A Recommendation System for Software Function Discovery

Naoki Ohsugi, Akito Monden, Ken-ichi Matsumoto
Graduate School of Information Science, Nara Institute of Science and Technology
8916-5 Takayama, Ikoma, Nara 630-0101, Japan
+81 743 72 5312
{ naoki-o, akito-m, matumoto } @is.aist-nara.ac.jp

Abstract

Since many of today's application software provide users with too many functions, the users sometimes cannot find the useful functions. This paper proposes a recommendation system based on a collaborative filtering approach to let users discover useful functions at low cost for the purpose of improving the user's productivity in using application software. The proposed system automatically collects histories of software function execution (usage histories) from many users through the Internet. Based on the collaborative filtering approach, collected histories are used to recommend the user a set of candidate functions that may be useful to the individual user. This paper illustrates conventional filtering algorithms and proposes a new algorithm suitable for recommendation of software functions. The result of an experiment with a prototype recommendation system showed that the average ndpm of our algorithm was smaller than that of the conventional algorithms; and, it also showed that the standard deviation of ndpm of our algorithm was smaller than that of the conventional algorithms. Furthermore, while every conventional algorithm had a case whose recommendation was worse than the random algorithm, our algorithm did not.

Keywords: Software Usability, User Interfaces, CSCW, Collaborative Filtering.

1. Introduction

Today, application software like word processing software and spreadsheet software are getting more complicated and providing more functions. For example, Microsoft Office applications have the following total number of menu items and buttons on toolbars: MS-Word 2000 has 663, and MS-Word 2002 has 782. MS-Excel 2000 has 694, and MS-Excel 2002 has 812. These facts indicate that the number of software functions in application software is rapidly growing year by year.

The growth of software functions is a bad influence on software usability in that the useful functions are hidden among many other functions. All users do not learn eagerly the new functions even if these functions are useful for them and improve their productivity in using application software [1][2]. It is because costs to find and learn new functions of the application software and to support it are very large for both users and developers.

This paper proposes a recommendation system based on a collaborative filtering approach to let users discover useful functions at low cost for the purpose of improving the user's productivity in using application software. The proposed system automatically collects histories of software function execution (usage histories) from many users through the Internet. Based on the collaborative filtering approach, collected histories are used to recommend the user a set of candidate functions that may be useful to the individual user.

However, since conventional collaborative filtering algorithms are not designed for discovering useful functions from usage histories of application software, the usefulness of the algorithms are not clear if we apply them to our recommendation system. This paper firstly illustrates some well-known conventional filtering algorithms, which have been proposed for different targets such as music items and e-commerce goods. Afterward we describe their problems arising when we apply them to the software functions. Finally, we propose a new algorithm suitable for recommendation of software functions, and show the result of an experiment to evaluate the algorithm.

In what follows, Section 2 introduces related works. Section 3 shows the current status of a prototype implementation of the proposed system for Microsoft Office 2000 applications. Section 4 describes conventional filtering algorithms and their problems arising when we apply them to the software functions. Section 5 proposes a new algorithm that solves their problems. Section 6 describes an empirical evaluation of the proposed algorithm. Section 7 discusses the usefulness of the proposed algorithm. In the end, Section 8 shows conclusions and future topics.

2. Related Work

2.1. Collecting Software Usage Histories

For the various purposes, systems that collect software usage histories have been developed [5]. According the person who users the history, such systems can be classified into the following two types.

- *Systems for software developers*: This type of system helps developers in enhancing the usability of software they developed.
- *Systems for software users*: This type of system helps users in enhancing the usability of software they are using.

Finlay and Harrison [3] proposed a *system for software developers*. It collects software usage histories, to improve usability of the user interface of software. Improvement procedure is as follows. At first, the developer of the software inputs "a correct software usage procedure", to the system. Next, the system collects "actual software usage histories" from users of that software. Next, in order to detect bad parts in the user interface, their system compares "a correct software usage procedure" and "practical software usage histories," and detects the users' erroneous usages. At last, the developer improves the detected bad parts so that the users will not give erroneous usages.

Yano et al. [9] developed the Sharlok that was a *system for software users*. It collects software usage histories in order to provide opportunities to begin collaboration among users. The Sharlok automatically collects each user's usage histories in the background process. And then, it provides each user with other users' usage histories so that the user can recognize other users' operations and can begin discussion about their operations. Like the Sharlok, our system also collects usage histories from many users, and feedbacks them to each user.

2.2. Collaborative Filtering

The Collaborative Filtering is considered as a key technology of recommendation systems, which provide the user a set of candidate items that may be useful or preferable to the individual user, from a large amount of items [6].

Huberman [7] developed the Beehive. Each user of the Beehive joins to some group associated with a certain topic so that the user can receive information recommended by other users in that group. The user who receives the recommended information also must recommend information to other users in order to keep on joining the group. If the user does not recommend any information for a certain period, that user will be removed from the group. One of the related researches is the Tapestry, developed by Goldberg et al. [4].

