NAIST-IS-DD0361006

Doctoral Dissertation

A Framework for Software Function Recommendation based on Collaborative Filtering

Naoki Ohsugi

September 30, 2004.

Department of Information Systems
Graduate School of Information Science
Nara Institute of Science and Technology

Doctoral Dissertation

submitted to the Graduate School of Information Science,

Nara Institute of Science and Technology

in partial fulfillment of the requirements for the degree of

DOCTOR of ENGINEERING

Naoki Ohsugi

Dissertation committee:	Professor Ken-ichi Matsumoto (Supervisor)
	Professor Masaki Koyama (Co-supervisor)
	Professor Hiroyuki Seki (Co-supervisor)
	Associate Professor Hajimu Iida (Co-supervisor)

A Framework for Software Function Recommendation Based on Collaborative Filtering [*]

Naoki Ohsugi

Abstract

High-Functionality Applications (*HFAs*) contain a large number of functions. However, most HFA users use only a few functions and are not aware of other useful functions. To let users discover other useful, not previously known (or: previously unknown) functions efficiently, this dissertation proposes a framework for software function recommendation based on *Collaborative Filtering* (*CF*). The proposed framework includes an abstract design of a function recommender system and an automated process for producing a recommendation, as well as system implementation techniques and new CF algorithms. To produce a recommendation for a target HFA user, first, histories of software function executions (called *usage histories*) are collected from many HFA users via the Internet. Next, similarities among users are calculated using the frequencies of the function executions of each user. Then, the potential execution frequencies of the target user's previously unknown functions are predicted based on similar users' already known frequencies. Finally, a list of functions ranked by their potential frequency is given as a recommendation to the target user. Since this framework does not require a previously constructed "user model" to make a recommendation, it is easily applicable to many HFAs. Typically, the CF algorithm consists of a similarity computation algorithm and a prediction algorithm. This dissertation describes three simple prediction algorithms (lacking similarity computation), ten similarity computation algorithms including two

[*] Doctoral Dissertation, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT0361006, September 30, 2004.

new algorithms, and seven prediction algorithms. Prediction accuracies of these CF algorithms were empirically evaluated using usage histories collected from 23 Microsoft Office Application users. To evaluate the recommendation accuracy, *ARE* (*Average Relative Error*) and *NDPM* (*Normalized Distance-based Performance Measure*) were used. The results showed that the average NDPM and the average ARE of all the CF algorithms were better than that of randomly generated recommendations. In particular, *Rank Correlation*, which is one of the proposed similarity computation algorithms, outperformed the other nine algorithms. Also, *Weighted Sum*, one of the conventional prediction algorithms, outperformed the other six prediction algorithms.

Keywords: Recommender System, Filtering Algorithms, Usage History, HFA (High Functionality Application), and CSCW

List of Major Publications

Journal Paper

1. Naoki Ohsugi, Akito Monden, Shuuji Morisaki, Ken-ichi Matsumoto, “Software Function Recommender System Based on Collaborative Filtering,” *Journal of Information Processing Society of Japan*, Vol.45, No.1, pp.267-278, January 2004 (in Japanese).

Patent

1. Naoki Ohsugi, Ken-ichi Matsumoto, Akito Monden, Masateru Tsunoda, “Device, Program and Method for Estimating Missing Data,” TOKUGAN2003-377617, Submitted to Japan Patent Office, November, 2003.

Refereed Conference Papers

1. Naoki Ohsugi, Akito Monden, Ken-ichi Matsumoto, Shuuji Morisaki, “Collaborative Filtering for Recommendation System of Software Function,” *Proceedings of the Software Symposium 2002*, pp.15-24, July 16-19, 2002 (in Japanese).
2. Naoki Ohsugi, Akito Monden, Shuuji Morisaki, “Collaborative Filtering Approach for Software Function Discovery,” *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE2002)*, pp.29-30, October 3-4, 2002.
3. Naoki Ohsugi, Hajimu Iida, Masaki Koyama, “Software Engineering Aspects for the Online Multicast Go Game”, *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE 2002)*, pp.45-46, October 3-4, 2002.
4. Naoki Ohsugi, Akito Monden, Ken-ichi Matsumoto, “A Recommendation System for Software Function Discovery,” *Proceedings of the 9th Asia Pacific Software Engineering Conference (APSEC2002)*, pp.248-257, December 4-6, 2002.
5. Naoki Ohsugi, Masateru Tsunoda, Akito Monden, Ken-ichi Matsumoto, “Effort

Estimation Based on Collaborative Filtering," *Proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES2004)*, pp.274-286, April 5-8, 2004.

Contents

List of Figures

List of Tables

1. Introduction

Currently, application software provides a large number of functions to satisfy various users' demands and needs [12]. For instance, Figure 1 shows the number of software functions provided by Microsoft Office applications, which are popular *HFAs* (*High-Functionality Applications*). In this dissertation, the number of functions is considered to be equal to the number of menu items because each function is executed by clicking a corresponding menu item in these applications. In summary, Excel 2002 had the largest number of functions, 792. Power Point 2000 contained the smallest number of functions, 565, which is still large. On average, these applications had 690 different functions.

However, most users use only a few functions. Figure 1 also shows the number of functions actually used (executed) by 32 Microsoft Office users in 22 months. The bars

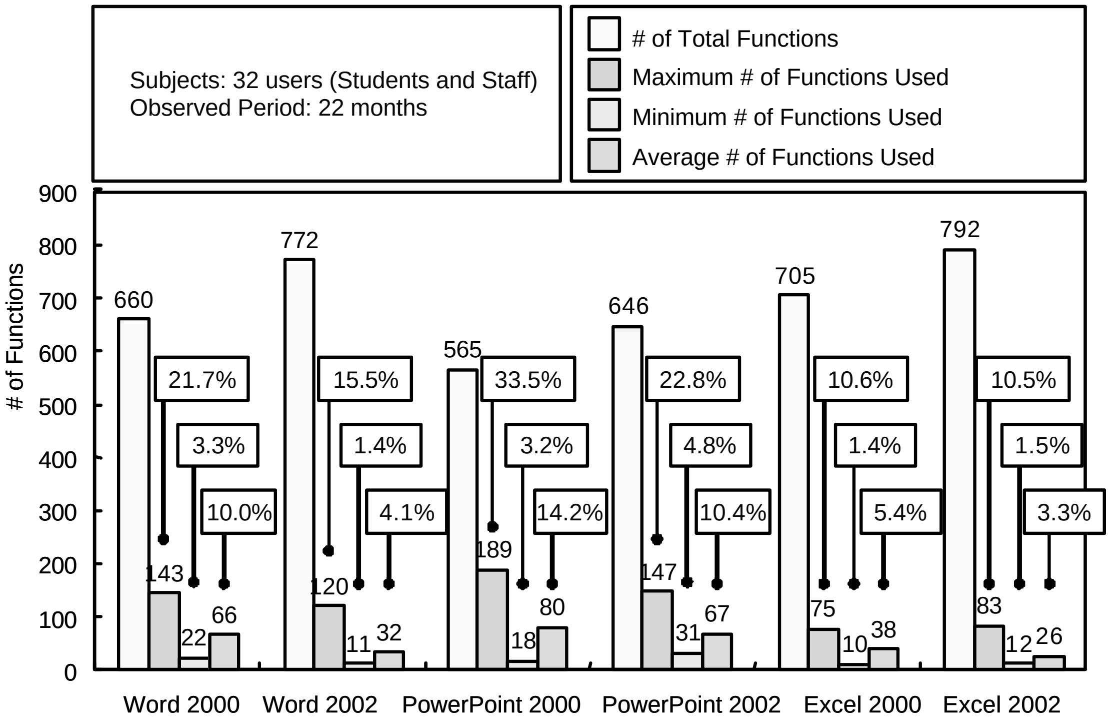


Figure 1. The Number of Functions Actually Used in High-Functionality Applications

in the figure show the maximum, minimum, and average number of functions actually used by these users. In summary, most users used approximately only 10% of all functions, i.e., over 90% of all functions remained unused. Morisaki et al., through an empirical study with 6 users of Microsoft Word 2000 and PowerPoint 2000 [28], [29], confirmed that many of these unused functions were indeed useful to users. Morisaki et al. let each user investigate a set of unused functions, which had not been executed by the user but had been executed by other users. As a result, users found 24.7 functions useful out of 63.7 investigations on average.

Nevertheless, although useful functions are involved in the set of unused functions, it is also pointed out that users do not spend the time and effort necessary to finding out these functions for the following reasons [57]:

- Users may not be aware of the existence of new functions.
- Users may not be motivated to learn if they think learning requires too much time and effort [5], [7].
- Users may not be able to find the new functions.
- Users may not be able to understand and apply the new functions.

In fact, in our preliminary analysis, some “diligent” users had continued discovering previously unknown functions, but others had stopped learning new functions. Figure 2 shows the relationship between the number of function executions and the number of executed functions of 22 Microsoft PowerPoint 2002 users. Typically, the number of function executions will increase as the user continues using an application. On the other hand, the number of executed functions will not increase unless the user continues discovering new functions. As shown in Figure 2, although the average number increased slowly, the minimum number almost did not. This suggests there were some users who were not motivated to seek previously unknown functions.

In order to promote the discovery of new functions, Morisaki et al. [28], [29], developed a system for sharing knowledge of useful, but previously unknown functions of a HFA in a small community whose members are doing similar tasks using the HFA. The system first collects records of the used functions (*usage history*) from all users in a community. Then, the system provides (shows) each user a set of unused functions, which have not been executed by the user, but have been executed by the other users. The users in a small community can share each other's knowledge by looking at other people's executed functions provided by this system. However, in the case of a large community whose members are doing various kinds of tasks by using the HFA, this system is not feasible because the number of each user's unused functions, which have been executed by other users, becomes too large.

For efficiently discovering useful functions of HFAs, we need a system that can predict and recommend useful, previously unknown functions for *individual* users. In such a system, the efficiency of function discovery may depend on the accuracy of predictions. To achieve higher accuracy, each user's individual needs as well as the similarity (in needs) between users have to be taken into account because obviously each

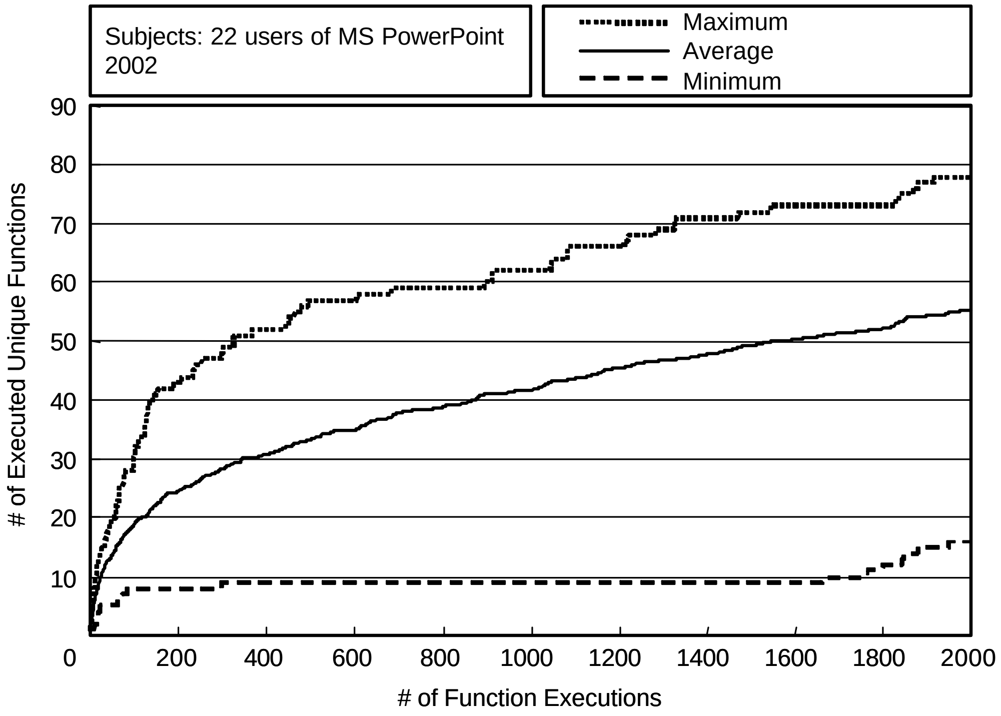


Figure 2. Relationship between the Number of Function Executions and the Number of Executed Functions

user has a different set of useful functions due to the distinctive tasks they select when using the HFA.

This dissertation proposes a framework for function recommendation based on a *Collaborative Filtering* (*CF*) that helps an individual user efficiently discover useful functions of HFAs. The CF was originated as a method of information filtering, which predicts individual users' preferences to filter out undesirable items. To date, the CF has been applied to implementing recommender systems as a prediction engine. Generally, the CF-based prediction consists of two steps: (1) first, the CF evaluates similarities between users from the preferences (e.g. a rating to an item on a 1 through 5 scale) given by the users themselves on already known (aware) items; (2) next, the CF predicts a target user u_a's preference of each unknown (unused) item i_j by using a similar users' preferences of the item i_j and their similarities to the user u_a, where the user u_b is a user who knows item i_j [4]. Since this process takes the user's individual needs into account, the CF can "individually" provide each user a prediction of preferred items.

The proposed framework includes an abstract design of a function recommender system and an automated process for making recommendations, as well as system implementation techniques and new CF algorithms suitable for function recommendation. Novel technology makes the proposed system differ from conventional CF-based recommender systems in that the proposed framework employs frequencies of function executions (*execution frequencies*) instead of the preferences (ratings) on the functions, in computing similarities and predictions. Since the execution frequency of HFA's functions can be automatically measured from HFA users' execution histories in this framework, users do not need any additional time or effort to "rate" the software functions they have used. By using CF algorithms, the usefulness of each function is individually predicted for each user. Based on the prediction, a list of functions ranked with their relative usefulness is provided to each user as a recommendation. Note that in this dissertation, a "useful" function means a function that has been "frequently used" by a user, or a function not previously known to a user, but will be "frequently used" when the user becomes aware of the functions.

This dissertation also describes two case studies for evaluating the accuracies of the predictions by applying the six algorithms including the *Rank Correlation Algorithm* and the *Magnitude Relation Algorithm*, which are newly proposed in this dissertation. Although researchers have developed many CF algorithms [1], [4], [41], [46], [48], whether or not these algorithms are suitable for recommending "software functions" is not clear since they do not consider the effect of "outlying" values. In software function recommendation, a CF algorithm has to avoid negative effects caused by outlying values in calculating similarities while the execution frequency of some HFA functions (such as "undo", "copy", "paste") can be considered "outliers" since they are used much more frequently than other functions. The proposed two algorithms are designed to avoid the negative effects of such outliers.

The remainder of this dissertation is structured as follows: Chapter 2 gives an overview of CF. Chapter 3 reviews conventional CF-based recommender systems. Chapter 4 explains the proposed framework including the design of a function recommender system and a process for producing a recommendation. Chapter 5 provides the details of the CF algorithms which can be employed by the proposed framework. Chapter 6 reports experiments for evaluating the CF algorithms described in Chapter 5. Chapter 7 discusses the results of the experiments. Chapter 8 compares the proposed method with related work. Finally, Chapter 9 concludes the dissertation and provides outlooks on future research activities.

2. Collaborative Filtering

Collaborative Filtering (CF) was originated by Goldberg et al. [14] for implementing “Tapestry”, which is a Usenet article filtering system. Tapestry stores Usenet articles in the database and analyzes whether or not each article received replies. Tapestry users can search “interesting” articles in a certain newsgroup even if they did not know how to write a search query expressing what is “interesting”. By using Tapestry, the users can retrieve “well-replied” articles, which might be more interesting than other articles. Tapestry is the most rudimentary style of CF because it uses the knowledge (Collaborative) of many users to find out preferable items (Filtering) although it does not employ any algorithmic procedures.

CF is one of the key techniques for implementing a recommender system which recommends to the user a set of candidate items which may be preferable or useful to individual users [19]. Typically, the recommended items are selected from a large number of items such as Usenet articles, Web pages, books, or movies. CF predicts the preferences (ratings) of previously unknown items for a particular user, called an *active user*. Thus, the CF-based recommender system can recommend a set of candidate items with a high possibility of being preferred by the active user.

