no.1, pp.76–80, 2003.
[3] Q. Song, T. Kawabata, F. Itoh, Y. Watanabe, and H. Yokota, "A file recommendation method based on task workflow patterns using file-access logs," 24th International Conference on Database and Expert Systems Applications (DEXA 2013), vol.8056, pp.410–417, 2013.
[4] J. Wang and J. Han, "Bide: Efficient mining of frequent closed sequences," Proceedings of 20th International Conference on Data Engineering, pp.79–90, 2004.
[5] D.S. Hirschberg, "Algorithms for the longest common subsequence problem," Journal of the ACM (JACM), vol.24, no.4, pp.664–675, 1977.
[6] W. Heeringa, Measuring Dialect Pronunciation Differences using Levenshtein Distance, Doctoral dissertation. University of Groningen, 2004.
[7] L.R. Dice, "Measures of the amount of ecologic association between species," Ecology, vol.26, no.3, pp.279–302, 1945.
[8] J. Han, M. Kamber, and J. Pei, Data Mining: Concepts and Techniques, Second Edition, Morgan Kaufmann, 2006.
[9] R. Yamamoto, D. Kobayashi, T. Yoshihara, T. Kobayashi, and H. Yokota, "Analyses of the effects of utilizing web access log lcs for web page recommendation (in Japanese)," Journal of Information Processing Society of Japan, vol.48, no.SIG 11 (TOD 34), pp.38–48, 2007.
[10] H. Okamoto and H. Yokota, "Access log based web page recommendation using multiple attributes of web pages (in Japanese)," Proceedings of WebDB Forum 2009, 1A–4, Nov. 2009.
[11] H. Taguchi, S. Sakojo, and M. Iwata, "Personalized document recommendation for field engineers (in Japanese)," 3rd Forum on Data Engineering and Information Management (DEIM2011), B1–4, 2011.
[12] C. Lai and D. Liu, "Integrating knowledge flow mining and collaborative filtering to support document recommendation," Journal of Systems and Software, pp.2023–2037, 2009.
[13] K. Otagiri, Y. Watanabe, and H. Yokota, "Access-log analysis for virtual directory creation to restore files used in user's works (in Japanese)," IPSJ SIG Techical Report, vol.2009-DBS-148, no.4, pp.1–8, 2009.
[14] Y. Wu, K. Otagiri, Y. Watanabe, and H. Yokota, "A file search method based on intertask relationships derived from access frequency and rmc operations on files," 22nd International Conference on Database and Expert Systems Applications (DEXA 2011), vol.6860, pp.364–378, 2011.

Qiang Song received his B.E. and M.E. degrees from Tokyo Institute of Technology in 2011 and 2013. He is currently with the NTT DATA Corporation. He is engaged in research on data engineering and data mining.

Takayuki Kawabata received his B.E. and M.E. degrees from the University of Meiji in 2003 and 2005. He is with Canon Inc. since 2005. His research areas include document retrieval, knowledge processing and data mining.

Fumiaki Itoh received his B.E. and M.E. degrees from the University of Tokyo in 1988 and 1990. He is with Canon Inc. since 1990. His research areas include image and document retrieval, knowledge processing and data mining. He is a member of IPSJ and JSAI.

Yousuke Watanabe received the B.S, M.E. and Dr.E degrees in University of Tsukuba in 2001, 2003 and 2006. He was a research fellow of the Japan Society for the Promotion of Science from April 2003 to March 2006. He was a researcher of the Japan Science and Technology Agency (JST/CREST) from April 2006 to June 2008. He is currently an assistant professor at Global Scientific Information and Computing Center, Tokyo Institute of Technology. His research issues are Information Integration, Data Stream Processing and Desktop Search.

Haruo Yokota received his B.E., M.E. and Dr.Eng. degrees from Tokyo Institute of Technology in 1980, 1982, and 1991, respectively. He joined Fujitsu Ltd. in 1982, and was a researcher at ICOT for the Japanese 5th Generation Computer Project from 1982 to 1986, and at Fujitsu Laboratories Ltd. from 1986 to 1992. From 1992 to 1998, he was an Associate Professor at Japan Advanced Institute of Science and Technology (JAIST). He is currently a Professor at the Department of Computer Science in Tokyo Institute of Technology. His research interests include the general research areas of data engineering, information storage systems, and the dependable computing. He was a chair of ACM SIGMOD Japan Chapter and a trustee member of IPSJ. He is a Vice Chair of the Database Society of Japan (DBSJ), the Editor-in-Chief of Journal of Information Processing, an associate editor of the VLDB Journal, a fellow of IEICE and IPSJ, and a member of JSAI, IEEE, IEEE-CS, ACM, and ACM-SIGMOD.