It is said that the first recommendation system that automatically selects the items to be recommended is the GroupLens [10][11]. The GroupLens recommends the user a set of candidate Netnews articles that may be preferred by the individual user, from a large amount of Usenet news articles. Each user of GroupLens explicitly rates each recommended Netnews article after reading it, with using a scale of one (bad) to five (good). Based on the user ratings, the GroupLens compute similarity among tendencies of users' preferences. Then, high-rated article by a particular user is recommended to other user who has the high similarity with that user. This recommendation procedure is a basis of today's recommendation systems. Based on the GroupLens' approach, many filtering algorithms have been proposed [13]. However, all these conventional algorithms are not designed for recommending software functions. In this paper, also based on GroupLens' approach, we propose an algorithm suitable for software functions.

3. Recommendation for Software Functions

In this section, recommendation system for software functions is described as follows; subsection 3.1 describes about usage history format of our system. Subsection 3.2 illustrates about architecture of our system. And subsection 3.3 describes about collaborative filtering procedure on our system.

3.1. Usage History Format

An instance of the usage history in our system is shown in Figure 1. In the usage history, each line has captions of clicked menu items and buttons on toolbars,

2002/02/03 18:50:41 Formatting->Font(&F)...

2002/02/03 18:50:45 Formatting->Font Size(&F)

2002/02/03 18:50:48 Standard->Centering(&C)

2002/02/03 18:51:16 File->Save As(&S)...

2002/02/03 18:51:23 File->Exit(&X)

Figure 1. An example of the usage history

with the time of the clicking. And the lines are ordered in time series. In addition, each caption of item in each line has a parent menu caption to represent the position of clicked place. Such parent menu caption is recorded in the left of "->". For example, the first line of usage history of Figure 1, clicked menu item "Font(&F)..." is put as a submenu of parent menu "Formatting". Meanwhile, each caption of clicked button in each line has a name of toolbar that owns the button. For example, the third line of the usage history in Figure 1, clicked button "Centering(&C)" is put upon the toolbar "Standard".

3.2. Architecture of the Recommendation System

Figure 2 shows a summarized architecture of the recommendation system. Our system consists of *Usage History Server* and some *User's Computers* connected with the server via the Internet. Usage History Server has a database to receive and store usage histories sent from User's Computers. Each User's Computer has three software components, *Usage History Collector*, *Usage History Receiver* and *Functions Recommender*, which are described below.

- *Usage History Collector*: Software component that collects each user's usage history of particular application software, and send it to the Usage History Server. Usage History Collector works automatically, as a background process, and does not disturb user's work.
- *Usage History Receiver*: Software component that receives all users' usage histories sent from Usage History Server. Usage History Receiver passes them to the *Functions Recommender*.
- *Functions Recommender*: Software component that applies collaborative filtering to all users' usage histories to find useful functions for individual user. Such collaborative filtering procedure is described in the next subsection.

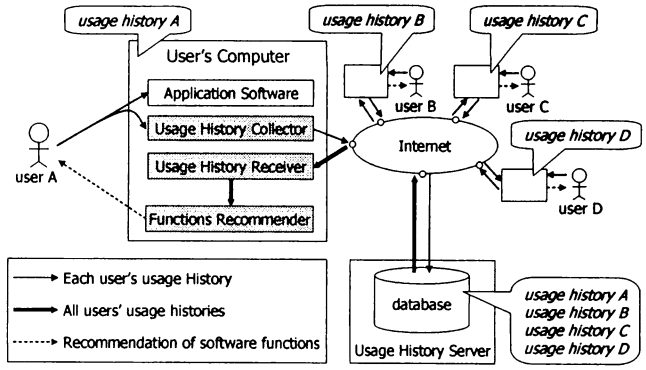

Figure 2. Recommendation system for software functions

3.3. Collaborative Filtering Procedure

Figure 3 illustrates an instance of collaborative filtering procedure of our system. Here the situation is that the user A and the user B use the system, and the system is applying collaborative filtering to make a recommendation to the user A. In Figure 3, "Name" columns show names of clicked menu items or buttons. "Frequency" columns show those ratios in total times. Each process in collaborative filtering procedure is illustrated as a white arrow that has a number like "P1". More details of the processes are described below, which correspond to the numbers in white arrows.

P1: The system counts the number of clicks and frequency of each function in "Each user's usage history" to make "Each user's summarized usage history".

P2: The system counts clicked number and frequency of each function in all users, in order to make "All users' summarized usage history".

P3: The system applies collaborative filtering algorithm described in section 4 and 5 to "All users' summarized usage history", in order to compute *rating* for each function in making "Recommendation for user A". The *rating* shows how useful the function is for the individual user. A function having higher value is more useful than that having lower value in the rating. Let r_{ak} denotes a rating for the user A on the function K.

Functions in "Recommendation for user A" are presented to the user A, depending on ratings. High rated function is presented preferentially if it has not been used by the user.