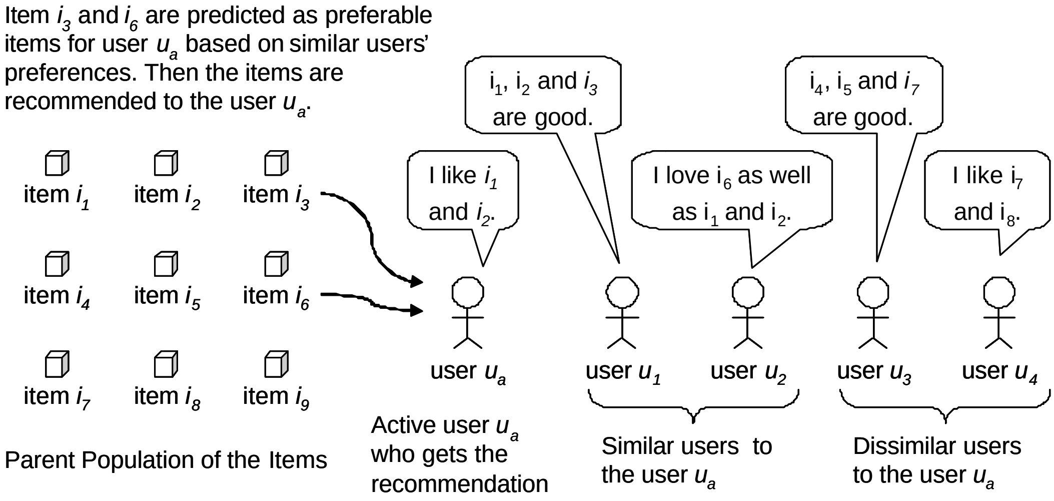

Item i_3 and i_6 are predicted as preferable items for user u_a based on similar users' preferences. Then the items are recommended to the user u_a.
item i_1
item i_2
item i_3
item i_4
item i_5
item i_6
item i_7
item i_8
item i_9
Parent Population of the Items
i_1, i_2 and i_3 are good.
I like i_1 and i_2.
I love i_6 as well as i_1 and i_2.
i_4, i_5 and i_7 are good.
I like i_7 and i_8.
user u_a
user u_1
user u_2
user u_3
user u_4
Active user u_a who gets the recommendation
Similar users to the user u_a
Dissimilar users to the user u_a

Figure 3. Overview of the Recommendation with Collaborative Filtering

The main feature of the CF is to employ other similar users' preferences to predict the active user's preferences [46]. Figure 3 shows the overview of the recommendation with CF. As described in the figure, an environment applying the CF consists of some items and some users. All users express their preferences about the items they have used. First, the CF finds a set of users who are similar to the active user. In Figure 3, the CF has evaluated user u_1 and u_2 as similar users because they have expressed similar preferences with user u_a who is the active user. Then, the CF predicts preferable items based on similar users' preferences. In Figure 3, the CF predicted that items i_3 and i_6

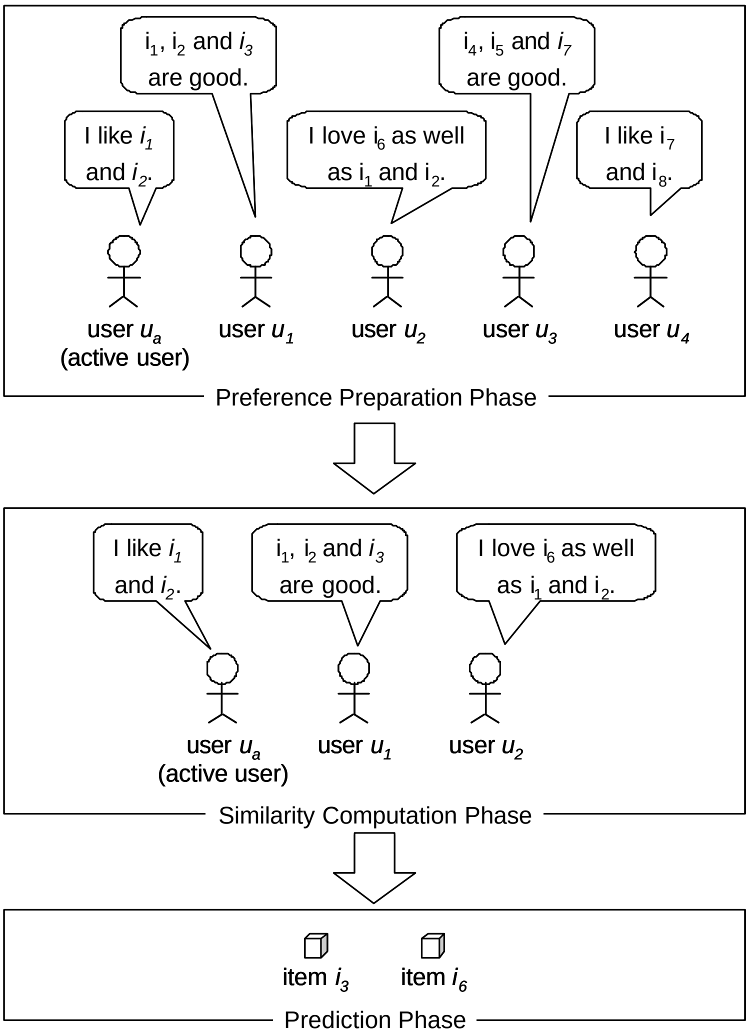

i_1, i_2 and i_3 are good.
i_4, i_5 and i_7 are good.
I like i_1 and i_2.
I love i_6 as well as i_1 and i_2.
I like i_7 and i_8.
user u_a (active user)
user u_1
user u_2
user u_3
user u_4
Preference Preparation Phase
I like i_1 and i_2.
i_1, i_2 and i_3 are good.
I love i_6 as well as i_1 and i_2.
user u_a (active user)
user u_1
user u_2
Similarity Computation Phase
item i_3
item i_6
Prediction Phase

Figure 4. Systematic Process of Collaborative Filtering

were preferable because users u_1 and u_2 have preferred them as well as items i_1 and i_2, which were preferred by user u_a.

As described in Figure 4, the CF is systematically conducted through three phases: the *preference preparation phase*, the *similarity computation phase* and the *prediction phase*. First, in the preference preparation phase, a CF conductor (i.e., a system which conducts CF) prepares the users' preferences. Each user's preferences are data expressing which items are preferred and how they are preferred. Typical CF systems prepare users' preferences by constructing a particular data object extracted from a database. Next, in the similarity computation phase, the CF conductor evaluates the similarities between the active user and the other users. Finally, in the prediction phase, the conductor predicts the active user's preferences of previously unknown items, using the similarities computed in the previous process. Generally, these three phases are implemented separately. Combining the processes at the similarity computation phase and the prediction phase is an important factor affecting the system's performance, for example, in terms of the accuracy of the prediction and the processing speed.

In the remainder of this dissertation, the systematic processes performed in the similarity computation phase and the prediction phase are called a *similarity computation algorithm* and a *prediction algorithm*, respectively. Likewise, a combination of a similarity computation algorithm and a prediction algorithm is simply called a *CF algorithm*. There are many CF algorithms which have been proposed in the conventional literature (for example, [4], [13], [41], [45], [46], [48] etc.); however, there is no perfect CF algorithm which achieves the best performance all the time because the system's performance is also affected by other factors such as characteristics of data of users' preferences. Thus, in developing a CF system, experimental evaluations of the performances are very important. The system developers should use data actually employed in the system for these evaluations. Then, based on the evaluations, the developers should choose the most suitable CF algorithm. Chapter 5 describes each CF algorithm precisely. Furthermore, Chapter 6 reports on the experimental evaluation of these CF algorithms.

3. CF-based Recommender Systems

3.1 Classification Categories

Figure 5 shows an input and output of a conventional CF-based recommender system. The system collects users' preferences from potential recommenders and the active user who gets recommendation. Next, the system generates recommendations to the active user by using CF. CF-based recommender systems can be classified into several categories based on their characteristics related to input and output. This chapter introduces some typical conventional systems and describes how these are classified. The classification categories are as follows:

- *Type of Items*: "What does the system recommend for users?" The answer to this question is a class of this classification category. Currently, many CF systems recommend various types of items.
- *Type of Recommenders*: "Who are the actual recommenders in the system?" The

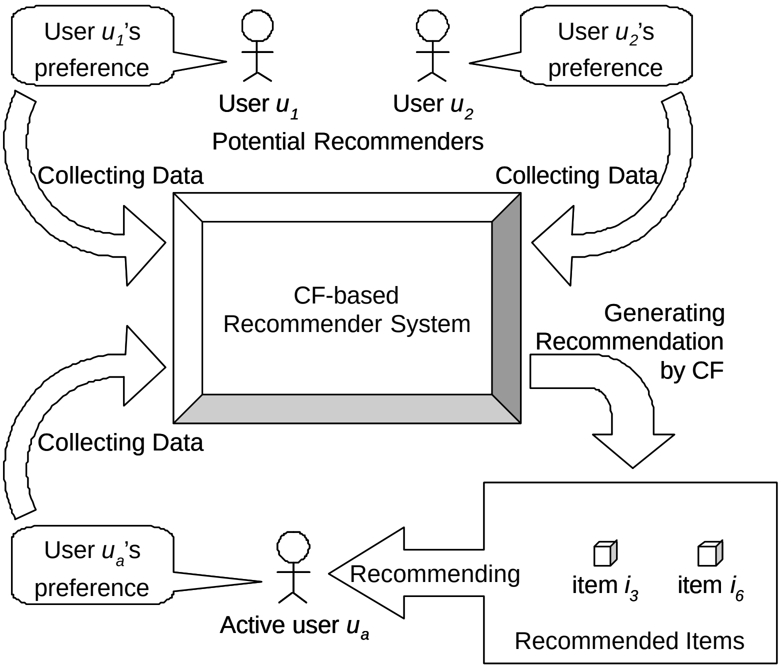


Figure 5. Input and Output of a Conventional CF-based Recommender System

answer to this question is a class of this classification category. Although the CF system itself is an immediate recommender, the recommended items given to an active user are computed based on other users' preferences. In this sense, some users are indirect recommenders. In this category, CF systems are classified according to the type of such indirect recommenders.

- *Type of Data Collection*: "Do the users have to enter their preference into the system explicitly?" The answer to this question is a class of this classification category. There are two types of CF systems in data collection. If the users of the system have to enter their preferences explicitly into the system, then, the system is classified as an *Explicit Data Collection*. Otherwise, the system is classified as an *Implicit Data Collection.* The latter system collects the users' preferences implicitly using sophisticated methods (e.g. examining the users' purchase records).
- *Type of Data Values*: "What kind of values are employed for expressing the users' preferences?" The answer to this question is a class of this classification category. The users' preferences are expressed with different types of values, depending on the implementation of the CF system. For example, some systems employ binary values: 1 (prefer) or 0 (do not prefer). However, other systems employ the discrete values of a five grade scale: 5 (prefer) to 1 (do not prefer).

3.2 Conventional Recommender Systems

Table 1 shows the classification of conventional recommender systems based on the categories described in Section 3.1. In the Table, columns indicate the categories; and, each row indicates a classification result of each system.

Tapestry proposed by Goldberg et al. [14] is the origin of the CF-based recommender systems. Tapestry is classified in the "Usenet articles" class in the "Type of Items" category in Table 1, because it is a Usenet article filtering system as described in Chapter 2. In the "Type of Recommenders" category, Tapestry is classified in the

Table 1. Classification of Conventional CF-Based Recommender Systems

	Type of Items	Type of Recommenders	Type of Data Collection	Type of Data Values
Tapestry	Usenet articles	System outsiders	Implicit	Binary values: Replied (1) or Not (0)
GroupLens	Usenet articles	Similar users	Explicit	Discrete values: 5 (Good) to 1 (Bad)
Siteseer	URLs	Similar users	Implicit	Binary values: Bookmarked(1) or Not(0)
Fab	URLs	Similar users	Explicit	Discrete values: 7 (Excellent) to 1 (Terrible)
PHOAKS	URLs	System outsiders	Implicit	Binary values: Mentioned (1) or Not (0)
Beehive	Text messages	Topic group participants	Explicit	Text Messages
Ringo	Pieces of music	Similar users	Explicit	Discrete values: 7 (BOOM!) to 1 (Pass the earplugs)
MovieLens	Movies	Similar users	Explicit	Discrete values: Must See(5) to Awful(1)
ReferralWeb	People	System outsiders	Implicit	Online Text Resources
PTVPlus	TV Program	Similar users	Explicit	User Profile and Binary Values: Positive (1) or Negative (0)

"System outsiders" class because Usenet users who replied to articles are on the outside of the system. A Tapestry user can retrieve Usenet articles that have been replied to by someone else. If retrieved articles are preferable for a Tapestry user, the repliers to the articles indirectly recommended them to the users. In the "Type of Data Collection" category, Tapestry is classified in the "Implicit" class because the users do not need to enter any preferences into the system. Instead, Tapestry analyzes whether or not each article got replies for tracking interesting articles. In the "Type of Data Values" category, Tapestry is classified into the “Binary Values” class because it records an analyzed result of each article with a binary value: 1 (replied article) and 0 (not replied).

GroupLens proposed by Resnick et al. [41] is also a Usenet article recommender system. GroupLens originated a basic form of the CF algorithm to help users to focus on interesting items. It draws on a simple idea; people who agreed in their subjective evaluation of past articles are likely to agree again in the future. After reading articles, users explicitly assign numeric ratings to the articles. GroupLens uses the ratings in two ways. First, it correlates the ratings in order to determine which users’ ratings are most similar to each other. Second, it predicts how well users will like new articles, based on the ratings from similar users. As described in Table 1, it is clear that GroupLens is classified in the “Usenet articles” class in the “Type of Items” category; “Similar users” class in the “Type of Recommenders” category; and “Explicit” class in the “Type of Data Collection” category. In the “Type of Data Values” category, GroupLens is classified as “Discrete Values” because the numeric rating assigned to each article is a discrete value of the five grade scale: 5 (it is a good article) to 1 (it is a bad one).

Siteseer proposed by Rucker and Marcos [43] is a URL recommender system. Siteseer uses the findings of one user as implicit recommendations for other users, based on the URLs bookmarked by the users qualified as trusted recommenders. Siteseer looks at each user’s bookmarks and measures the degree of overlap (i.e., common URLs) of each user’s bookmark with the other user’s bookmarks. Then, Siteseer determines the similarities between the users by using an overlap of the URLs (not: titles and the contents of Web pages). Siteseer provides as recommendations those pages which have been bookmarked by the user’s virtual neighbors, giving preference to pages drawn from multiple bookmarks in the neighborhood. Siteseer contextualizes its recommendations by delivering them to each user’s bookmarks. As described in Table 1, it is clear that Siteseer is classified in the “URLs” class in the “Type of Items” category; and in the “Similar users” class in the “Type of Recommenders” category. In the “Type of Data Collection” category, Siteseer is classified as “Implicit” because users’ preferences are automatically examined from bookmarks. In the “Type of Data Values” category, Siteseer is classified in the “Binary Values” class because it analyzes preferences on each URL as a binary value: 1 (it is bookmarked) and 0 (it is not bookmarked).

Fab proposed by Balabanovic and Shoham [1] is a URL recommender system. Fab combines a content-based approach and a CF-based approach to make recommendations. As a content-based approach, Fab recommends URLs based on comparisons between user profiles and text-contents of the Web pages specified with the URLs. When a Web page is shown to a user, the user can put feedback into the system to improve the user profile. If the user liked the Web page, weights for the words extracted from it can be added to track the user’s interests. In contrast to a content-based approach, Fab recommends URLs that have been preferred by other similar users. The users’ feedback, which is used for improving the users’ profiles in the content-based approach, is also used for evaluating similarities between the users. Scores for URLs related to the unread Web pages are predicted based on a combination of the scores known from some similar users. As described above, it is clear that Fab is classified in the “URLs” class in the “Type of Items” category; “Similar users” class in the “Type of Recommenders” category; and “Explicit” class in the “Type of Data Collection” category. In the “Type of Data Values” category, Fab is classified as “Discrete values” because the users can put feedback to Fab as a discrete value of a seven grade scale: 7 (it is an excellent page) and 1 (it is a terrible one).

PHOAKS proposed by Terveen et al. [52] is a URL recommender system. PHOAKS uses Usenet articles as recommenders of the URLs. Some Usenet users often post their impressions and opinions about all sorts of items, including Web pages. They may state what a page is useful for and how useful it is. PHOAKS searches messages for mentions of Web pages (URLs) and counts a mention as a recommendation. The URLs are recommended according to the frequency of the mentions. As described above, PHOAKS is classified in the “URLs” class in the “Type of Items” category. In the “Type of Recommenders” category, PHOAKS is classified in the “System outsiders” class because all Usenet users can be potential recommenders. In the “Type of Recommenders” category, PHOAKS is classified in the “Implicit” class because it counts previously mentioned URLs on the Web. PHOAKS users do not need to enter any preferences into the system. In the “Type of Data Values” category, PHOAKS is classified in the “Binary Values” class because it counts each URL as a binary value: 1 (it is mentioned in an article) and 0 (it is not mentioned).

Beehive proposed by Huberman and Kaminsky [23] is a text message recommender system. Beehive users join in various topic groups to receive recommended text messages. A text message posted to a group is delivered to the participants of the group. A user, who receives the recommended text messages, must also recommend text messages to the 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. As described above, Beehive is classified in the “Text messages” class in the “Type of Items” category. In the “Type of Recommenders” category, Beehive is classified in the “Topic group participants” class. The topic group is managed manually by the operator. In the “Type of Recommenders” category, Beehive is classified in the “Explicit” class because Beehive users have to explicitly post a text message to continue to receive the recommendations. In the “Type of Data Values” category, Beehive is classified in the “Text messages” class because it does not use any numeric values for making recommendations. Beehive only delivers the posted text messages to the group participants.

3.3 Advantages and Limitations of Conventional Systems

All conventional recommender systems have both advantages and limitations. These can be summarized and explained according to their classification results. Each class also has its own advantages and limitations, which are commonly seen in recommender systems belonging to each particular class.

The classes in the “Type of Items” category strongly reflect the ability of the system because the system cannot recommend any other types of items. For instance, a TV program recommender system cannot recommend any Usenet articles. This is not only because the implementation has been specified to a certain type of items, but also because the CF procedure strongly depends on the characteristics of the type of items. For instance, it is difficult to modify Fab, which is a URL recommender system, so as to become a movie recommender system. Fab provides a hybrid recommendation based on a content-based approach and a CF-based approach; however, Fab cannot conduct a content-based approach for movies because the contents of movies are not text data. It is important for a developer of a recommender system to design a CF procedure after carefully examining the characteristics of the particular type of recommended items.

The classes in “Type of Recommenders” category were either “Similar users” or “System outsiders” as in existing recommender systems. Here, “Similar users” class indicates that recommenders are a part of the system subscribers (i.e. not system outsiders) selected according to their similarities to an active user. This class has the advantage that the system can provide accurate recommendations (i.e. is desirable for the users) since the system evaluates similarities between users by a reasonable method. However, the disadvantage is that the system may produce poor recommendations if the number of users is much smaller than the number of items. In such a case, the system cannot find similar users among the system subscribers because there are few items rated by more than two users. MovieLens [45] challenged this problem in that the number of the users is much smaller than the number of the movies. Few movies were co-rated by more than two users because most users rated at most 50 movies although millions of movies exist. In such a case, the item-based CF, which computes the similarity between items instead of users, is suitable for making recommendations. However, the item-based CF is outside the scope of this dissertation. On the other hand, for the “System outsiders” class, recommenders are not subscribers of the system. The system extracts their knowledge from public information such as Usenet news or Web resources. The system simply reuses their knowledge to make recommendations. This class has the advantage, nonetheless, of allowing the system to provide recommendations even if there are few system subscribers. In addition, the recommendation accuracy does not depend on the number of users and items. However, the disadvantage to this system is that the accuracy of the recommendation is liable to be lower than in the systems that follow the “similar users” class because the system cannot make a “personalized” recommendation to each user.

The class of “Type of Data Collection” must be either “Explicit” or “Implicit”. A system in the “Explicit” class explicitly collects data, which are used for working recommendations. In other words, each user manually rates the items the user already used. This class has the advantage that the data contain completely accurate preferences of the users because these preferences are entered by the users themselves. However, the disadvantage is that the users have to spend time and effort to rate items. In addition, even if the users do not mind spending time and effort to manually rate items, the users sometimes cannot express their preferences (ratings) explicitly. For instance, ReferralWeb [24], which is a recommender system for human relations, implicitly collects data from online Web resources because some users do not recognize their own human relations clearly. On the other hand, a system in the “Implicit” class implicitly collects data used for making recommendations. In other words, the system analyzes users’ preferences automatically from data sources such as users’ purchase records. This class has the advantage that the users do not need to spend the time and effort needed to rate items manually. However, the disadvantage is that collected data might be different from the users’ real preferences.

The class of “Type of Data Values” typically includes either “Binary values” or “Discrete quantities”. A system in the “Binary values” class employs binary values (i.e. 1 and 0) to express the users’ preferences. This class has the advantage that it is easy for a user to decide a value to be entered. If the system employs explicit data collection, users may enter 1 for the items, which represents a positive feeling, otherwise, 0. The user does not have to consider a grade of preference for the items. The disadvantage of this class is that since the binary values have less information than the “Discrete quantities” described below, these values might lower the accuracy of the recommendations. On the other hand, a system in the “Discrete quantities” class employs discrete quantities for expressing users’ preferences. Typically, integer values are used, and their range is limited (e.g. 5 (good) to 1 (bad)). This class has the advantage that the system can track the grades of the users’ preferences. Due to this, the system is liable to make more accurate recommendations than the systems employing “Binary values”. The disadvantage is that deciding grades for their preferences might be difficult for users. In addition, users might be annoyed by the limitation of the range of values if the user wants to rate an item higher than the highest score ever rated. For instance, in using MovieLens [45], users cannot enter a score higher than the best score 5.