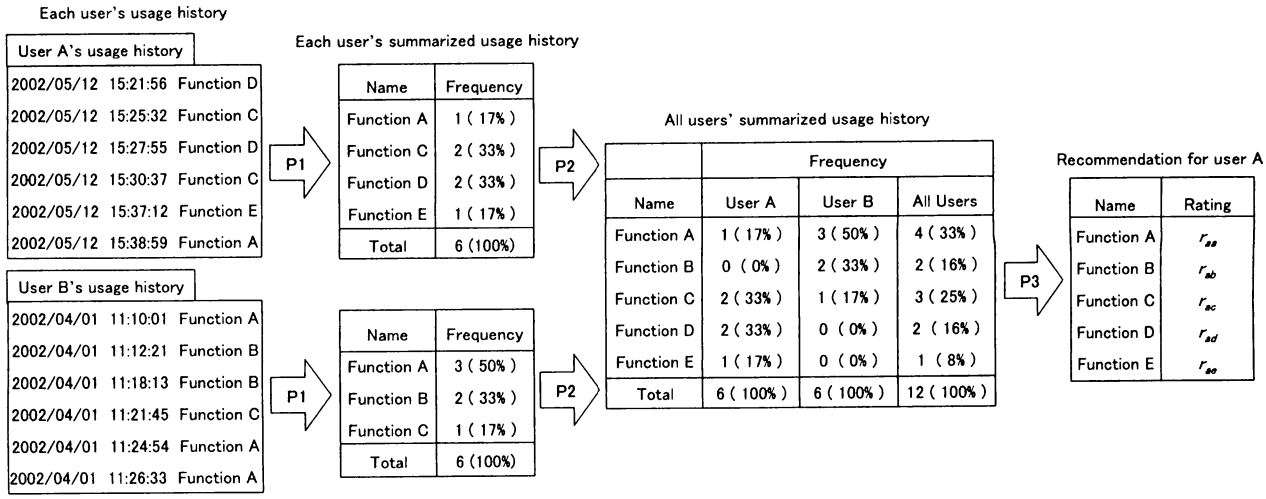

Figure 3. An example of collaborative filtering procedure on the recommendation system

4. Collaborative Filtering Algorithms

In what follows, subsection 4.1 describes some conventional collaborative filtering algorithms. Subsection 4.2 explains problems in applying them to the usage history of our system.

4.1. Conventional Algorithms

Today, many collaborative filtering algorithms are proposed by many researches. In this section, some typical algorithms are described. They are employed in the experiment in section 6.

4.1.1. User Count Algorithm

Matsumoto et al. [8] employed a simple algorithm called the *User Count Algorithm* in developing the CLAS system, which is a prototype of our system. Let U and E denote the set of all users and all functions, and e_{ij} is the number of executions of function j by the user i. The rating r_{ak} for the user a on function k is:

$$r_{ak} = \sum_{i \in U} t_{ik} + \frac{\sum_{i \in U} e_{ik}}{1 + \sum_{i \in U} \sum_{j \in E} e_{ij}} \quad (1)$$

where t_{ik} is calculated by the following equation.

$$t_{ik} = \begin{cases} 0 \, (e_{ik} = 0) \\ 1 \, (e_{ik} \geq 1) \end{cases} \quad (2)$$

The equation (2) denotes t_{ik} is 0 if the user i has never use the function k. Otherwise it is 1. So, the first term of the equation (1) equals to the number of users who has executed the function k. And the range of the value of the second term is [0, 1), corresponding to the number of executions of all users. As a result, "Recommendation for user" will be a list of functions in descending order that are sorted by the number of users who has executed the function at first, and by the number of executions of all users in the second.

4.1.2. Base Case Algorithm

The *Base Case Algorithm* is commonly used in previous collaborative filtering researches as a basis of evaluating the performance of new algorithms. This algorithm was first used by Shardanand and Maes [14], to evaluate their music recommendation system *Ringo*. Each user of the *Ringo* explicitly votes music that the user has heard, with using a scale of 1 (bad) to 7 (good). Let U denotes the set of all users, and v_{ik} is a voting of user i on music k. The rating r_{ak} for user a on music k is:

$$r_{ak} = \frac{\sum_{i \in U} v_{ik}}{|U|} \quad (3)$$

In the equation (3), r_{ak} is the average value of votes to music k among all users.

4.1.3. User Similarity Computation Algorithm

The *User Similarity Computation Algorithm* was proposed by Resnick et al. [11] in GroupLens research,

which is one of the basic algorithms in collaborative filtering researches. For example, Sarwar et al. [12] proposed faster algorithm based on this algorithm, to apply to their commodity recommendation system in e-commerce.

Here the situation is that system computes rating of item k for user a. At first, if I_i is the set of item on which user i has voted, then the mean vote for user i is calculated as:

$$\bar{v}_i = \frac{1}{|I_i|}\sum_{j \in I_i} v_{ij} \tag{4}$$

Next, the similarity $c(a, i)$ between each user i and the user a is calculated as:

$$c_{(a,i)} = \frac{\sum_{j \in I_a \cap I_i} \left(v_{aj} - \bar{v}_a\right)\left(v_{ij} - \bar{v}_i\right)}{\sqrt{\sum_{j \in I_a \cap I_i} \left(v_{aj} - \bar{v}_a\right)^2 \sum_{j \in I_a \cap I_i} \left(v_{ij} - \bar{v}_i\right)^2}} \tag{5}$$

At last, rating r_{ak} of item k for user a is calculated as:

$$r_{ak} = \bar{v}_a + \sum_{i \in U} c(a,i)\left(v_{ik} - \bar{v}_i\right) \tag{6}$$

The equation (5) is a correlation coefficient between user a and i. And its range of value is [-1, 1]. The first term of the equation (6) is an average voting of user a and the second term is a weighted sum of the votes of all users except user a. Results of the equation (6) depend on the votes of other users who are similar to user a, so that high voted items by similar other users get high rating.

4.2. Problems in Applying Conventional Algorithms to Usage History

4.2.1. User Count Algorithm

In the *User Count Algorithm*, functions used by many users are recommended preferentially. So, the user who has common tendency with many other users in using functions can get useful recommendations. But otherwise the user cannot get them since the similarity between users is not considered. Furthermore, some useful functions that are not used by many users are not recommended in this algorithm.

Now, we assume a system recommends useful functions of word processing software, and the functions related to "creating and editing tables" are frequently used by many users. Then, such functions are recommended to the users even if they will never use word processing software to edit tables.

4.2.2. Base Case Algorithm

The *Base Case Algorithm* described in the subsection 4.1.2 cannot be applied to usage histories, since it was designed for voting of a discrete numerical scale. So, we transform it, as follows.

At first, the equation (3) is transformed by replacing the voting v_{ik} of user i on music k with the frequency f_{ik} of user i on function k. If e_{ik} is the executed times of user i on function k, and E is the set of all functions. The frequency f_{ik} is:

$$f_{ik} = \frac{e_{ik}}{\sum_{j \in E} e_{ij}} \tag{7}$$

Then, the equation (3) is transformed to the following equation:

$$r_{ak} = \frac{\sum_{i \in U} f_{ik}}{|U|} \tag{8}$$

In this algorithm, functions frequently used by many users are recommended preferentially. So, the recommendation of this algorithm is similar to the *User Count Algorithm*. And this algorithm has the same problems with the *User Count Algorithm*.

4.2.3. User Similarity Computation Algorithm

The *User Similarity Computation Algorithm* described in the subsection 4.1.3 also cannot be applied to usage histories like the *Base Case Algorithm*. So, we transform it, as follows.

At first, the equation (4) is transformed by replacing the voting v_{ij} of user i on music j with the frequency f_{ij} of user i on function j given by the equation (7). Then, the equation (4) is transformed to the equation (9) that calculates the average frequency $\bar{f}_i$ of user i as follows:

$$\bar{f}_i = \frac{1}{|E_i|}\sum_{j \in E_i} f_{ij} = \frac{1}{|E_i|} \quad \left(\because \sum_{j \in E_i} f_{ij} = 1.0\right) \tag{9}$$

where E_i is the set of functions used by user i. Next, equation (5) is transformed to the following equation:

$$c(a,i) = \frac{\sum_{j \in I_a \cap I_i} \left(f_{aj} - \bar{f}_a\right)\left(f_{ij} - \bar{f}_i\right)}{\sqrt{\sum_{j \in I_a \cap I_i} \left(f_{aj} - \bar{f}_a\right)^2 \sum_{j \in I_a \cap I_i} \left(f_{ij} - \bar{f}_i\right)^2}} \tag{10}$$

At last, equation (6) is transformed to following equation:

$$r_{ak} = \overline{f}_a + \sum_{i \in U} c(a,i)\left(f_{ik} - \overline{f}_i\right) \quad (11)$$

In this algorithm, the functions that are frequently used by other users who have similar tendency with the user *a*, are recommended preferentially. This algorithm does not have the same problem with the *User Count Algorithm* and the *Base Case Algorithm* since the similarity between users is considered.

However, there is another problem. The similarity depends on the frequencies of the frequently executed functions, excessively, because the similarity is calculated from frequencies of executed functions as described in the equation (10). In our investigation, the most frequently used functions in Microsoft Word 2000 were "Save", "Undo" and "Redo". And the frequencies of their executed counts in some users' usage histories were 80% or more. Therefore, the other functions did not affect the similarity computation. This is one of the serious problems in employing the User Similarity Computation Algorithm because when we use application software, this situation will happen in many cases. So, we propose a new algorithm to solve this problem, in Section 5.