4. Framework for Software Function Recommendation

4.1 Design Concept

Figure 6 shows the input and output of a software function recommender system in the proposed framework. Obviously, the system belongs to the “Software functions” class in the “Type of Items” category. It is desirable that the type of recommender represent the “Similar users” (i.e. not the “System outsiders”) because accurate recommendations are especially needed for software function discovery. Horvitz et al. [22] argued that poor recommendation could be quite costly to users. In their experiment, even when a recommendation was off the mark, the users would often become distracted by the recommendation and begin to use the recommended functions. This would give the system false confirmation of successful recommendations, and would bolster the system to continue giving recommendations, thus pushing the user down a distracting path. Such patterns of poor guesses and “confirmatory” feedback could lead to focusing

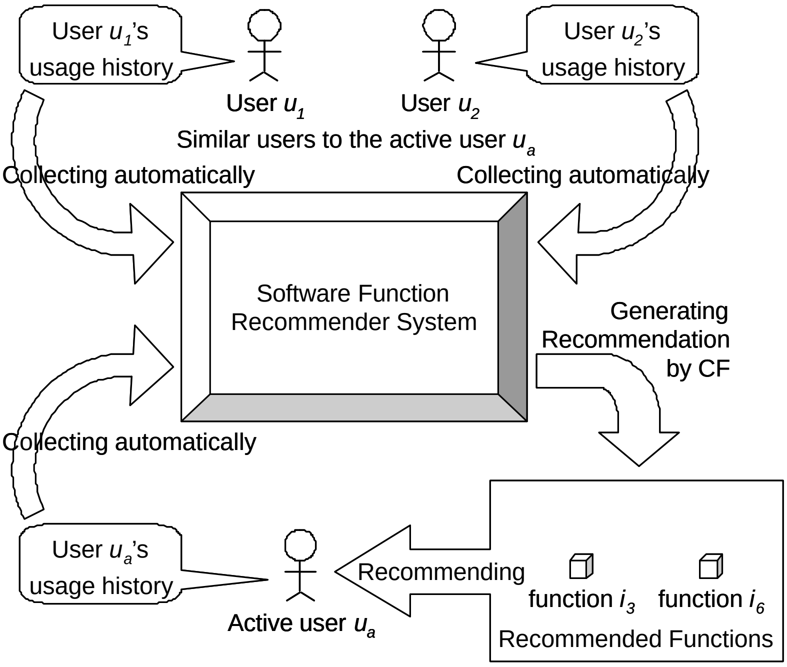

User u_1's usage history
User u_1
User u_2
User u_2's usage history
Similar users to the active user u_a
Collecting automatically
Collecting automatically
Software Function Recommender System
Generating Recommendation by CF
Collecting automatically
User u_a's usage history
Active user u_a
Recommending
function i_3
function i_6
Recommended Functions

Figure 6. Input and Output of a Software Function Recommender System

on the wrong problem and a loss in efficiency. A system of "Similar users" can provide an accurate recommendation if the system employs a reasonable similarity computation method. Also, in the context of the function recommendation, the limitation of the "Similar users" class (i.e. the system cannot find similar users if the number of users is much smaller than the number of items) might not affect the system because the number of items (i.e. software functions) is at most 800 (see, Figure 1). It is easier for a software function recommender system to find similar users than for the conventional recommender systems, such as movie recommender systems, to do so because the number of items is not large in a software recommendation system.

Also, it is desirable that "Type of Data Collection" is "Implicit" because users do not want to spend time and effort entering the ratings of executed functions while they are working. Conveniently, frequently used functions can be automatically detected by collecting records of used functions (usage histories) [17]. In fact, some HFAs such as Microsoft Office applications themselves observe usage of the software. In laying out menus and toolbars, these applications relocate frequently used functions into places which can be easily recognized and accessed. In our framework, a recommender system automatically collects usage histories, and observes the frequency of executions (execution frequency) of each function.

Obviously, the "Type of Data Values" represents "Continuous quantity" because the execution frequency of a software function is a continuous quantity. To date, no system belongs to this class. In the proposed framework the execution frequency of function f_j is defined as:

$$\mathit{Execution\ Frequency\ of}\ \ f_j = \frac{\mathit{Number\ of\ \ Executions\ of}\ \ f_j}{\mathit{Total\ Number\ of\ \ Executions\ by\ a\ User}} \qquad (1)$$

The value defined by (1) takes a range of 0.0 to 1.0. A larger value denotes that the function f_j is more frequently used. Continuous quantity can track levels of users' preferences in detail. Thus, a system in this class has the potential to make more accurate recommendations than the systems employing "Binary values" or "Discrete quantities".

In addition, in the proposed framework, execution frequency is relativized by comparison with the other functions' execution frequencies.

4.2 Architectural Design

The proposed framework defines the basic architecture of the system described in the previous section. Figure 7 shows basic architecture of a software function recommender system. As shown in the figure, the system consists of a client software installed in each user's computer, and a function recommender server. Each user's computer has a HFA, which is a target of function recommendation. Users' computers are connected to the server via a network such as the Internet. Each user's computer contains two software components: the *Usage History Collector* and the *Recommendation Receiver*. The Usage History Collector is a software component that observes used functions in using HFA, and records the functions as usage history. The Recommendation Receiver is a software component that receives recommendations sent from the server, and shows them to the user. These components are installed into all

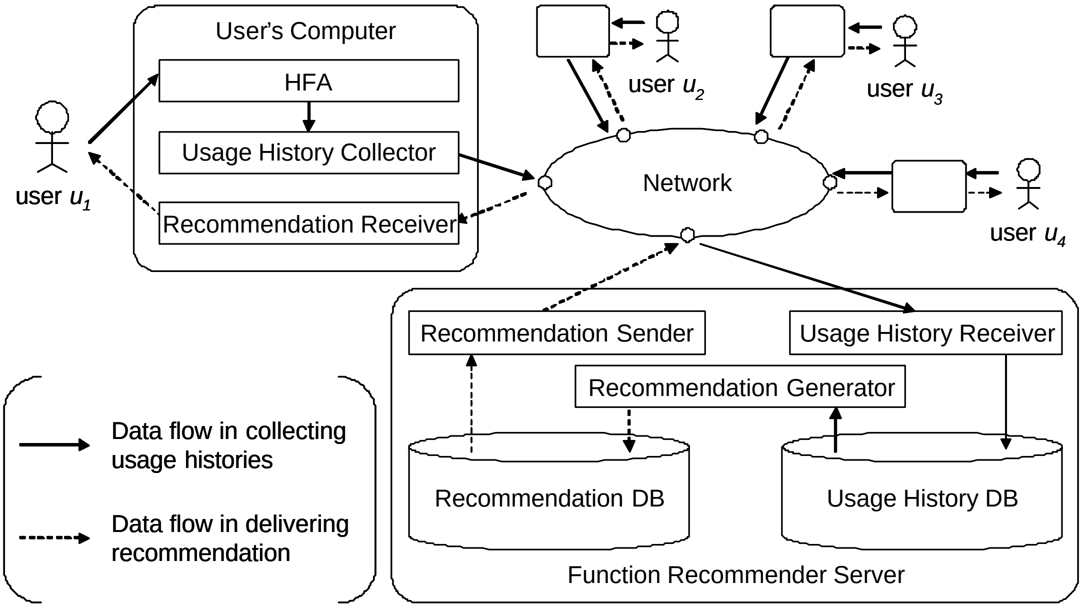


Figure 7. Architecture of Software Function Recommender System

computers whose users want to get recommendations through the system.

The function recommender server contains three software components and two databases: the *Usage History Receiver*, the *Recommendation Generator*, the *Recommendation Sender*, the *Usage History DB*, and the *Recommendation DB*. The Usage History Receiver is a software component that receives usage histories sent from the users' computers and stores them to the Usage History DB. The Recommendation Generator is a software component that produces recommendations (a set of candidate functions given priorities) for individual users by using CF, and stores them to the Recommendation DB. The Recommendation Sender is a software component that sends each recommendation to each user when the user requests the recommendation to be sent. The Usage History DB and the Recommendation DB store the usage histories collected from the users and the recommendations produced for the individual users, respectively. The system needs at least one server computer. If the system needs higher scalability, the system must be built on two or more server computers. One reasonable implementation of the multi-server system is to deploy each server component into each server computer

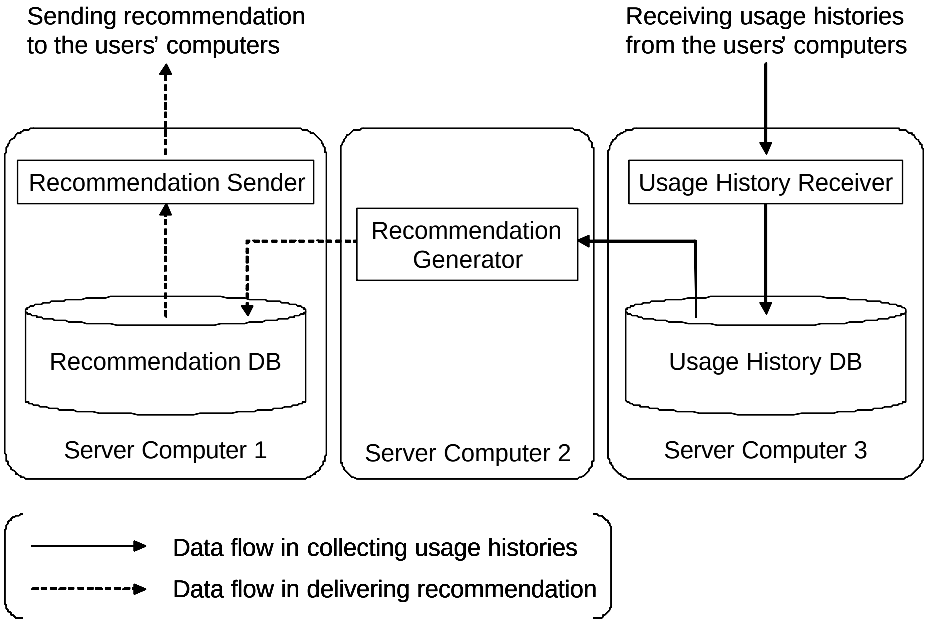

Sending recommendation to the users' computers
Receiving usage histories from the users' computers
Recommendation Sender
Recommendation Generator
Usage History Receiver
Recommendation DB
Usage History DB
Server Computer 1
Server Computer 2
Server Computer 3
Data flow in collecting usage histories
Data flow in delivering recommendation

Figure 8. Example of Server Components Deployment on the Multi-Server System

separately. Figure 8 shows an example of the server components deployment on the multi-server system. As described in the figure, the Usage History DB is deployed into the computer with the Usage History Receiver. The Recommendation DB is deployed into the computer with the Recommendation Sender. Due to this, these server computers do not have to synchronize their databases with each other. In addition, they can share the process load for producing recommendations and sending them.

4.3 Recommendation Process

The proposed framework defines the recommendation process of the system architecture described in the previous section. Figure 9 shows the entire process of the recommendation. This process consists of the following five completely automated steps.

Step 1: Collecting usage histories
Step 2: Preparing usage matrix
Step 3: Applying CF algorithm
Step 4: Sorting recommended functions
Step 5: Delivering recommendation

The remainder of this section describes how each step is processed by the software components shown in Figure 7.

Step 1: **Collecting usage histories**

The Usage History Collector on each user's computer and the Usage History Receiver on the server collect usage history through the network. The data flows in collecting usage histories are drawn as the solid lines in Figure 7. Each usage history is collected as follows:

1. The Usage History Collector inspects executed functions of the HFA on each user's computer, and temporally records them in the client's hard disk as a usage history.

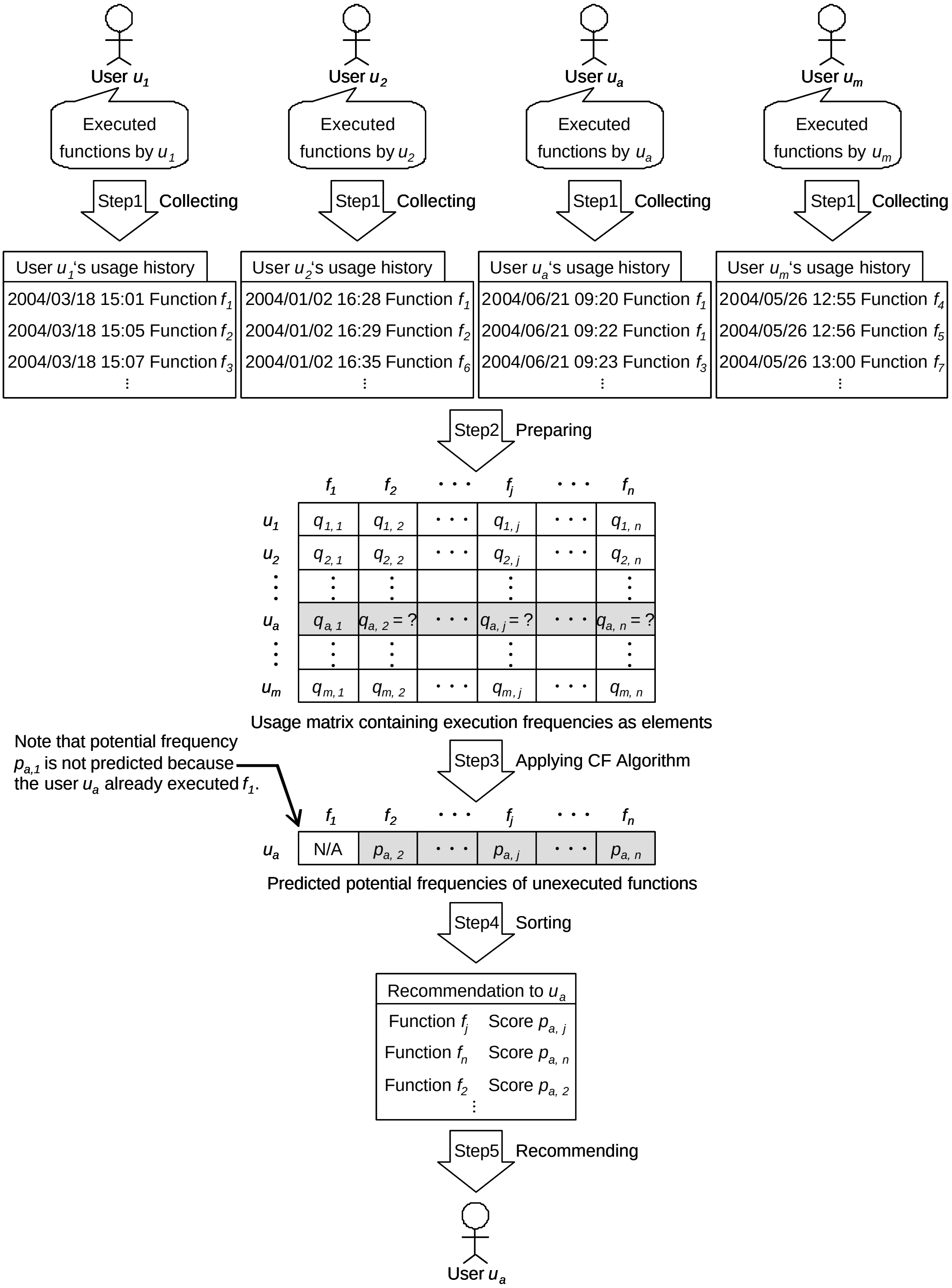

User u_1
User u_2
User u_a
User u_m
Executed functions by u_1
Executed functions by u_2
Executed functions by u_a
Executed functions by u_m
Step1 Collecting
User u_1's usage history
2004/03/18 15:01 Function f_1
2004/03/18 15:05 Function f_2
2004/03/18 15:07 Function f_3
User u_2's usage history
2004/01/02 16:28 Function f_1
2004/01/02 16:29 Function f_2
2004/01/02 16:35 Function f_6
User u_a's usage history
2004/06/21 09:20 Function f_1
2004/06/21 09:22 Function f_1
2004/06/21 09:23 Function f_3
User u_m's usage history
2004/05/26 12:55 Function f_4
2004/05/26 12:56 Function f_5
2004/05/26 13:00 Function f_7
Step2 Preparing
f_1 f_2 ··· f_j ··· f_n
u_1 $q_{1,1}$ $q_{1,2}$ ··· $q_{1,j}$ ··· $q_{1,n}$
u_2 $q_{2,1}$ $q_{2,2}$ ··· $q_{2,j}$ ··· $q_{2,n}$
u_a $q_{a,1}$ $q_{a,2} = ?$ ··· $q_{a,j} = ?$ ··· $q_{a,n} = ?$
u_m $q_{m,1}$ $q_{m,2}$ ··· $q_{m,j}$ ··· $q_{m,n}$
Usage matrix containing execution frequencies as elements
Note that potential frequency $p_{a,1}$ is not predicted because the user u_a already executed f_1.
Step3 Applying CF Algorithm
f_1 f_2 ··· f_j ··· f_n
u_a N/A $p_{a,2}$ ··· $p_{a,j}$ ··· $p_{a,n}$
Predicted potential frequencies of unexecuted functions
Step4 Sorting
Recommendation to u_a
Function f_j Score $p_{a,j}$
Function f_n Score $p_{a,n}$
Function f_2 Score $p_{a,2}$
Step5 Recommending
User u_a

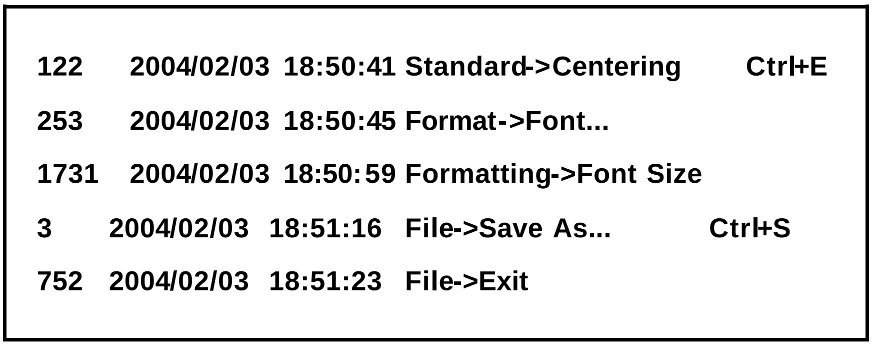
Figure 9. Recommendation Process of a Software Function Recommender System

2. The Usage History Collector sends the temporally recorded usage history to the server when the user shuts down the HFA.
3. The Usage History Receiver on the server receives the sent usage history from the user's computer, and stores it into the Usage History DB.