5. Proposed Algorithm

We propose the *Distance-based User Similarity Computation Algorithm*, to solve the problem described in 4.2.3. In the proposed algorithm, the similarity between users is calculated by the similarity of the order of functions sorted by execution frequency. In fact, the similarity *c(a, i)* in equation (10), is replaced with:

$$c(a,i) = 1 - 2 \times \frac{\sum_{j \in E_a \cup E_i} \sum_{k \in E_a \cup E_i} dist\left(f_{aj}, f_{ak}, f_{ij}, f_{ik}\right)}{\sum_{j \in E_a \cup E_i} \sum_{k \in E_a \cup E_i} distNormalizer\left(f_{aj}, f_{ak}\right)} \quad (12)$$

where, E_a and E_i are the sets of all executed functions by user *a* and user *i* respectively. And, the following functional equations are used.

$$dist\left(f_{aj}, f_{ak}, f_{ij}, f_{ik}\right) = \begin{cases} 0 & \left(f_{aj} > f_{ak}\right) \wedge \left(f_{ij} > f_{ik}\right) \\ & \vee \left(f_{aj} = f_{ak}\right) \wedge \left(f_{ij} = f_{ik}\right) \\ & \vee \left(f_{aj} < f_{ak}\right) \wedge \left(f_{ij} < f_{ik}\right) \\ 1 & \left(f_{aj} = f_{ak}\right) \wedge \left(f_{ij} \neq f_{ik}\right) \\ & \vee \left(f_{aj} \neq f_{ak}\right) \wedge \left(f_{ij} = f_{ik}\right) \\ 2 & \left(f_{aj} > f_{ak}\right) \wedge \left(f_{ij} < f_{ik}\right) \\ & \vee \left(f_{aj} < f_{ak}\right) \wedge \left(f_{ij} > f_{ik}\right) \end{cases} \quad (13)$$

$$distNormalizer\left(f_{aj}, f_{ak}\right) = \begin{cases} 1 & \left(f_{aj} = f_{ak}\right) \\ 2 & \left(f_{aj} \neq f_{ak}\right) \end{cases} \quad (14)$$

where in the equation (13), $x \wedge y$ denotes "*x* AND *y*" of the logical operator, and $x \vee y$ denotes "*x* OR *y*".

In this algorithm, the similarities between users are calculated from the distance function *dist* defined by Yao [15]. The numerator in the second term of the equation (12) is the sum of results of the distance function *dist* on the set of executed functions E_a and E_i by user *i* and user *a*. We assume that frequencies f_{ij} of user *i* on functions *j* that are $(j \in E_a) \wedge (j \notin E_i)$, and frequencies f_{aj} of the user *a* on functions *j* that are $(j \notin E_a) \wedge (j \in E_i)$, are 0.

The denominator in the second term of the equation (12) normalizes the numerator to the range [0, 1]. So the expression (12) is normalized to the range [-1, 1], just like the expression (5) and (10).

In this algorithm, the similarity between users is considered like the *User Similarity Computation Algorithm*. And this algorithm does not have the same problem with the *User Similarity Algorithm* since frequencies of executions are not directly used in the user similarity calculation.

6. Experiment for Performance Evaluation

6.1. Outline of Experiment

We have conducted an experiment to evaluate the performance of algorithms described in Section 4 and 5. We used software usage histories of Microsoft Word 2000 collected by the system described in Figure 2. The usage histories were collected from 6 users for 3 to 22 months (the average is 11 months). Each user used 62 to 108 kind of functions (the average is 90), and the 209 kind of functions were used 6 users in total.

We employed the Yao's *ndpm* measure [15] that is an evaluation metrics to measure the system performance. *Ndpm* is calculated by comparing the difference between "order of items in ideal recommendation for the user" and "order of items in system's recommendation for the user". The lower number of *ndpm* denotes that system provides better recommendation.

6.2. Experiment Protocol

Figure 4 shows an experiment protocol of evaluating the performance of the recommendation for user *A*. Each process in the experiment is illustrated as a white arrow that has a number like "E1". More details of the processes