Figure 10 shows an instance of a usage history collected from the Microsoft Office XP applications (Word2002, Excel2002 or PowerPoint2002). In this instance, each row corresponds to each function execution. Each row is made from the following elements: the identification number of the executed function, the time and date of the execution and the function's label on the software. These elements are arranged as a row, from left to right, respectively. In addition, the pushed shortcut keys are put as the last element if the user used the shortcut keys to access the function. For example, in Figure 10, the first row shows the user executed the function 122 labeled "Standard->Centering" at 6:50:41 PM February 3, 2004, by pushing the shortcut keys Ctrl+E.

Step 2: **Preparing usage matrix**

The Recommendation Generator on the server prepares a *usage matrix* containing the execution frequencies as its elements. In Figure 9, suppose the system has been used by a set of users $U = \{u_1, u_2, ..., u_m\}$ and the system is making a recommendation for an active user u_a who is included in U. Each user u_i has used each function f_j included in the set of total functions $F = \{f_1, f_2, ..., f_n\}$. By using these mathematical symbols, The Usage



```
122    2004/02/03  18:50:41 Standard->Centering      Ctrl+E
253    2004/02/03  18:50:45 Format->Font...
1731   2004/02/03  18:50:59 Formatting->Font Size
3     2004/02/03  18:51:16  File->Save As...        Ctrl+S
752   2004/02/03  18:51:23  File->Exit
```

Figure 10. Example of Usage History

History Receiver calculates each execution frequency $q_{i,\,j}$ of a function f_j contained in each user's usage history, by the following equation:

$$q_{i,j} = \frac{e_{i,j}}{\sum_{f_k \in F_i} e_{i,k}} \qquad (2)$$

where, $e_{i,j}$ denotes the number of the function f_j executions by the user u_i, and F_i denotes the set of executed functions by user u_i. Note that the Usage History Receiver does not calculate the execution frequencies of the unexecuted functions. For instance, if the active user u_a has not executed function f_j, the execution frequency $q_{a,\,j}$ must be an empty element in the usage matrix rather than 0.0.

***Step 3:* Applying CF algorithm**

The Recommendation Generator on the server applies a CF algorithm to the usage matrix in order to predict each potential execution frequency the active user u_a will execute for the function f_j after the function is recommended. Note that the Usage History Receiver does not calculate the execution frequencies of the already executed functions. For instance, if the active user u_a has executed function f_1, the potential execution frequency $p_{a,1}$ is not predicted. The user perhaps knows the already executed functions. Because the user may not want to get recommendations which include the functions the user already knows, these functions will not be included in the recommendation. In order to reduce processing time, the Recommendation Generator ignores these already executed functions in applying the CF algorithm. The concrete equations of the CF algorithms are described in the next chapter.

***Step 4:* Sorting recommended functions**

The Recommendation Generator sorts the unexecuted functions according to the predicted potential frequencies, in descending order. Next, the sorted functions and the predicted potential frequencies are stored into the Recommendation DB as a recommendation. Figure 11 shows an instance of a recommendation for Microsoft Office XP applications. In this instance, each row corresponds to each unexecuted function.

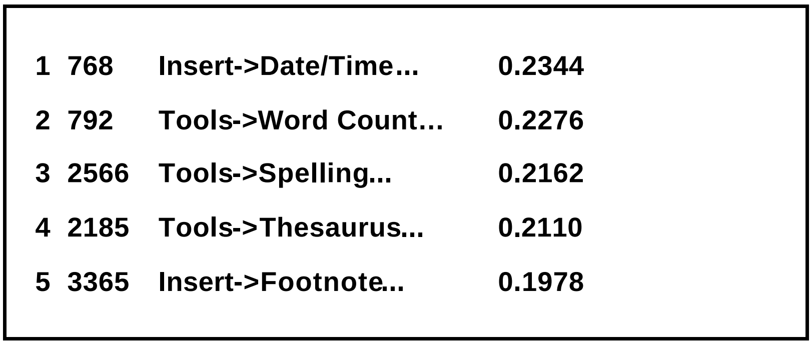
Each row is made from the following elements: the order of recommendation, the identification number of the function, the function's label and the predicted potential execution frequency. These elements are arranged as a row, from left to right. For example, in Figure 11, the first row shows that the most recommendable function has the identification number 768. The function is labeled as “Insert->Date/Time...”. The potential execution frequency of the function 768 is predicted to be 0.2344 (23.44%).

Typically, Steps 2, 3, and 4 are processed to produce recommendations for all users individually. These processes might spend the largest processing resources in the system. One reasonable implementation is that the Recommendation Generator updates recommendations in its spare time periodically. This implementation can improve response time to the users’ requests because producing recommendations requires some time if the number of the users is very large.

***Step 5:* Delivering recommendations**

The Recommendation Sender on the server and the Recommendation Receiver on each user’s computer delivers the recommendations to each user. The data flows in delivering recommendations are drawn as the broken lines in the Figure 7. Technically, each recommendation is delivered as follows:

1. The Recommendation Sender on the server sends the recommendation stored in Recommendation DB to each user when the user requests the recommendation.
2. The Recommendation Receiver on each client receives the sent recommendation from the server.



```
1  768    Insert->Date/Time...     0.2344
2  792    Tools->Word Count...     0.2276
3  2566   Tools->Spelling...       0.2162
4  2185   Tools->Thesaurus...      0.2110
5  3365   Insert->Footnote...      0.1978
```

Figure 11. Example of a Recommendation

3. The Recommendation Receiver displays the recommendation to the user with suitable format and timing for support, while not disturbing the user's work.

Note that the system should use a suitable timing when displaying the recommendations in order not to disturb the users' primary tasks [27]. Generally, users work to complete their primary tasks such as writing documents and making spreadsheets. The effectiveness of the system is reduced if it disturbs the users by displaying recommendations, even if it can make helpful recommendations. Thus, the system should take into account a suitable timing when displaying recommendations. For example, a suitable timing exists when the users have no interaction with their computers. In this case, the system can display the recommendations through various protocols such as the e-mail systems and the Web pages.

The system can adjust the visual format by converting the predicted frequencies to the scores indicating the usefulness of the recommended function, in order to improve the system's effectiveness. It is possible that users cannot interpret the predicted frequencies as the usefulness of the functions. In general, the execution frequency of a function is invisible to the users, so that the frequency of execution is not a familiar concept for the users. The system can help users' understanding of the usefulness of the recommended functions by converting the predicted frequencies to the scores indicating usefulness.

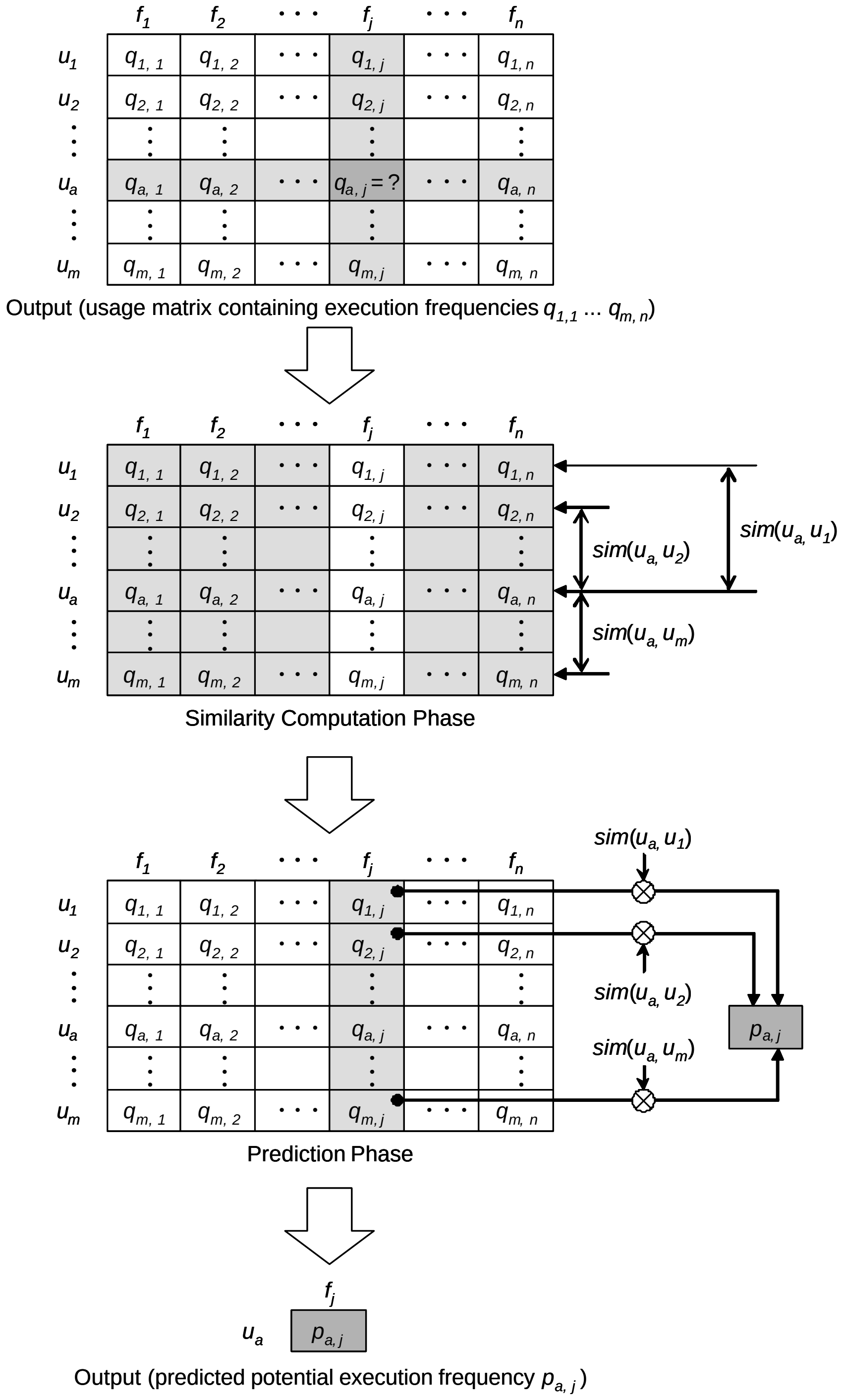
5. Collaborative Filtering Algorithms

5.1 A Common Computation Process

Figure 12 shows the computation process defined by the CF algorithm. Input to this computation process is a usage matrix containing the execution frequencies $q_{1,1}$ ... $q_{m,n}$ as its elements. $q_{i,j}$ denotes the execution frequency of which user u_i has executed the function f_j. Output from this computation process is a predicted potential execution frequency of the unexecuted function $p_{a,j}$, which denotes the potential execution frequency the active user u_a will execute for the function f_j after the function is recommended. Note that $p_{a,j}$ is one of the empty elements of the usage matrix.

A CF algorithm consists of a similarity computation algorithm and a prediction algorithm which define equations to process the similarity computation phase and the prediction phase respectively [41]. In the similarity computation phase, similarity $sim(u_a, u_i)$ is calculated with the known execution frequencies of u_a and u_i, where $sim(u_a, u_i)$ denotes similarity between active user u_a and the other user u_i. In the prediction phase, the potential execution frequency $p_{a,j}$ is calculated with the known execution frequencies of k-similar users and their similarities, where k is called *neighborhood size.* The neighborhood size decides how many similar users' are used in the prediction phase. This size affects the accuracy of the recommendation.

To date, many CF researchers have proposed similarity computation and prediction algorithms. The remainder of this section describes three simple prediction algorithms, ten similarity computation algorithms, and seven prediction algorithms. Simple prediction algorithms lack the similarity computation phase. These algorithms are liable to make lower accuracies; however, they have the advantage that few processing resources are needed because similarity computation is skipped. These algorithms can be used as baseline in evaluating the other CF algorithms. Ten similarity computation algorithms include two new algorithms suitable for function recommendation. Their accuracy is evaluated and compared to the other CF algorithms in the next Chapter 6.


f_1 f_2 · · · f_j · · · f_n
u_1 u_2 u_a u_m
$q_{1,1}$ $q_{1,2}$ · · · $q_{1,j}$ · · · $q_{1,n}$
$q_{2,1}$ $q_{2,2}$ · · · $q_{2,j}$ · · · $q_{2,n}$
$q_{a,1}$ $q_{a,2}$ · · · $q_{a,j} = ?$ · · · $q_{a,n}$
$q_{m,1}$ $q_{m,2}$ · · · $q_{m,j}$ · · · $q_{m,n}$
$sim(u_a, u_1)$
$sim(u_a, u_2)$
$sim(u_a, u_m)$
$p_{a,j}$

Output (usage matrix containing execution frequencies $q_{1,1}$ ... $q_{m,n}$)

Similarity Computation Phase

Prediction Phase

Output (predicted potential execution frequency $p_{a,j}$)

Figure 12. Computation Process Defined by CF Algorithm

5.2 Simple Prediction Algorithms

5.2.1 Random Algorithm

The *Random Algorithm* recommends the functions randomly. Accuracy of the recommendations made by the Random Algorithm must be the lowest border the CF algorithm should achieve because this algorithm has no prediction mechanism. This algorithm can be used as one of the evaluation baselines of the other algorithms. The system predicts the potential execution frequency $p_{a,j}$, as:

$$p_{a,j} = random(0.0 \ldots 1.0) \tag{3}$$

where the term *random*(0.0 ... 1.0) generates a real number between 0.0 to 1.0 randomly.

5.2.2 User Count Algorithm

The *User Count Algorithm* preferentially recommends the functions executed by many users. This algorithm was employed by Terveen et al. [52] for implementing a Web page recommender system *PHOAKS*. Let U denote the set of users who have used the system, and $|U|$ denote the number of users in U. Also, let $q_{i,j}$ denote the execution frequency of which user u_i has executed the function f_j. Therefore, the system predicts the potential execution frequency $p_{a,j}$ as:

$$p_{a,j} = \frac{\sum_{u_i \in U} binaryOf\left(q_{i,j}\right)}{|U|} \tag{4}$$

where the term $binaryOf(q_{i,j})$ is calculated as:

$$binaryOf\left(q_{i,j}\right) = \begin{cases} 0.0 & \left(q_{i,j} \text{ is a empty element}\right) \\ 1.0 & (\text{otherwise}) \end{cases}. \tag{5}$$

The term $binaryOf(q_{i,j})$ of the numerator of equation (5) will be 0.0 if user u_i has not executed the function f_j, and 1.0 otherwise. Therefore, equation (4) will be the ratio of the users who used the function f_j. As a result, the predicted value $p_{a,j}$ of the function used by more users will be larger, and give larger priority in recommending the function.

5.2.3 User Average Algorithm

The *User Average Algorithm* preferentially recommends the functions frequently used among all users. This algorithm was employed by Shardanand et al. [48] to evaluate the accuracy of the recommendations of a music recommender system *Ringo*. Using this algorithm, the system predicts the potential execution frequency $p_{a,j}$, as:

$$p_{a,j} = \frac{\sum_{u_i \in U} q_{i,j}}{|U|} \tag{6}$$

The numerator of equation (6) calculates the sum of all users' frequencies about the function f_j. Then, the sum divided by the denominator which is the number of users $|U|$. As a result, the calculated value $p_{a,j}$ of the function used more frequently will be larger, and give larger priority in recommending the function.

5.3 Similarity Computation Algorithms

5.3.1 Cosine Similarity

The *Cosine Similarity* evaluates the similarity using the traditional information retrieval method. In the field of information retrieval, the similarity between two documents is often measured by treating each document as a vector of word frequencies and by computing the cosine of the angle formed by the two frequency vectors [44]. The Cosine Similarity adopts this formalism to collaborative filtering, where users take the role of documents, functions take the role of words, and execution frequencies take the role of word frequencies. If $q_{i,j}$ denotes the execution frequency of which user u_i has executed function f_j, the system evaluates the similarity $sim(u_a, u_i)$ between the active user u_a and the other user u_i, as:

$$sim(u_a, u_i) = \frac{\sum_{f_j \in F_a \cap F_i} q_{a,j} \cdot q_{i,j}}{\sqrt{\sum_{f_j \in F_a \cap F_i} q_{a,j}^2} \sqrt{\sum_{f_j \in F_a \cap F_i} q_{i,j}^2}} \tag{7}$$

where the squared terms in the denominator serve to normalize execution frequencies so that users who have executed more functions will not *a priori* be more similar to the other users.

5.3.2 Adjusted Cosine Similarity with Average

The *Adjusted Cosine Similarity with Average* extends the Cosine Similarity by assuming that smaller execution frequencies indicate users' negative feelings on the functions [48]. Under the Cosine Similarity, all execution frequencies indicate users' positive feelings of the functions. No role exists for indicating negative feelings. The Adjusted Cosine Similarity with Average employs the average of execution frequencies about each function as the threshold to decide both positive and negative feelings. Execution frequencies below the average are negative, while frequencies above the average are positive. The Adjusted Cosine Similarity with Average will increase the similarity, only when there is an instance where both users have a positive feeling for a function or both of them have a negative feeling. More specifically, equation (7) was altered to become:

$$sim(u_a, u_i) = \frac{\sum_{f_j \in F_a \cap F_i} \left(q_{a,j} - \overline{q_j}\right)\left(q_{i,j} - \overline{q_j}\right)}{\sqrt{\sum_{f_j \in F_a \cap F_i} \left(q_{a,j} - \overline{q_j}\right)^2} \sqrt{\sum_{f_j \in F_a \cap F_i} \left(q_{i,j} - \overline{q_j}\right)^2}} \tag{8}$$

where F_i denotes the set of functions executed by u_i, and $|F_i|$ denotes the number of functions included in F_i. $\overline{q_j}$ denotes the average of the execution frequencies about a function f_j. If U_j denotes a set of users who have executed function f_j, and $|U_j|$ denotes the number of users included in U_j, $\overline{q_j}$ is calculated as:

$$\overline{q_j} = \frac{1}{|U_j|} \sum_{u_k \in U_j} q_{k,j} \, . \tag{9}$$

5.3.3 Adjusted Cosine Similarity with Median

The *Adjusted Cosine Similarity with Median* is an alternative implementation of the Adjusted Cosine Similarity with Average. The Adjusted Cosine Similarity with Median employs the median of execution frequencies about each function as the threshold to decide positive and negative feelings. The arithmetic average is affected by extreme values in the data. If there are several very large values in the data, the average is affected so that it becomes larger. On the other hand, if there are several very small values in the data, the average is affected so that it becomes smaller. The affected average might be not suitable for the threshold. The Adjusted Cosine Similarity with Median avoids this issue by using the median instead of the average. In using this algorithm, equation (7) was altered to become:

$$sim(u_a, u_i) = \frac{\sum_{f_j \in F_a \cap F_i} (q_{a,j} - medianOf(U_j))(q_{i,j} - medianOf(U_j))}{\sqrt{\sum_{f_j \in F_a \cap F_i} (q_{a,j} - medianOf(U_j))^2} \sqrt{\sum_{f_j \in F_a \cap F_i} (q_{i,j} - medianOf(U_j))^2}} \qquad (10)$$

where $medianOf(U_j)$ denotes the median of the execution frequencies related to U_j. If U_j denotes a set of users who have executed function f_j, and $|U_j|$ denotes the number of users included in U_j, $medianOf(U_j)$ is calculated by:

- sorting the execution frequencies related to U_j in ascending order.
- if $|U_j|$ is odd, then the median is the $\{(|U_j|+1)/2\}^{th}$ user's execution frequency.
- if $|U_j|$ is even, then the median is
  $\{(|U_j| / 2)^{th}$ user's frequency + $(|U_j| / 2 + 1)^{th}$ user's frequency$\} / 2$.