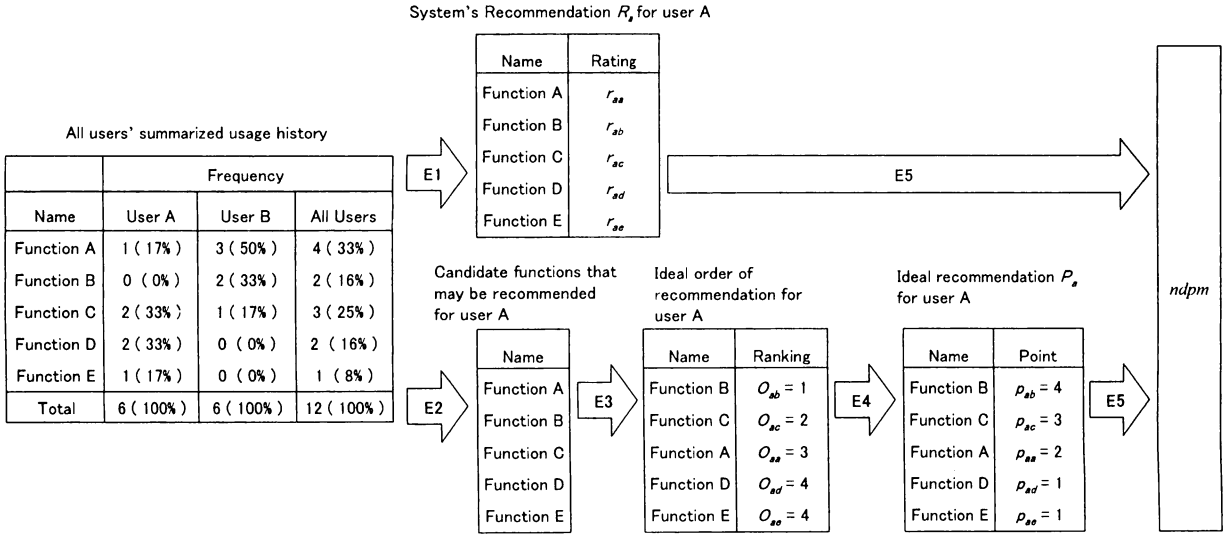

Figure 4. An experiment protocol in evaluating the recommendation for user A

are described below, which correspond to the numbers in the white arrows.

E1: Each collaborative filtering algorithm described in section 4 and 5 is applied to "All users' summarized usage history" to make "System's Recommendation R_a for user A". In Figure 4, r_{ak} denotes the rating of function K for user A.

E2: The function names are taken out from "All users' summarized usage history", to make "Candidate functions that may be recommended for user A".

E3: The user A is asked to align the function names in "Candidate functions that may be recommended for user A" in the order of the following criteria, to make "Ideal order of recommendation for user A". We assume o_{ai} is the rank of it function i determined by user A. The first rank is 1, and is increased corresponding to the place in the ordered list. So the second order is 2, the third is 3, and so on. And the last rank is n.

At first, functions that the user A already knows are placed in the last rank n because they are meaningless in the recommendation. Next, functions that were already known but not used are placed into the second last rank $n - 1$ because these will not be used by user A even if it is recommended. At last, functions that the user A did not know are placed between the first rank and the third last rank, corresponding to the estimates of the frequencies of using the functions. Functions that have similar estimates are placed into the same rank.

E4: Each ideal rating p_{ai} of function i for the user A is calculated by the following equation, to make "Ideal recommendation P_a for user A".

$$p_{ai} = n - o_{ai} + 1 \tag{15}$$

E5: The *ndpm* is calculated from "Ideal recommendation P_a for user A" and "System's Recommendation R_a for user A" by the following equation.

$$ndpm(P_a, R_a) = \frac{\sum_{j \in P_a \cup R_a} \sum_{k \in P_a \cup R_a} dpm(p_{aj}, p_{ak}, r_{aj}, r_{ak})}{\sum_{j \in P_a \cup R_a} \sum_{k \in P_a \cup R_a} dpmNormalizer(p_{aj}, p_{ak})} \tag{16}$$

where

$$dpm(p_{aj}, p_{ak}, r_{aj}, r_{ak}) = \begin{cases} 0 & \left((p_{aj} > p_{ak}) \wedge (r_{aj} > r_{ak})\right) \\ & \vee \left((p_{aj} < p_{ak}) \wedge (r_{aj} < r_{ak})\right) \\ 1 & \left((p_{aj} \neq p_{ak}) \wedge (r_{aj} = r_{ak})\right) \\ 2 & \left((p_{aj} > p_{aj}) \wedge (r_{aj} < r_{ak})\right) \\ & \vee \left((p_{aj} < p_{aj}) \wedge (r_{aj} > r_{ak})\right) \end{cases} \tag{17}$$

$$dpmNormalizer(p_{aj}, p_{ak}) = \begin{cases} 0 & (p_{aj} = p_{ak}) \\ 2 & (p_{aj} \neq p_{ak}) \end{cases} \tag{18}$$

The numerator of the equation (16) is the sum of results of the function *dpm* on the set of ideal recommendation P_a and the set of system's