5.3.4 Correlation Coefficient

The *Correlation Coefficient* evaluates similarity as an arithmetic correlation coefficient. This algorithm was proposed by Resnick et al. [41] for implementing an e-mail filtering system *GroupLens*. This algorithm can be considered an alternative implementation of the Adjusted Cosine Similarity with Average. The Correlation

Coefficient employs the average of execution frequencies about each user as the threshold to decide positive and negative feelings. The numbers of functions users have executed are different from each other. These numbers affect execution frequency. When the user has executed more functions, each execution frequency tends to become smaller; when the user has executed less functions, each execution frequency tends to become larger. If the system employs the average of execution frequencies about each function as a threshold, the former execution frequency is liable to be decided as a positive feeling, while the latter is liable to be decided as negative. To avoid this issue, this algorithm calculates the threshold about each user individually. In using this algorithm, the system evaluates the similarity *sim*(u_a, u_i) between active user u_a and the other user u_i, as:

$$sim(u_a, u_i) = \frac{\sum_{f_j \in F_a \cap F_i} \left(q_{a,j} - \overline{q_a}\right)\left(q_{i,j} - \overline{q_i}\right)}{\sqrt{\sum_{f_j \in F_a \cap F_i} \left(q_{a,j} - \overline{q_a}\right)^2} \sqrt{\sum_{f_j \in F_a \cap F_i} \left(q_{i,j} - \overline{q_i}\right)^2}} \tag{11}$$

where F_i denotes the set of functions executed by u_i, and $|F_i|$ denotes the number of functions included in F_i. In addition, $\overline{q_i}$ denotes the average of the execution frequencies by user u_i and is calculated as:

$$\overline{q_i} = \frac{1}{|F_i|} \sum_{f_k \in F_i} q_{i,k} = \frac{1}{|F_i|} \quad \left(\because \sum_{f_k \in F_i} q_{i,k} = 1\right). \tag{12}$$

5.3.5 Correlation Coefficient with Median

The *Correlation Coefficient with Median* is an alternative implementation of the Correlation Coefficient. This algorithm employs the median of execution frequencies about each user as the threshold to decide positive and negative feelings, thus avoiding the issue of the average described in section 5.3.3. Equation (11) was altered to become:

$$sim(u_a, u_i) = \frac{\sum_{f_j \in F_a \cap F_i} \left(q_{a,j} - medianOf\left(F_a\right)\right)\left(q_{i,j} - medianOf\left(F_i\right)\right)}{\sqrt{\sum_{f_j \in F_a \cap F_i} \left(q_{a,j} - medianOf\left(F_a\right)\right)^2} \sqrt{\sum_{f_j \in F_a \cap F_i} \left(q_{i,j} - medianOf\left(F_i\right)\right)^2}} \tag{13}$$

where *medianOf*(F_i) denotes the median of the execution frequencies related to F_i. If F_i denotes the set of functions executed by u_i, and $|F_i|$ denotes the number of functions included in F_i, *memdianOf*(F_i) is calculated by:

- sorting the execution frequencies related to F_i in ascending order.
- if $|F_i|$ is odd, then the median is the execution frequency of $\{(|F_i|+1)/2\}^{th}$ function.
- if $|F_i|$ is even, then the median is

  {frequency of $(|F_i| / 2)^{th}$ function + frequency of $(|U_j| / 2 + 1)^{th}$ function} / 2.

5.3.6 Binary Cosine Similarity

The *Binary Cosine Similarity* is a variant of the Cosine Similarity. This algorithm was employed by O'Sullivan et al. [36] for implementing a TV program recommender system *PTVPlus*. This algorithm is used by many CF recommender systems as well as PTVPlus because the data interpretation is simpler and more accurate than the other algorithms. Binary Cosine Similarity ignores the value of execution frequency, and only focuses on whether the function has been executed or not. Even if execution frequency has valuable information about the usage of a software function, if the system misinterprets such usage, the accuracy of recommendation will be lower than in conventional systems. This algorithm avoids this misinterpretation and debasement of accuracy. More specifically, the equation (7) was altered to become:

$$sim(u_a, u_i) = \frac{\sum_{f_j \in F} \left\{ binaryOf(q_{a,j}) \times binaryOf(q_{i,j}) \right\}}{\sqrt{\sum_{f_j \in F} binaryOf(q_{a,j})} \sqrt{\sum_{f_j \in F} binaryOf(q_{i,j})}} \tag{14}$$

where F_i denotes the set of executed functions by user u_i, and the term *binaryOf*($q_{i,j}$) is calculated by the equation (5).

5.3.7 Rank Correlation

The *Rank Correlation* evaluates similarity as *Spearman's Rank Correlation* [20]. This algorithm is a new algorithm defined by this Framework for improving the accuracy of the recommendations. This algorithm reduces the impacts of the outliers in evaluating similarity [32]. Outliers are observations that deviate so much from other observations that they can impact the result of the Correlation Coefficient [42]. The proposed framework has to take into account the outliers because the system based on the framework evaluates similarities between the execution frequencies which containing outliers as shown in Table 2. In the table, each column indicates the HFAs. Each row indicates the Range of the frequencies where $\overline{q_i}$ and ∂_i denote the average and standard deviation of the frequencies of user u_i, respectively. Each element has the percentage of execution frequencies included in the range of the corresponding row, regarding each HFA user. Table 2 indicates that the unignorable outliers deviate more than $3\partial_i$ from the average. As a result, these outliers can be dominative in calculating correlation coefficients. In other words, they tend to generate extremely high similarities. Conventional recommender systems such as GroupLens do not consider the outliers since these systems employ data limited by range of values to track the users' preferences. In using this algorithm, the system evaluates the similarity $sim(u_a, u_i)$ between the active user u_a and the other user u_i, as:

$$sim(u_a, u_i) = \frac{\left\{ \sum_{f_j \in F_a \cap F_i} rankOf\left(q_{a,j}\right)^2 + \sum_{f_j \in F_a \cap F_i} rankOf\left(q_{i,j}\right)^2 - d^2 \right\} \times \frac{n}{2} - T^2}{\sqrt{n \sum_{f_j \in F_a \cap F_i} \left(rankOf\left(q_{a,j}\right)^2 - T^2 \right)} \sqrt{n \sum_{f_j \in F_a \cap F_i} \left(rankOf\left(q_{i,j}\right)^2 - T^2 \right)}} \tag{15}$$

where F_i denotes the set of executed functions by user u_i, and n denotes the number of functions included in $F_a \cap F_i$. In addition, T and d^2 are calculated as:

$$T = n(n+1)/2 \tag{16}$$

$$d^2 = \sum_{f_j \in F_a \cap F_i} \left(rankOf\left(q_{a,j}\right) - rankOf\left(q_{i,j}\right) \right)^2 \tag{17}$$

where the term $rankOf(q_{i,j})$ denotes the descending order of frequency $q_{i,j}$ in the sorted frequencies of the functions included in F_i. For instance, if f_j is the most frequently used function by user u_i, $rankOf(q_{i,j})$ must be the smallest (i.e., 1). On the other hand, if f_j is used only once by user u_i, $rankOf(q_{i,j})$ must be the largest.

Table 2. Distribution of the Execution Frequencies

(Subjects: 19 users in our lab --- Consultation Period: average 9.44 months)

Range of the Frequencies	Word2002	PowerPoint2002	Excel2002	Average
$q_i \leq \overline{q_i} - 4\partial_i$	0.00%	0.00%	0.00%	0.00%
$\overline{q_i} - 4\partial_i < q_i \leq \overline{q_i} - 3\partial_i$	0.00%	0.00%	0.00%	0.00%
$\overline{q_i} - 3\partial_i < q_i \leq \overline{q_i} - 2\partial_i$	0.00%	0.00%	0.00%	0.00%
$\overline{q_i} - 2\partial_i < q_i \leq \overline{q_i} - \partial_i$	0.00%	0.00%	0.00%	0.00%
$\overline{q_i} - \partial_i < q_i \leq \overline{q_i}$	87.05%	86.04%	84.42%	85.84%
$\overline{q_i} < q_i \leq \overline{q_i} + \partial_i$	8.41%	6.84%	7.61%	7.62%
$\overline{q_i} + \partial_i < q_i \leq \overline{q_i} + 2\partial_i$	1.10%	3.06%	2.61%	2.26%
$\overline{q_i} + 2\partial_i < q_i \leq \overline{q_i} + 3\partial_i$	1.08%	1.03%	2.27%	1.46%
$\overline{q_i} + 3\partial_i < q_i \leq \overline{q_i} + 4\partial_i$	0.84%	0.99%	0.93%	0.92%
$\overline{q_i} + 4\partial_i < q_i$	1.53%	2.03%	2.15%	1.90%
Total	100.00%	100.00%	100.00%	100.00%

5.3.8 Magnitude Relation

The *Magnitude Relation* evaluates similarity as Yao's *Distance-based Performance Measure* (*DPM*) [56]. This algorithm is a new algorithm defined by this Framework for improving the accuracy of the recommendations [30], [31]. The Magnitude Relation reduces the impact of the outliers in evaluating similarity as well as in evaluating the Rank Correlation. Also, this algorithm avoids the issue of using the Rank Correlation. The Rank Correlation converts execution frequencies to the ranks in its computation process. The value of a rank has equal distance to the other ranks' values. If many execution frequencies have few differences, some execution frequencies might be converted to some much distanced ranks. As a result, the similarity between users may be evaluated as relatively smaller than the actual difference. The Magnitude Relation avoids this problem by using Yao's DPM instead of the Rank Correlation. The system evaluates each similarity $sim(u_a, u_i)$ between the active user u_a who gets the recommendation and each other user u_i, as:

$$sim(u_a, u_i) = 1 - 2 \times \frac{\sum_{f_j \in F_a \cap F_i} \sum_{f_k \in F_a \cap F_i} dpm(q_{a,j}, q_{a,k}, q_{i,j}, q_{i,k})}{\sum_{f_j \in F_a \cap F_i} \sum_{f_k \in F_a \cap F_i} dpmNormalizer(q_{a,j}, q_{a,k})} \tag{18}$$

where, F_a and F_i denotes the set of executed functions by user u_a and u_i respectively. In addition, the terms $dpm(q_{a,j}, q_{a,k}, q_{i,j}, q_{i,k})$ and $dpmNormalizer(q_{a,j}, q_{a,k})$ are calculated as:

$$dpm(q_{a,j}, q_{a,k}, q_{i,j}, q_{i,k}) = \begin{cases} 0 & \left(\begin{array}{c} (q_{a,j} > q_{a,k}) \wedge (q_{i,j} > q_{i,k}) \vee (q_{a,j} = q_{a,k}) \wedge (q_{i,j} = q_{i,k}) \\ \vee (q_{a,j} < q_{a,k}) \wedge (q_{i,j} < q_{i,k}) \end{array} \right) \\ 1 & ((q_{a,j} = q_{a,k}) \wedge (q_{i,j} \neq q_{i,k}) \vee (q_{a,j} \neq q_{a,k}) \wedge (q_{i,j} = q_{i,k})) \\ 2 & ((q_{a,j} > q_{a,k}) \wedge (q_{i,j} < q_{i,k}) \vee (q_{a,j} < q_{a,k}) \wedge (q_{i,j} > q_{i,k})) \end{cases} \tag{19}$$

$$dpmNormalizer(q_{a,j}, q_{a,k}) = \begin{cases} 1 & (q_{a,j} = q_{a,k}) \\ 2 & (q_{a,j} \neq q_{a,k}) \end{cases} \tag{20}$$

DPM calculated with the equation (19) compares magnitude relations between each couple of the execution frequencies ($q_{a,j}$, $q_{a,k}$) and ($q_{i,j}$, $q_{i,k}$), corresponding to each couple of functions (f_k, f_l) included in the intersection $F_a \cap F_i$. DPM must be 0 when the magnitude relations between ($q_{a,j}$, $q_{a,k}$) and ($q_{i,j}$, $q_{i,k}$) are equal. Also, the DPM must be 2 when the magnitude relations between ($q_{a,j}$, $q_{a,k}$) and ($q_{i,j}$, $q_{i,k}$) are different from each other. In addition, the DPM must be 1 when either ($q_{a,j}$, $q_{a,k}$) or ($q_{i,j}$, $q_{i,k}$) has no difference in magnitude relation. This algorithm is not affected by the outliers since the DPM does not use the values of execution frequencies directly. Furthermore, this algorithm is not affected by the functions with few differences in the execution frequencies since each execution frequency is not converted to any rank. The second term of the equation (18) is a variant of the *NDPM* (*Normalized Distance-based Performance Measure*) calculated with the equation (32) as described later. NDPM can be considered a similarity computation method because it evaluates the difference between "user u_a's ideal recommendation R_a required to the system" and the "system's actual recommendation R_a' generated to user u_a". The second term of the equation (18) evaluates the difference between user u_a and user u_i by assigning the execution frequencies of the functions included in the intersection $F_a \cap F_i$ instead of the recommendations.

5.3.9 Average Squared Difference

The *Average Squared Difference* measures the degree of dissimilarity between two users' frequencies. This algorithm was employed by Shardanand et al. [48] for implementing a music recommender system *Ringo*. The basic idea of this algorithm is to calculate non-similarity among users by Euclidean distance. The numerator and denominator of calculated non-similarity are inverted for converting as a similarity. In using this algorithm, the system evaluates the similarity $sim(u_a, u_i)$ between active user u_a and the other user u_i as:

$$sim(u_a, u_i) = \frac{|F_a \cap F_i|}{\sum_{f_j \in F_a \cap F_i} \sqrt{(q_{a,j} - q_{i,j})^2}}. \tag{21}$$

The numerator of the equation normalizes squared difference with the number of execution frequencies used in calculation, so that users who have executed more functions will not *a priori* be more similar to the other users.

5.3.10 Median of Squared Difference

The *Median of Squared Difference* is an alternative implementation of the Average Squared Difference. This algorithm employs the median to normalize squared difference, thereby avoiding the issue of the average described in section 5.2.3. More specifically, the equation (21) was altered to become:

$$sim(u_a, u_i) = \frac{1}{medianOf\left(results_{f_j \in F_a \cap F_i}\left(\sqrt{(q_{a,j} - q_{i,j})^2}\right)\right)} \tag{22}$$

where $medianOf\left(results_{f_j \in F_a \cap F_i}\left(\sqrt{(q_{a,j} - q_{i,j})^2}\right)\right)$ denotes the median of $\sqrt{(q_{a,j} - q_{i,j})^2}$, for the function f_j included in $F_a \cap F_i$. If F_i denotes the set of functions executed by u_i, and $|F_i|$ denotes the number of functions included in F_i, $medianOf\left(results_{f_j \in F_a \cap F_i}\left(\sqrt{(q_{a,j} - q_{i,j})^2}\right)\right)$ is calculated by:

- sorting the the results of the equation $\sqrt{(q_{a,j} - q_{i,j})^2}$ in ascending order.
- if $|F_a \cap F_i|$ is odd, then the median is the $\{(|F_a \cap F_i| + 1)/2\}^{\text{th}}$ result.
- if $|F_a \cap F_i|$ is even, then the median is
  { $(|F_a \cap F_i|/2)^{\text{th}}$ result + $(|F_a \cap F_i|/2 + 1)^{\text{th}}$ result} / 2.

5.4 Prediction Algorithms

5.4.1 Weighted Sum

The *Weighted Sum* calculates the active user's potential execution frequencies with the weighted sum of similar users' execution frequencies. This algorithm uses similarity $sim(u_a, u_i)$ between the active user u_a and user u_i as the weights. If $q_{i,j}$ denotes the execution frequency of which user u_i has executed function f_j, and $sim(u_a, u_i)$ denotes the similarity between the active user u_a and user u_i, the system predicts the potential execution frequency $p_{a,j}$, as:

$$p_{a,j} = \frac{\sum_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i) q_{i,j}}{\sum_{u_i \in k\text{-}NearestNeighbors} |sim(u_a, u_i)|} \tag{23}$$

where *k-NearestNeighbors* denotes the set of *k*-users who have executed function f_j and are the most similar to the active user u_a. *k* is a natural number specifying the number of similar users whose execution frequencies are used for prediction. *k* affects the result and accuracy of the prediction, so it should be decided by empirical evaluation by using actually collected data. The potential frequency $p_{a,j}$ of the function more frequently used by similar users will be larger, and give larger priority in recommending the function.

5.4.2 Adjusted Weighted Sum with Average of Column

The *Adjusted Cosine Similarity with Average of Column* extends the Weighted Sum by introducing the idea of a user's negative and positive feelings as described in section 5.3.2. This algorithm calculates the deviation from the average execution frequency regarding each function. The calculated deviation is summed with the average execution frequency regarding the function. The "column" in the algorithm name indicates that each average is calculated from the column of the usage matrix shown in the Figure 12. The system predicts the potential execution frequency $p_{a,j}$ as:

$$p_{a,j} = \overline{q_j} + \frac{\sum_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i)\left(q_{i,j} - \overline{q_j}\right)}{\sum_{u_i \in k\text{-}NearestNeighbors} \left|sim(u_a, u_i)\right|} \tag{24}$$

where *k-NearestNeighbors* denotes the set of *k*-users who have executed function f_j and are the most similar to active user u_a. $\overline{q_j}$ denotes the average of the execution frequencies about a function f_j. $\overline{q_j}$ is calculated with equation (9).

5.4.3 Adjusted Weighted Sum with Median of Column

The *Adjusted Weighted Sum with Median of Column* is an alternative implementation of the Adjusted Weighted Sum with Average of Column. This algorithm employs the median of execution frequencies regarding each function as baseline in calculating deviation, thereby avoiding the issue of the average described in section 5.3.3. More specifically, equation (24) was altered to become:

$$p_{a,j} = \overline{q_a} + \frac{\sum_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i)\left(q_{i,j} - medianOf\left(U_j\right)\right)}{\sum_{u_i \in k\text{-}NearestNeighbors} \left|sim(u_a, u_i)\right|} \tag{25}$$

where *k-NearestNeighbors* denotes the set of *k*-users who have executed function f_j and are the most similar to active user u_a. *medianOf*(U_j) denotes the median of the execution frequencies related to U_j. *medianOf*(U_j) is calculated as described in section 5.3.3.