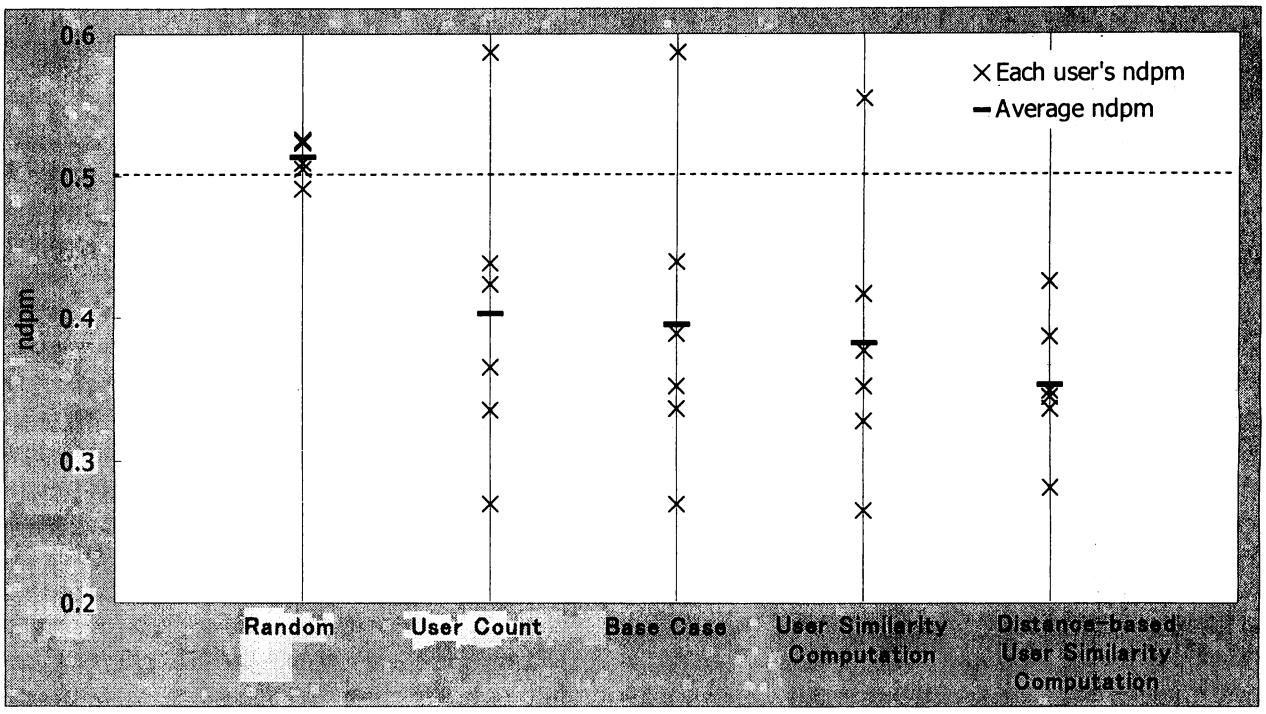

Figure 5. *Ndpm* of each algorithm

recommendation R_a. We assume that system's ratings r_{aj} for the user A on functions j that are $(j \in P_a) \wedge (j \notin R_a)$, and ideal ratings p_{aj} of the user A on functions j that are $(j \notin P_a) \wedge (j \in R_a)$, are 0. The denominator of the equation (16) normalizes the numerator to the range [0, 1].

6.3. Result of Experiment

Figure 5 shows the result of the experiment in a scatter graph. The y-axis of the graph indicates the values of *ndpm*. And the x-axis of the graph indicates each algorithm described in section 4 and 5. The values of *ndpm* were calculated from "Ideal recommendation P_i for each user i" and "System's recommendation R_i for user i" made by each algorithm. The lower *ndpm* value denotes that the algorithm provides better recommendation. A symbol "×" is each user's *ndpm* value on each algorithm. And symbol "–" is the average *ndpm* of 6 users in each algorithm.

The result for each algorithm is as follows.

- ***Random***: In this paper, randomly calculated recommendations are used for showing the worst case of the recommendation. The theoretical value of *ndpm* of random algorithm is 0.5. In our experiment, the average value of *ndpm* was 0.514. It was nearly equal to the theoretical value. The standard deviation of *ndpm* was very small (0.014).
- ***User Count***: The average value of *ndpm* was 0.403. And the standard deviation of *ndpm* was 0.109. One user had worse recommendation than the *Random Algorithm*.
- ***Base Case***: The average value of *ndpm* was 0.355. And the standard deviation of *ndpm* values was 0.109. One user had worse recommendation than the *Random Algorithm*.
- ***User Similarity Computation***: The average value of *ndpm* was 0.383. And the standard deviation of *ndpm* was 0.099. One user had worse recommendation than the *Random Algorithm*.
- ***Distance-based User Similarity Computation***: The average value of *ndpm* was 0.355. And the standard deviation of *ndpm* was 0.049. All of users had better recommendations than the *Random Algorithm*. Furthermore, 4 out of 6 users had better recommendations than all the other algorithms.

7. Discussion

Although we have collected function execution histories for a long time in our experiment, the number of subjects may not be sufficient to generalize the result. However, from the following viewpoints, the result of the experiment suggests that the proposed filtering algorithm solves the problems of conventional algorithms, and that it is useful for the recommendation of software functions:

- The average *ndpm* of the proposed algorithm was smaller than that of the conventional algorithms. This indicates that our algorithm has a potential to provide better recommendation than the conventional algorithms.
- The standard deviation of *ndpm* of the proposed algorithm was smaller than that of the conventional algorithms (excluding random algorithms). This indicates that our algorithm has a potential to constantly provide good recommendation to every users.
- While every conventional algorithm had a case whose recommendation was worse than the random algorithm, our algorithm did not. This suggests that our algorithm is superior to the conventional algorithms in providing better recommendation than the random algorithm.