5.4.4 Adjusted Weighted Sum with Average of Neighbors

The *Adjusted Weighted Sum with Average of Neighbors* is an alternative implementation of the Adjusted Weighted Sum with Average of Column. This algorithm employs the average execution frequencies calculated among *k*-similar users as the baseline in calculating deviation. Under the Adjusted Weighted Sum with Average of Column, although deviation of the active user's potential frequency is predicted from similar users' execution frequencies, the baseline for calculating deviation is decided from all users' execution frequencies. This inconsistency might debase the accuracy of prediction. The Adjusted Weighted Sum with Average of Neighbors avoids this issue by calculating the average from *k*-nearest neighbors. More specifically, the equation (24) was altered to become:

$$p_{a,j} = averageOf\left(k\text{ - }NearestNeighbors\right) + \frac{\sum_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i)\left(q_{i,j} - averageOf\left(k\text{ - }NearestNeighbors\right)\right)}{\sum_{u_i \in k\text{-}NearestNeighbors} \left|sim(u_a, u_i)\right|} \tag{26}$$

where *k-NearestNeighbors* denotes the set of *k*-users who have executed function f_j and are the most similar to active user u_a. *averageOf*(*k-NearestNeighbors*) denotes the average of the nearest neighbors' execution frequencies about a function f_j. *averageOf*(*k-NearestNeighbors*) is calculated as:

$$averageOf\left(k\text{ - }NearestNeighbors\right) = \frac{1}{k} \sum_{u_l \in k\text{-}NearestNeighbors} q_{l,j} \quad . \tag{27}$$

5.4.5 Adjusted Weighted Sum with Median of Neighbors

The *Adjusted Weighted Sum with Median of Neighbors* is an alternative implementation of the Adjusted Weighted Sum with Median of Column. This algorithm employs the median of execution frequencies among *k*-similar users as the baseline in calculating deviation. This algorithm can avoid the issue of inconsistency described in the section 5.4.4. More specifically, equation (25) was altered to become:

$$p_{a,j} = medianOf\left(k\text{ - }NearestNeighbors\right) + \frac{\sum\limits_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i)\left(q_{i,j} - medianOf\left(k\text{ - }NearestNeighbors\right)\right)}{\sum\limits_{u_i \in k\text{-}NearestNeighbors} \left|sim(u_a, u_i)\right|} \qquad (28)$$

where *k-NearestNeighbors* denotes the set of *k*-users who have executed function f_j and are the most similar to active user u_a. *medianOf*(U_j) denotes the median of the execution frequencies related to U_j. *medianOf*(U_j) is calculated as described in section 5.3.3. *medianOf*(*k-NearestNeighbors*) denotes the median of the nearest neighbors' execution frequencies about a function f_j. *medianOf*(*k-NearestNeighbors*) is calculated by:

- sorting the execution frequencies related to function f_j executed by users included in *k-NearestNeighbors,* in ascending order.
- if *k* is odd, then the median is the {(*k*+1)/2}th neighbor's execution frequency.
- if *k* is even, then the median is
  {(*k* / 2)th neighbor's frequency + (*k* / 2 + 1)th neighbor's frequency} / 2.

5.4.6 Adjusted Weighted Sum with Average of Row

The *Adjusted Cosine Similarity with Average of Row* extends the Weighted Sum by introducing the idea of a user's negative and positive feelings as described in section 5.3.2. This algorithm can be considered an alternative implementation of the Adjusted Weighted Sum with Average of Column described in the section 5.4.2. In contrast to the Adjusted Weighted Sum with Average of Column, this algorithm calculates the deviation from the average execution frequency regarding each "user". Likewise, the calculated deviation is summed with the average execution frequency regarding the "user". The "row" in the algorithm name indicates each average is calculated from the row of the usage matrix shown in the Figure 12. The system predicts the potential execution frequency $p_{a,j}$, as:

$$p_{a,j} = \overline{q_a} + \frac{\sum_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i)\left(q_{i,j} - \overline{q_i}\right)}{\sum_{u_i \in k\text{-}NearestNeighbors} \left|sim(u_a, u_i)\right|} \tag{29}$$

where, *k-NearestNeighbors* denotes the set of *k*-users who have executed function f_j and are the most similar to active user u_a. $\overline{q_i}$ denotes the average of the execution frequencies by user u_i. $\overline{q_i}$ is calculated with equation (12).

5.4.7 Adjusted Weighted Sum with Median of Row

The *Adjusted Weighted Sum with Median of Row* is an alternative implementation of the Adjusted Weighted Sum with Average of Row. This algorithm employs the median of execution frequencies regarding each user as the baseline in calculating deviation, thereby avoiding the issue of the average described in section 5.3.3. More specifically, equation (29) was altered to become:

$$p_{a,j} = medianOf\left(F_a\right) + \frac{\sum_{u_i \in k\text{-}NearestNeighbors} sim(u_a, u_i)\left(q_{i,j} - medianOf\left(F_i\right)\right)}{\sum_{u_i \in k\text{-}NearestNeighbors} \left|sim(u_a, u_i)\right|} \tag{30}$$

where, *medianOf*(F_i) denotes the median of the execution frequencies related to F_i. If F_i denotes the set of functions executed by u_i, and $|F_i|$ denotes the number of functions included in F_i, *memdianOf*(F_i) is calculated as described in section 5.3.5.

6. Empirical Evaluation

6.1 Overview

In order to evaluate the effectiveness of the proposed framework, two experimental evaluations (Experiments 1 and 2) have been executed [30], [31], [32]. Furthermore, in order to make clear which algorithms are suitable for recommending software functions, an experimental evaluation (Experiment 3) has been executed. The purpose of Experiments 1 and 2 is to compare the accuracy of some CF algorithms and the accuracy of the three simple prediction algorithms. The purpose of Experiment 3 is to compare the accuracy of the CF algorithms to each other.

In order to simulate a real situation in operating the function recommender system, the usage histories were collected from the users who had used Microsoft Office

Table 3. Collected Usage Histories in the Experiments

Experiment	#1	#2			#3		
HFA	Word 2000	Word 2002	PPT 2002	Excel 2002	Word 2002	PPT 2002	Excel 2002
# of users	6	23	21	20	19	22	16
Max Collection Period (months)	22	14	10	14	25	21	25
Min Collection Period (months)	3	1	1	1	1	1	1
Avg. Collection Period (months)	12.0	5.0	4.6	5.5	8.9	7.9	11.5
Max # of Functions Used	108	143	162	77	226	249	157
Min # of Functions Used	62	16	29	15	26	46	30
Avg. # of Functions Used	90.0	67.3	90.9	42.0	87.9	115.0	64.2
# of Total Functions Used	209	227	254	132	387	397	256

Applications for various objectives. Table 3 shows the usage histories collected in Experiments 1, 2 and 3. In Experiment 1, usage histories were collected in an academic environment. In Experiment 2 and 3, usage histories were collected in an academic environment and in an industrial environment. In the academic environment, subjects were the students and staff of the Graduate School of Information Science, Nara Institute of Science and Technology. In the industrial environment, subjects were the employees of a Japanese software development company. The subjects in the academic environment had used mainly MS Word for writing academic papers, MS PowerPoint for crafting presentations, and MS Excel for carrying out statistical work. The subjects in the industrial environment had used mainly MS Word for writing software specifications, MS PowerPoint for crafting presentations, and MS Excel for writing software specifications. The usage histories were collected from these subjects from 1 to 25 months.

Using the collected usage histories, each CF algorithm was evaluated with two kinds of evaluation criteria (*NDPM*: *Normalized Distance-based Performance Measure* and *ARE*: *Average Relative Error*) in two kinds of procedures (*Interview-based Procedure* and *History-based Procedure*). The remainder of this chapter describes the employed evaluation criteria, the evaluation procedures, and the evaluation results, in the sections 6.2, 6.3 and 0 respectively.

6.2 Evaluation Criteria

Two basic classes of recommendations based on CF exist. In the first class [4], individual items are recommended one-at-a-time to the users along with a score indicating the potential user's preference. For example, *GroupLens* [41] was in this category. Each article is recommended with a GUI Netnews reader interface that includes an ASCII bar-chart indicating the system's prediction regarding the user's potential interest in that article. Thus, each piece of content has an associated estimated score, and the user interface displays this score along with a recommendation of the item. The second class of the recommendation presents the user with a ranked list of recommended items. For example, *PHOAKS* [52] and *Siteseer* [43] are in this category. Likewise, the Internet search engine provides a ranked list of items (URLs) according to scores indicating the potential user's preferences. In the experiments, two kinds of scoring metrics were employed for evaluating the accuracy of both classes of recommendations. In both cases, the basic evaluation sequence proceeds as follows. The usage matrix is prepared from the collected usage histories. An execution frequency in the usage matrix is picked out and hidden. The other execution frequencies are used as the CF database to predict the hidden execution frequency. After this cycle is repeated through all execution frequencies, for each user, the following metrics are calculated with the predicted execution frequencies. regarding each user. The calculated metrics are averaged among all users, then used as an evaluation criteria indicating the accuracy of recommendations.

In order to evaluate the accuracy of individual scoring, the *Average Relative Error* (*ARE*) is calculated between the predicted execution frequency and the actual execution frequency. ARE has been widely employed for the evaluation accuracy of various prediction algorithms [6], [25], [50]. ARE takes the value of the real number of the range [0.0, 1.0]. A smaller value of ARE indicates that there is a smaller error between $p_{a,\,j}$ and $q_{a,\,j}$, i.e., the accuracy of the individual scoring is high. If $q_{a,j}$ denotes the execution frequency of which the active user u_a has executed with function f_j, and the $p_{a,j}$ denotes the predicted potential execution frequency, ARE is calculated as:

$$ARE(u_a) = \frac{1}{|F_a|} \sum_{f_j \in F_a} \frac{|p_{a,j} - q_{a,j}|}{q_{a,j}} \tag{31}$$

An implicit assumption in using ARE as a measure of accuracy is that the error is proportional to the size of the actual value. For example, an overestimation of 0.01 for the actual value of 0.05 is more serious than an over estimation of 0.01 for the actual value of 0.5. The denominator of the equation (31) normalizes size of the actual value, so that larger execution frequency will not *a priori* make a larger error.

In order to evaluate the accuracy of the ranked items, the *Normalized*

Distance-based Performance Measure (*NDPM*) [56] is calculated by comparing the ranked functions regarding the predicted potential execution frequencies to the ranked functions regarding the actual execution frequencies. NDPM has been widely employed for evaluating accuracies of the CF algorithms or information retrieval algorithms [1], [2], [40]. NDPM measures the difference between "the user u_a's ideal recommendation R_a" and "the system's recommendation R_a' for the user u_a". NDPM takes the value of a real number in the range of [0.0, 1.0]. A smaller value of NDPM indicates there is less difference between R_a and R_a', i.e. the accuracy of the ranked items is high. The intermediate value 0.5 indicates the theoretical accuracy of the Random Prediction algorithm [56]. Accuracy of the Random Prediction algorithm is the same when the system displays all candidate functions, without any ranks or scores indicating the usefulness of the recommended function. Thus, the value of NDPM 0.5 is the lowest border of accuracy the CF algorithms should achieve in making recommendations. If $r_{a,j}$ and $r_{a,j}$' denote the ranks (priority for recommending) of the function f_k in the recommendation R_a and R_a' respectively, NDPM is calculated as:

$$NDPM\left(R_a, R_a'\right) = \frac{\sum_{r_j \in R_a \cup R_a'} \sum_{r_k \in R_a \cup R_a'} dpm\left(r_{a,j}, r_{a,k}, r_{a,j}', r_{a,k}'\right)}{\sum_{r_j \in R_a \cup R_a'} \sum_{r_k \in R_a \cup R_a'} dpmNormalizer\left(r_{a,j}, r_{a,k}\right)} \tag{32}$$

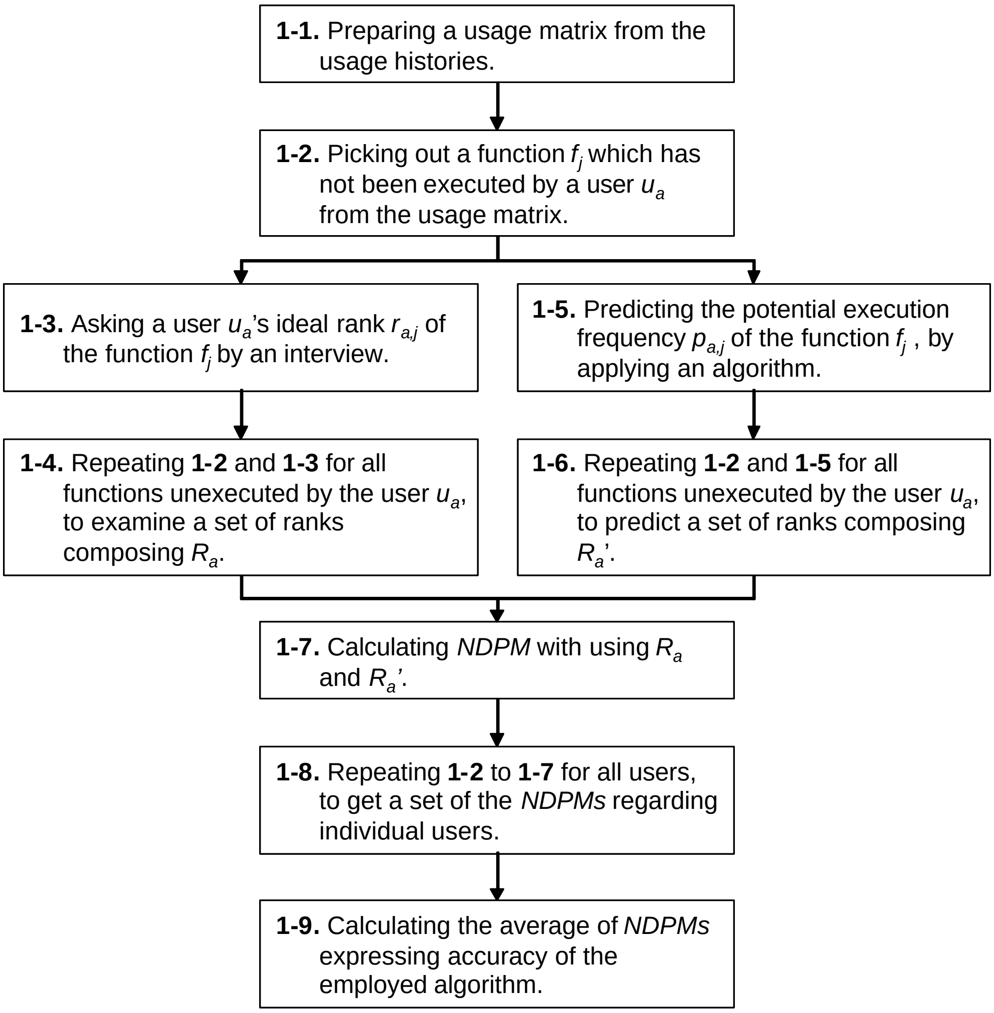
where, the term *dpm* and the term *dpmNormalizer* are calculated with equations (19) and (20) respectively.

6.3 Evaluation Procedure

6.3.1 Interview-based Evaluation

Figure 13 shows the procedure of the first type of evaluation called the *Interview-based Evaluation.* In this evaluation, each user's ideal recommendation is obtained by questioning to the user directly through an interview. The ideal recommendation R_a is examined by asking each user u_a the rank of each function f_j with the following criteria:

- If user u_a knows the function f_j and has used it before for collecting the usage histories, user u_a should choose the lowest rank for function f_j.
- If user u_a knows the function f_j but has not used it, user u_a should choose the second lowest rank for function f_j.
- If user u_a does not previously know the function f_j, then the experiment conductor (in this case interviewer) gives user u_a a brief description about function f_j. Next, user u_a decides the ideal rank $r_{a,j}$ of function f_j, according to the expected future execution of function f_j. In other words, I asked user u_a the following: how frequently does the user uses function f_j after the function is recommended? The user was able to choose



1-1. Preparing a usage matrix from the usage histories.
1-2. Picking out a function f_j which has not been executed by a user u_a from the usage matrix.
1-3. Asking a user u_a's ideal rank $r_{a,j}$ of the function f_j by an interview.
1-5. Predicting the potential execution frequency $p_{a,j}$ of the function f_j , by applying an algorithm.
1-4. Repeating **1-2** and **1-3** for all functions unexecuted by the user u_a, to examine a set of ranks composing R_a.
1-6. Repeating **1-2** and **1-5** for all functions unexecuted by the user u_a, to predict a set of ranks composing R_a'.
1-7. Calculating *NDPM* with using R_a and R_a'.
1-8. Repeating **1-2** to **1-7** for all users, to get a set of the *NDPMs* regarding individual users.
1-9. Calculating the average of *NDPMs* expressing accuracy of the employed algorithm.

Figure 13. Procedure of Interview-based Evaluation

the same level rank for more than two functions if the user thought these were used with the same frequency.

The evaluation criteria described in section 6.2 are calculated by comparing each R_a and the system's recommendation R_a'. Note that ARE cannot be calculated in the Interview-based Evaluation because the subjects cannot decide the actual execution frequency for functions that have not been used. The Interview-based Evaluation has the advantage that the users' real needs affect the evaluation; however, finding a lot of subjects to answer questions is difficult because both the interviewer (i.e. experimenter) and interviewee have to spend a lot of time and effort in the interviewing process.

6.3.2 History-based Evaluation

Figure 14 shows the procedure of the second type of evaluation called the History-based Evaluation. In this evaluation, each user's ideal recommendation R_a is obtained by examining actual execution frequencies from the user's usage histries. Note that the system's recommendation R_a' is obtained by calculating potential execution frequencies of already executed functions; however, some algorithms can make perfect predictions if the actual value is available. In order to avoid this type of prediction obtained by cheating, the *Jackknife* method [8] is used. The actual execution frequency $p_{a,j}$ is temporally hidden when the potential execution frequency $q_{a,j}$ is predicted. In other words, each algorithm assumes that each user u_a has not executed the function f_j so that the actual execution frequency is not available. Next, the algorithm predicts the potential execution frequency $p_{a,j}$ without using the actual frequency $q_{a,j}$. The jackknife method has been widely used in various literatures evaluating the accuracies of CF algorithms [4], [41], [46], [48]. The History-based Evaluation has the advantage that it can query many of the subjects because evaluation can be automated without a face-to-face interview with the subjects; however, an inconsistency between the users' real needs and the actual execution frequencies might exist.