8. Conclusion

In this paper we proposed a recommendation system based on a collaborative filtering approach to let users discover useful functions at low cost for the purpose of improving the user's productivity in using application software. We firstly illustrated some well-known conventional filtering algorithms, which have been proposed for different targets such as music items and e-commerce goods. Next we described their problems arising when we apply them to the software functions. Finally, we proposed a new algorithm suitable for recommendation of software functions, and conducted an experiment to evaluate the algorithm.

The result of the experiment showed that the average *ndpm* of the proposed algorithm was smaller than that of the conventional algorithms. And, it also showed that the standard deviation of *ndpm* of the proposed algorithm was smaller than that of the conventional algorithms. Furthermore, while every conventional algorithm had a case whose recommendation was worse than the random algorithm, our algorithm did not. These results suggest that our filtering algorithm has a potential to provide better recommendation of software functions than the conventional algorithms.

Although we have collected function execution histories for a long time (from 3 months up to 22 months) in our experiment, the number of subjects (6 users) may not be sufficient for the evaluation. Further experiments are needed to confirm the usefulness of our algorithms.

In the future we must consider about collecting execution histories of short-cut keys. In this paper, we have collected the executions of menu items and short-cut icons. However, in much application software, we can also execute functions by pressing short-cut keys. Typically, short-cut keys enable us to execute commonly-used functions such as "copy", "cut", and "paste." Since these functions are usually executed by short-cut keys, we are planning to collect the executions of them in the future.

Another future topic is about the criterion to distinguish non-recommended functions. In this paper, we regarded that "a user already knows a function if that function has been executed once in the past by that user." Such functions are not recommended to the user in our system. However, this criterion may not be valid in case a user randomly executes unknown functions but he/she does not learn their usage. In the future, we are to change this criterion to more valid ones to enhance the performance of our recommendation system.

References

[1] J.M. Carroll, and M.B. Rosson, "Paradox of the Active User", *Interfacing Thought: Cognitive Aspect of Human-Computer Interaction*, MIT Press, Cambridge, MA, 1987.

[2] A. Cypher, "Eager: Programming Repetitive Tasks by Example", In *Proc. of the ACM Conference on Human Factors in Computing Systems*, April 1991, pp.33-39.

[3] J. Finlay, and M. Harrison, "Pattern Recognition and Interaction Models", In *Proc. of the INTERACT '90*, pp.149-154, August 1990.

[4] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry, "Using Collaborative Filtering to Weave an Information Tapestry", *Communications of the ACM*, Vol.35, No.12, December 1992, pp.61-70.

[5] D.M. Hilbert, and D. F. Redmiles, "Extracting Usability

Information from User Interface Events", *ACM Computing Surveys*, Vol.32, No.4, December 2000, pp.384-421.

[6] W. Hill, L. Stead, M. Rosenstein, and G. Furnas, "Recommending and Evaluating Choices in a Virtual Community of Use", In *Proc. of the 1995 Conference on Human Factors in Computing Systems (CHI'95)*, 1995, pp.194-201.

[7] B. A. Huberman, and M. Kaminsky, "Beehive: A System for Cooperative Filtering and Sharing of Information", *Computer Human Interaction*, 1996, pp.210-217.

[8] K. Matsumoto, S. Morisaki, A. Monden, K. Torii, "CLAS: An Approach for Full Use of Application Software", *Nara Institute of Science and Technology, Information Science Technical Report: TR2001002*, 2001.

[9] Y. Yano, H. Ogata, and J. Qun, "Sharlok: Combining a Collaborative Learning Environment and an Active Database", *Advanced Database Systems for Integration of Media and User Environments '98, Advanced Database Research and Development Series*, World Scientific, Vol.9, pp.329-332, 1998.

[10] D. M. Pennock, E. Horvitz, S. Lawrence, and C.L. Giles, "Collaborative Filtering by Personality Diagnosis: A Hybrid Memory- and Model-Based Approach", In *Proc. of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-2000)*, 2000, pp.473-480.

[11] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens: An Open Architecture for Collaborative Filtering of Netnews", In *Proc. of CSCW '94*, 1994, pp.175-186.

[12] B.M. Sarwar, G. Karypis, J.A. Konstan, and J. Riedl, "Analysis of Recommendation Algorithms for E-Commerce", In *Proc. of the ACM Conference on ECommerce (EC00)*, Minneapolis, MN, 2000, pp. 158-167.

[13] B.M. Sarwar, G. Karypis, J.A. Konstan, and J. Riedl, "Item-Based Collaborative Filtering Recommendation Algorithms", In *Proc. of the 10th International World Wide Web Conference (WWW10)*, Hong Kong, May 2001, pp. 285-295.

[14] U. Shardanand, and P. Maes, "Social information Filtering: Algorithms for Automating 'Word of Mouth'", In *Proc. of the 1995 Conference on Human Factors in Computing Systems (CHI'95)*, 1995, pp. 210-217.

[15] Y.Y. Yao, "Measuring Retrieval Effectiveness Based on User Preference of Documents", *Journal of the American Society for Information Science*, Vol.46, No.2, February 1995, pp.133-145.