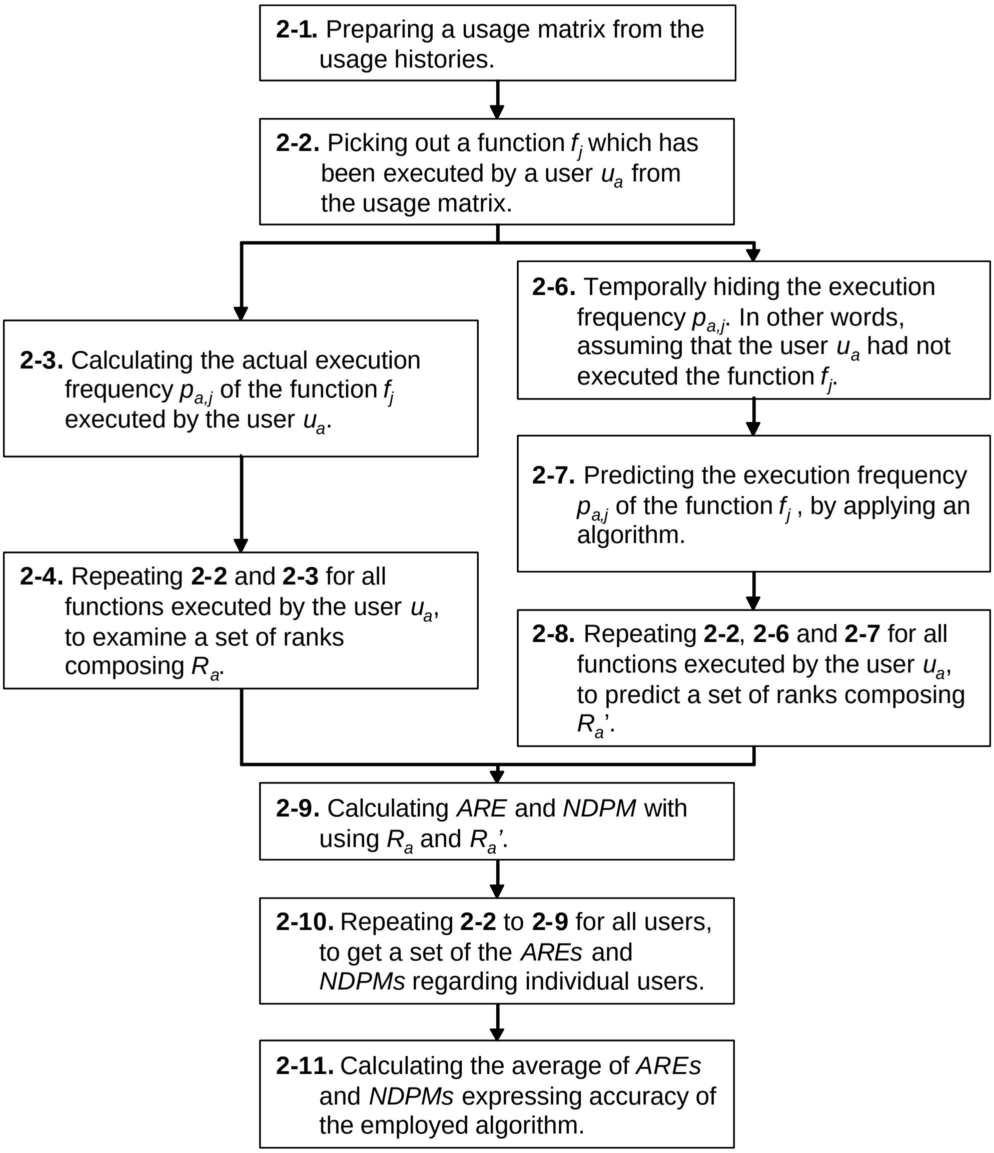

2-1. Preparing a usage matrix from the usage histories.
2-2. Picking out a function f_j which has been executed by a user u_a from the usage matrix.
2-3. Calculating the actual execution frequency $p_{a,j}$ of the function f_j executed by the user u_a.
2-4. Repeating **2-2** and **2-3** for all functions executed by the user u_a, to examine a set of ranks composing R_a.
2-6. Temporally hiding the execution frequency $p_{a,j}$. In other words, assuming that the user u_a had not executed the function f_j.
2-7. Predicting the execution frequency $p_{a,j}$ of the function f_j , by applying an algorithm.
2-8. Repeating **2-2**, **2-6** and **2-7** for all functions executed by the user u_a, to predict a set of ranks composing R_a'.
2-9. Calculating *ARE* and *NDPM* with using R_a and R_a'.
2-10. Repeating **2-2** to **2-9** for all users, to get a set of the *AREs* and *NDPMs* regarding individual users.
2-11. Calculating the average of *AREs* and *NDPMs* expressing accuracy of the employed algorithm.

Figure 14. Procedure of History-based Evaluation

6.4 Evaluation Results

Experiment 1: Interview-based Evaluation of the Proposed Framework

In Experiment 1, usage histories were collected from 6 users of MS Word 2000 in approximately 12 months as shown in Table 3. Two simple prediction algorithms (User Count and User Average) and two similarity computation algorithms (Correlation

Coefficient and Magnitude Relation) were evaluated by the Interview-based Evaluation. Only the Adjusted Weighted Sum with Average of Row with the neighborhood size $k = 6$ (all users) was employed as the prediction algorithm.

Figure 15 shows the result of Experiment 1. In the graph, the vertical axis indicates NDPM; and the horizontal axis indicates each algorithm. Note that the algorithm names are written with the abbreviations for convenience of description (for the abbreviations, see Appendix A). Each NDPM calculated with the recommendation for each user is marked with a symbol “✕”; each average NDPM of the six users is marked with a symbol “▬”, and its each average value is written at the right of the symbol. These NDPMs represent the accuracy of each algorithm; NDPM 0.5 is emphasized with a broken line “----” because it is the theoretical accuracy of the Random Prediction Algorithm and is the lowest border the CF algorithm should achieve. As described in 6.2, a smaller NDPM (i.e. the symbols closer to bottom of the graph) indicates a higher accuracy. Therefore, if certain algorithms outperformed other algorithms, the average NDPM is lower than the other’s average NDPM. The algorithms are ranked in

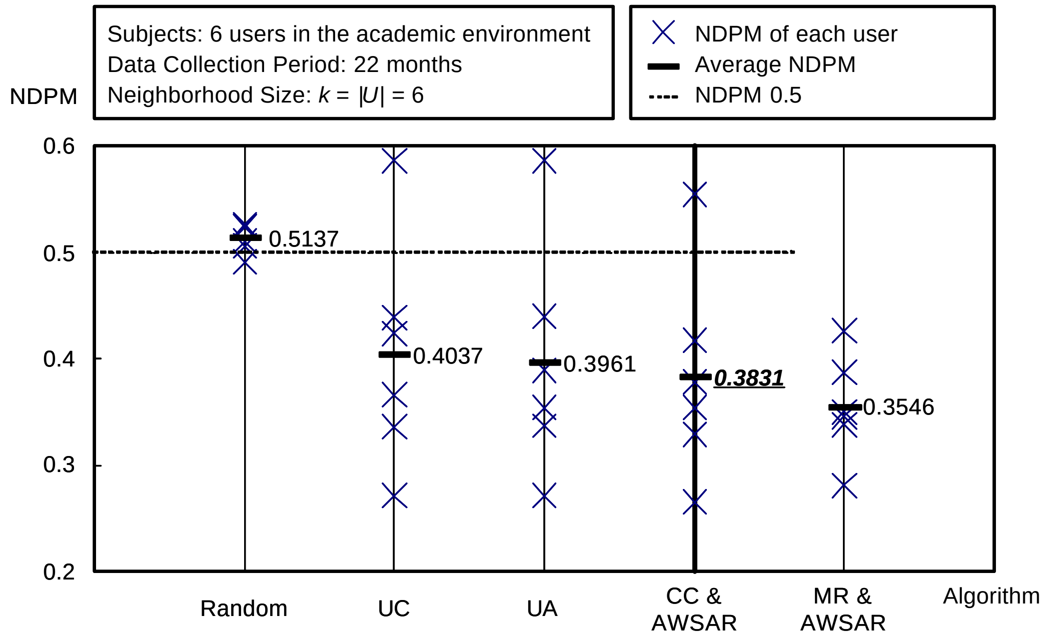


Figure 15. Result of Experiment 1: NDPM

descending order of the average NDPMs from left to right below the horizontal axis, so that the algorithms closer to the right side of graph achieved better accuracy. In addition, the average NDPM written with emphasized letters (italic, underlined, boldface) and an emphasized vertical line indicate the paired Student's *t*-test statistically observed significant differences to the left algorithm's NDPM, at the level of $p < 0.05$.

As a result, all average NDPMs of the CF algorithms are lower than the 0.5. These algorithms were specially sorted according to the average NDPMs as: Magnitude Relation (Best), Correlation Coefficient, User Average, and User Count (Worst). The paired Student's *t*-test statistically observed significant difference between the Correlation Coefficient and the User Average at the level of p < 0.018. For more detail of the evaluation results, see Appendix C.1.

***Experiment 2*: History-based Evaluation of the Proposed Framework**

In Experiment 2, usage histories were collected from 20 to 23 users of MS Word 2002, PowerPoint 2002, and Excel 2002 in approximately 5 months as shown in Table 3.

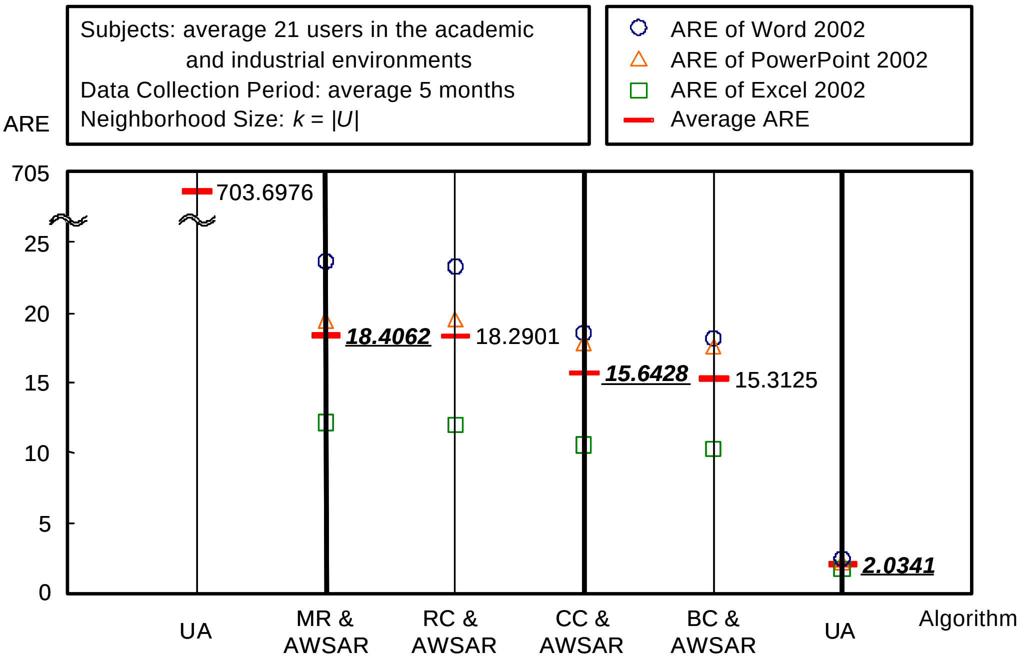


Figure 16. Result of Experiment 2: ARE

Three simple prediction algorithms (User Count and User Average) and four similarity computation algorithms (Correlation Coefficient, Binary Correlation, Rank Correlation, and Magnitude Relation) were evaluated by the History-based Evaluation. Only the Adjusted Weighted Sum with Average of Row was employed as the prediction algorithm. In this evaluation, all the users' execution frequencies were employed by the prediction algorithm. In other words, the neighborhood size *k* was the number of users.

Figure 16 shows the calculated AREs of Experiment 2. In the graph, the vertical axis indicates ARE; the horizontal axis indicates each algorithm. The algorithm names are written with the abbreviations for convenience of description (for the abbreviations, see Appendix A). AREs of Word 2002 (the average AREs of 23 users having Word 2002 usage histories) are marked with a symbol "○"; AREs of PowerPoint 2002 (the average AREs of 21 users having PowerPoint 2002 usage histories) are marked with a symbol "△"; AREs of Excel 2002 (the average AREs of 20 users having Excel 2002 usage histories) are marked with a symbol "□"; the average AREs of three kinds of AREs are marked with a symbol "▬" , and each average ARE is written at the right of the symbol. As described in 6.2, smaller AREs (i.e. the symbols closer to bottom of the graph) indicate higher accuracy. Therefore, if a certain algorithm outperformed others, its average ARE must be lower than the other algorithm's average ARE. The algorithms are ranked in descending order of the average AREs from left to right below the horizontal axis, so that the algorithms closer to the right side of graph achieved better accuracy. In addition, the average AREs written with emphasized letters (italic underlined boldface) and emphasized with a vertical line indicate the paired Student's *t*-test statistically observed significant difference in contrast to the left algorithm's AREs, at the level of $p < 0.05$. Note that the letters are emphasized when the statistical difference is observed when using at least one of the three kinds of usage histories.

As a result, the evaluated algorithms were sorted by the average AREs as follows: User Average (Best), Binary Correlation, Correlation Coefficient, Rank Correlation, Magnitude Relation, and User Count (Worst). The paired Student's *t*-test statistically observed significant difference between the User Average and the Binary Correlation, the Correlation Coefficient and the Rank Correlation, and the Magnitude Relation and the User Count at the level of p < 0.05. For more detail of the evaluation results, see Appendix C.2.

Figure 17 shows the calculated NDPMs of Experiment 2. This graph is plotted in the same manner as the graph in Figure 16, where the NDPMs are marked instead of the AREs. As a result, the evaluated algorithms were sorted by the average NDPMs as: Rank Correlation (Best), Magnitude Relation, Binary Correlation, User Average, Correlation Coefficient, and User Count (Worst). The paired Student's *t*-test observed a statistic significance in difference among some algorithms, at the level of p < 0.05. The difference between the Correlation Coefficient and the User Count was observed using every usage history. The difference between the Binary Correlation and the User Average was observed using the Word 2002 and PowerPoint 2002 usage history. The difference between the Rank Correlation and the Magnitude Relation was observed using the Word 2002 usage history. The difference between the Magnitude Relation and the Binary Correlation was observed using the PowerPoint 2002 usage history. For more detail of

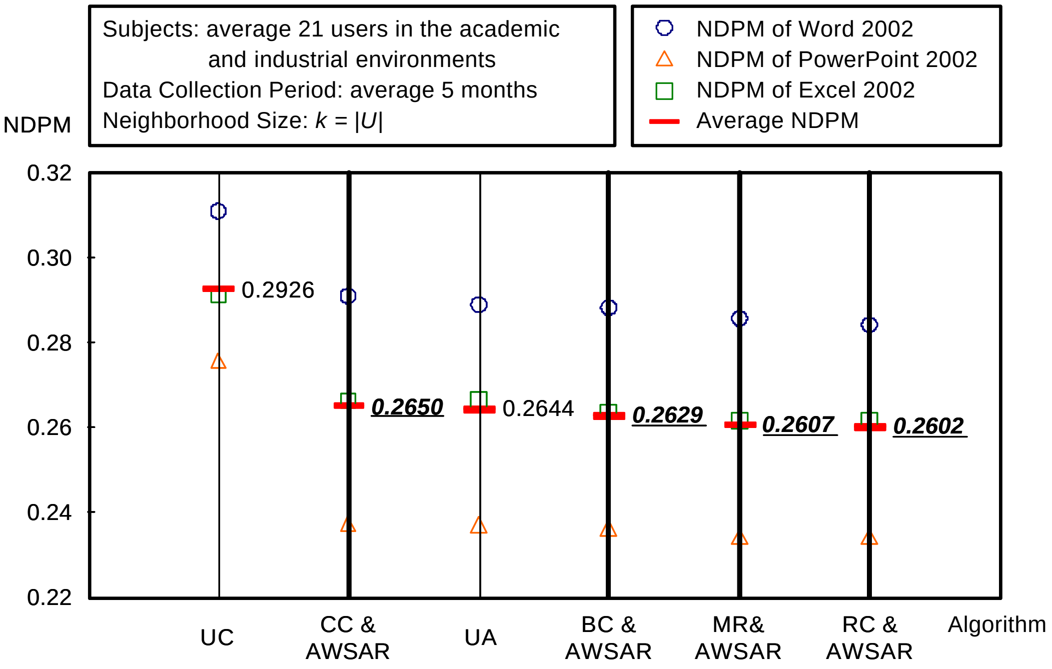


Figure 17. Result of Experiment 2: NDPM

the evaluation results, see Appendix C.3.

***Experiment 3*: History-based Evaluation of Accuracy of Various CF Algorithms**

In Experiment 3, usage histories were collected from 16 to 22 users of MS Word 2002, PowerPoint 2002, and Excel 2002 during approximately 9.5 months as shown in Table 3. In order to make clear which algorithms are suitable for recommending software functions, all simple prediction algorithms, all similarity computation algorithms except the Binary Correlation and the Magnitude Relation, and all prediction algorithms were evaluated. In Experiment 3, History-based Evaluation was executed with the CF algorithms composed by the combinations of the similarity computation and the prediction algorithms. In this evaluation, each neighborhood size from $k = 1$ up to $k = 10$, and $k = |U|$ (i.e. the number of users) was evaluated. Then, an appropriate neighborhood size, which achieved the highest accuracy, was chosen regarding each algorithm (Appendix B shows the evaluation results for selecting appropriate neighborhood sizes). The highest accuracy of each CF algorithm with the selected neighborhood size was compared to the accuracy of the other CF algorithms.

Figure 18 shows the calculated AREs of Experiment 3. This graph is plotted in the same manner as the graph of Figure 16. Note that this graph is rotated 90 degrees to the right and separated into left and the right parts, because the graph needs an extremely long horizontal axis in order to describe all evaluated CF algorithms. As a result, the evaluated algorithms were sorted by the average AREs, as shown in Figure 18. The paired Student's *t*-test statistically observed significant difference among some algorithms, at the level of $p < 0.05$. The difference between the combination of the Median of Squared Difference and the Adjusted Weighted Sum with Average of Row, and the Random Prediction was observed using every usage history. The difference between the Random Prediction and the User Count Prediction was observed using the Word 2002 and Excel 2002 usage histories. In addition, when using one particular kind of usage history, the differences were observed between the following CF algorithms:

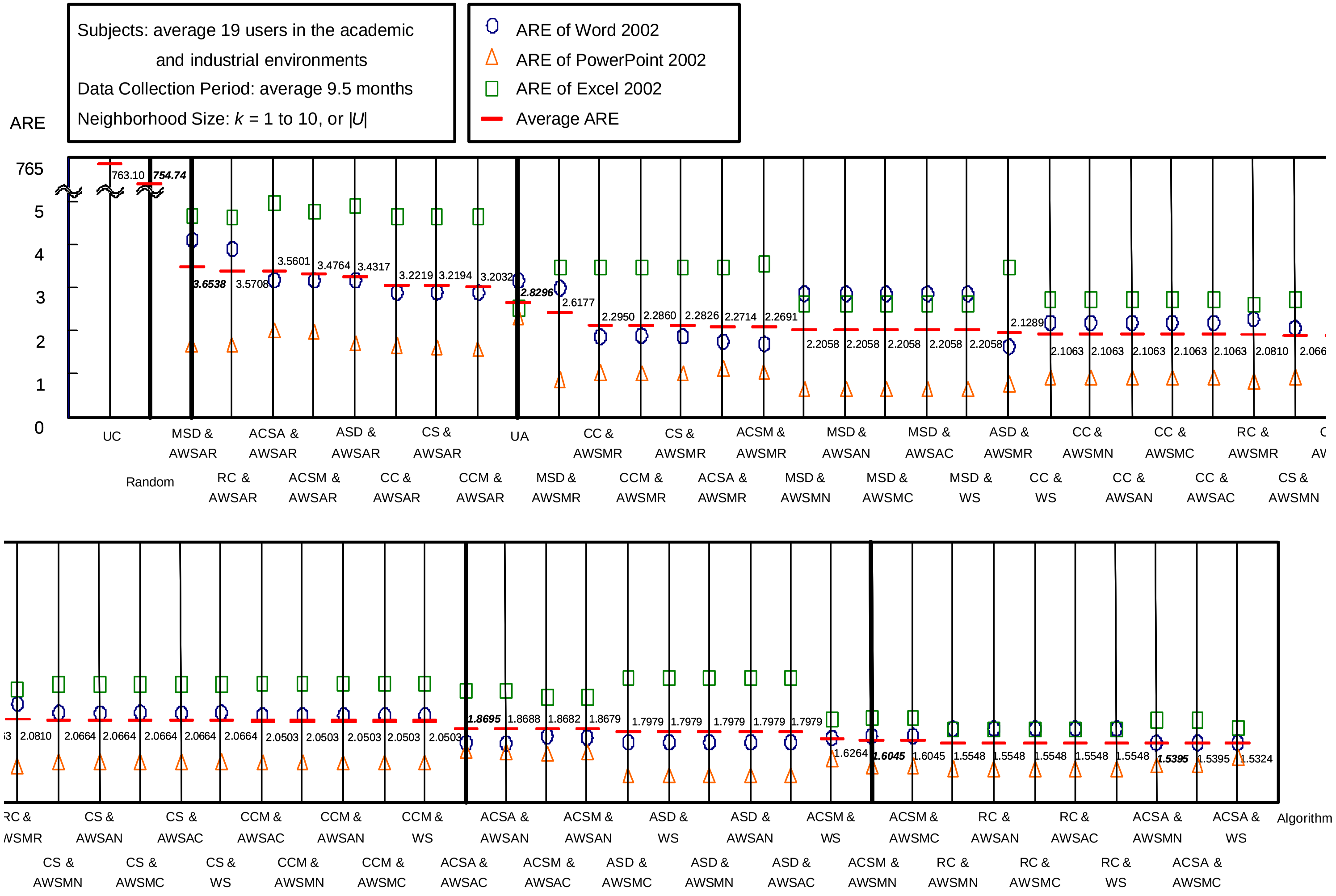


Figure 18. Result of Experiment 3: ARE

- Between the combination of the Adjusted Cosine Similarity with Average and the Adjusted Weighted Sum with Average of Column, and the combination of the Correlation Coefficient with Median and the Weighted Sum
- Between the combination of the Adjusted Cosine Similarity with Median and the Adjusted Weighted Sum with Median of Neighbors, and the combination of the Adjusted Cosine Similarity with Median and the Weighted Sum
- Between the User Average Prediction and the combination of the Correlation Coefficient with Median and the Adjusted Weighted Sum with Average of Row

For more detail, see Appendix C.4.

Figure 19 shows the calculated NDPMs of Experiment 3. This graph is plotted in the same manner as the graph of Figure 16, where the NDPMs are marked instead of the AREs. Note that this graph is rotated 90 degrees to the right and separated into left and the right parts as in Figure 18. As a result, the evaluated algorithms were sorted by the average NDPMs, as shown in Figure 19. The paired Student's *t*-test statistically observed significant difference among some algorithms, at the level of $p < 0.05$. The difference between the User Count Prediction and the Random Prediction was observed using every usage history. The difference between the combination of the Rank Correlation and the Adjusted Weighted Sum with Median of Neighbors, and the combination of the Median of Squared Difference and the Weighted Sum was observed using the PowerPoint 2002 and Excel 2002 usage histories. Likewise, the difference between the combination of the Rank Correlation and the Adjusted Weighted Sum with Average of Row, and the combination of the Adjusted Cosine Similarity with Median and the Adjusted Weighted Sum with Median of Row, was observed using the Word 2002 and Excel 2002 usage histories. In addition, when using one particular kind of usage history, the differences were observed between the following CF algorithms:

- Between the combination of the Rank Correlation and the Adjusted Weighted Sum with Median of Row, and the combination of the Median of Squared Difference and the Adjusted Weighted Sum with Median of Row
- Between the combination of the Adjusted Cosine Similarity with Median and the Adjusted Weighted Sum with Average of Neighbors, and the Adjusted Cosine

Similarity with Median and the Adjusted Weighted Sum with Median of Column

- Between the combination of the Cosine Similarity and the Adjusted Weighted Sum with Median of Row, and the combination of the Average Squared Difference and the Adjusted Weighted Sum with Average of Row
- Between the combination of the Correlation Coefficient and the Adjusted Weighted Sum with Average of Row, and the combination of the Rank Correlation and the Adjusted Weighted Sum with Average of Row
- Between the User Average and the combination of the Adjusted Cosine Similarity with Median and the Adjusted Weighted Sum with Average of Row

For more detail of the evaluation results, see Appendix C.5.

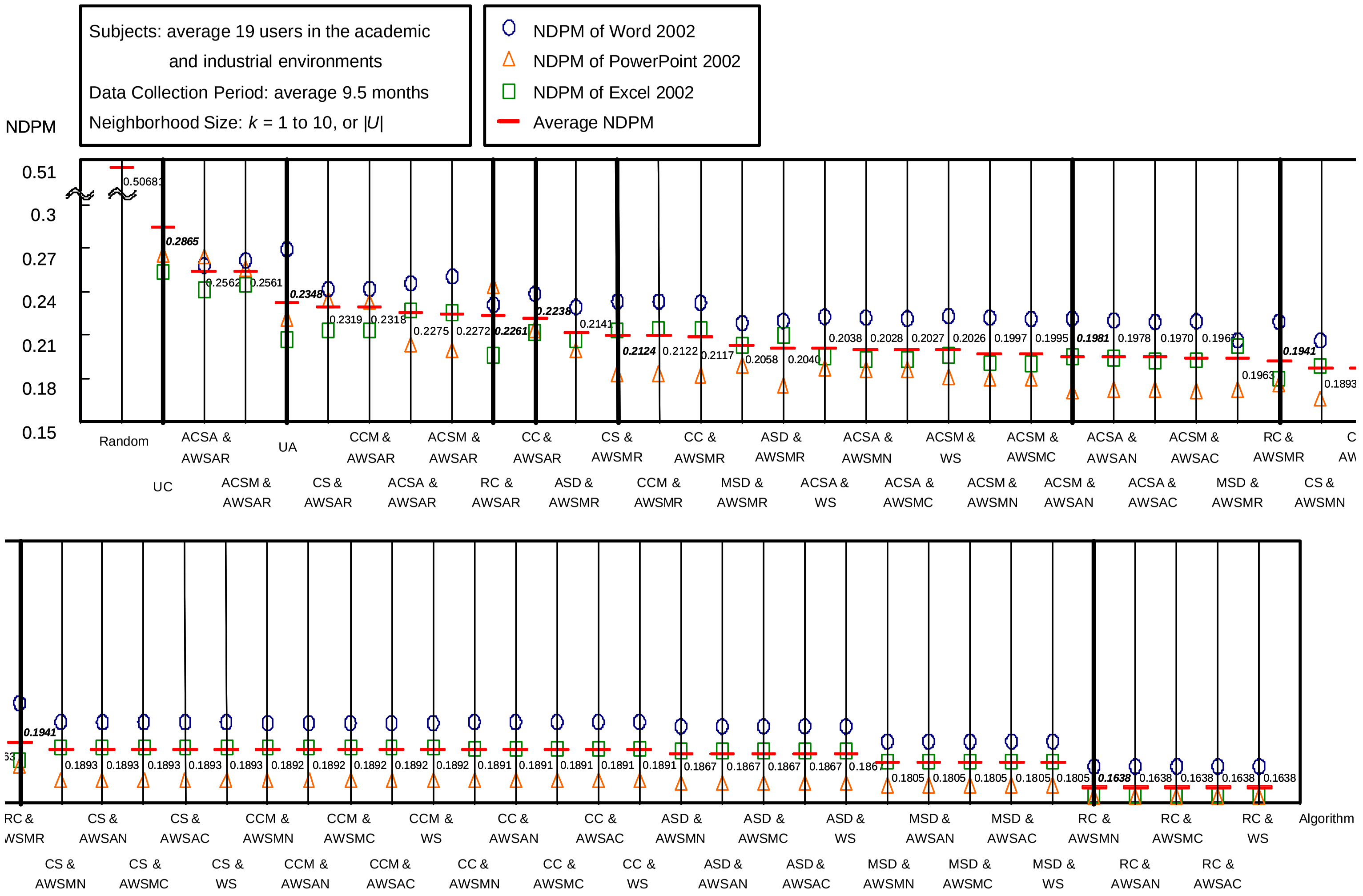


Figure 19. Result of Experiment 3: NDPM

7. Discussion

Experiments 1 and 2 were executed to evaluate the effectiveness of the proposed framework. Through Experiments 1 and 2, all CF algorithms and two simple prediction algorithms (the User Count and the User Average) achieved smaller NDPM than the Random Prediction Algorithm's theoretical NDPM (i.e., 0.5). Thus, these algorithms achieved at least the lowest accuracy the CF algorithm has to have. In focusing on the calculated NDPMs, the Rank Correlation, the Magnitude Relation, and the Binary Correlation consistently outperformed the User Average Algorithm. This suggests these similarity computation algorithms worked well, and could produce recommendations corresponding to each user's individual needs. These users' personal needs had to be taken into account because the users' objectives and the useful functions are different from each other as described in the section 6.1. In Experiment 2, the Correlation Coefficient did not achieve higher accuracy than the User Average Algorithm. This suggests that the correlation coefficients did not work enough in the similarity computation phase because the outliers gave bad influences to the evaluated similarities [42]. In fact, there were some frequently used functions (Clear, Save, Undo, or Paste, etc.) in the collected usage histories, as described in Table 4. These functions brought outliers which gave dominative impact in calculating similarities, so that the frequencies of the other functions were not reflected. On the other hand, the Rank Correlation, the Magnitude Relation, and the Binary Correlation were not influenced by the outliers because these algorithms did not directly use the execution frequencies.

In focusing on the calculated AREs, calculated AREs indicated inconsistent results with the NDPMs. The NDPM of every CF algorithm was larger than the User Average Algorithm. This suggests these algorithms could not predict the potential execution frequencies accurately. One of the reasons was that the neighborhood size was not appropriate. In Experiment 2, all users' execution frequencies were used in the prediction phase; i.e., "the number of users" was assigned as neighborhood size *k*. If this neighborhood size was not appropriate, then a larger number of relative errors were

Table 4. Frequently Used Functions of Microsoft Office Applications

(Examined from the Usage Histories shown in Table 3)

	Word 2002 (Frequency)	PowerPoint 2002 (Frequency)	Excel 2002 (Frequency)
1st	Clear->Clear All (43.21%)	Slide Show->Next (17.16%)	Edit->Paste (16.26%)
2nd	File->Save (17.30%)	Screen->Erase Pen (15.69%)	Clear->Contents (14.15%)
3rd	Edit->Undo (12.13%)	Screen->Stop (12.89%)	Edit->Copy (14.11%)
4th	Edit->Paste (4.76%)	Edit->Clear (11.31%)	File->Save (10.99%)
5th	Edit->Copy (3.80%)	Edit->Undo (6.40%)	Edit->Undo (9.29%)
6th	Edit->Redo (2.13%)	File->Save (5.46%)	Edit->Redo (3.63%)
7th	Edit->Cut (1.05%)	Slide Show->Help (5.25%)	Format->Format Cells (2.52%)
8th	File->Print (0.83%)	Edit->Paste (3.43%)	Edit->Cut (2.35%)
9th	Edit->Paste Special(0.82%)	Edit->Copy (3.03%)	Edit->Paste Special (1.98%)
10th	Formatting->Italics(0.77%)	Slide Show->Previous (2.59%)	Edit->Find (1.90%)
Total	(86.80%)	(83.21%)	(77.19%)

made although there is still need to improve the prediction accuracy of potential execution frequencies, the results of Experiment 1 and 2 suggest that the proposed framework can be effective for discovering useful the functions of HFAs.

Experiment 3 was executed to make clear which algorithms are suitable for recommending software functions. Through Experiment 3, most CF algorithms outperformed the three simple prediction algorithms (Random, User Count, and User Average). This result is not inconsistent with the result of Experiment 2. However, in Experiment 2, these algorithms made larger AREs than in the User Average Algorithm because an appropriate neighborhood size was selected for every CF algorithm in Experiment 3. Another important observation through Experiment 3 is that the following five prediction algorithms consistently outperformed the other prediction algorithms: Weighted Sum, Adjusted Weighted Sum with Average of Column, Adjusted Weighted Sum with Median of Column, Adjusted Weighted Sum with Average of Neighbors, and Adjusted Weighted Sum with Median of Neighbors. This result suggests these prediction algorithms were appropriate for the software function recommendation. One reason for this appropriateness is that these algorithms did not use the average or the median of execution frequencies regarding each user, in contrast to the other two prediction algorithms (Adjusted Weighted Sum with Average of Row and Adjusted Weighted Sum with Median of Row). The average and the median of execution frequencies regarding each user gave bad influences to the accuracy, so that these two algorithms created a lower accuracy. As shown in Table 4, there were some extremely large execution frequencies regarding each user, while other execution frequencies were relatively smaller. If the prediction algorithm employed the average of execution frequencies, the small execution frequencies were given bad influences by the average affected by the outliers. As a result, the prediction algorithm overestimated these small frequencies. On the other hand, if the prediction algorithm employed the median of execution frequencies, the large execution frequencies were given bad influences by the median which took a much smaller value. As a result, the prediction algorithm underestimated these large frequencies. In the context of software function recommendation, the system should avoid using the average and the median of execution frequencies regarding each user because these are liable to cause problems.

In focusing on the calculated NDPMs, most CF algorithms outperformed three simple prediction algorithms. In particular, the combination of the Rank Correlation Algorithm and the above “appropriate” five prediction algorithms (Weighted Sum, Adjusted Weighted Sum with Average of Column, Adjusted Weighted Sum with Median of Column, Adjusted Weighted Sum with Average of Neighbors, and Adjusted Weighted Sum with Median of Neighbors) achieved the smallest NDPM. The paired Student’s *t*-test statistically observed significant difference between these CF algorithms and the other algorithms, at the level of $p < 0.05$. In addition, the Rank Correlation consistently outperformed the other algorithms in Experiments 1, 2, and 3. This suggests that the Rank Correlation is the appropriate similarity computation algorithm.

In focusing on the calculated AREs, most CF algorithms outperformed three simple prediction algorithms. In particular, four similarity computation algorithms

(Adjusted Cosine Similarity with Average, Adjusted Cosine Similarity with Median, Rank Correlation, and Average Squared Difference) achieved the smallest NDPM; however, the most accurate algorithm was different, depending on the employed usage histories. For example, in employing Word 2002 usage histories, the Adjusted Cosine Similarity with Average achieved the smallest ARE; in employing Excel 2002 usage histories, the Rank Correlation achieved the smallest ARE. In addition, the paired Student's *t*-test did not reveal a statistically significant difference among the Adjusted Cosine Similarity with Average and the Rank Correlation, at the level of $p < 0.05$. This result suggests that all of these algorithms can be the appropriate prediction algorithm.

In context of the software function recommendation, the following CF algorithm can be considered suitable from a comprehensive evaluation of Experiment 3. The Rank Correlation is a suitable similarity computation algorithm because this algorithm achieved the smallest NDPMs in every usage history, and the smallest ARE using Excel 2002 usage histories. The Weighted Sum is a suitable prediction algorithm because this algorithm consistently achieved the smallest NDPMs and AREs in every usage history. Moreover, the Weighted Sum is simpler than the other appropriate algorithms (Adjusted Weighted Sum with Average of Column, Adjusted Weighted Sum with Median of Column, Adjusted Weighted Sum with Average of Neighbors, and Adjusted Weighted Sum with Median of Neighbors), so that its throughput (processing capacity) is larger than the other algorithms. Although there is a possibility for improving the recommendation accuracy, the combination of the Rank Correlation and the Weighted Sum has enough accuracy to generate software function recommendations for practical use.

8. Related Work

8.1 Usage History Collection

In order to achieve richer usability of the software, systems that collect software usage histories have been developed [18]. According to people who use mainly the collected histories, these conventional systems can be classified into the following two types:

- *System for software developers*: This type of system helps developers improve the usability of the software they are developing.
- *System for software users*: This type of system helps users enhance the usability of the software they are using.

Finlay and Harrison [9] proposed a system for software developers. Their system collects software usage histories to improve the usability of the software's user interface. Improvement can be achieved as follows. First, the developer of the software records an ideal usage history by operating the user interface, which is a target of the improvement. Next, the system collects some actual usage histories from people who actually use the targeted software. The system compares ideal usage history and actual usage histories to detect different points between them. In these instances, it is possible that users could have operated the user interface with erroneous usage. Developers can improve the detected bad parts so that the users will not give erroneous usages. Hilbert and Redmiles [16] proposed a large-scale collection system of usage histories for software developers. They took a similar approach to the system proposed by Finlay and Harrison, which collects usage histories to improve software usability. However, unlike Finlay and Harrison, Hilbert and Redmiles' system collects usage histories from real users of the software. If software users make different usage histories than the ideal usage histories, the system will sends the users' usage histories to the developers. The developers can collect usage histories from a large amount of the users and can use such histories to improve user interfaces.

In contrast to the above systems for software developers, this dissertation proposes a framework for systems for the users; however, collected usage histories with the systems based on the proposed framework are extremely important for improving the usability of the software because the developers can recognize frequently used functions and unused functions. In developing the next version of the software, the developers should enhance the accessibility of these frequently used functions (e.g. by assigning shortcut keys). In addition, the developers should consider the possibility of discarding unused functions. Or, if these functions are still considered important, the developers should consider the reasons why these were not used.

Yano et al. [55] proposed Sharlok, which is a system for software users. Sharlok collects software usage histories to provide opportunities to begin collaboration among users. Sharlok automatically collects each user's usage history with the background process. Each user can see the other users' usage histories anytime, and therefore each user will be aware of the other users' operations. Due to this openness of histories the users will have the opportunity to ask or discuss about the other users' behavior. Morisaki et al. [28], [29] proposed a system for users to support software function discovery. They took a similar approach with the Sharlok system for discovering new software functions. First, the system collects usage histories from users via the network. The system summarizes the collected usage histories by extracting information about the names of the executed functions, the time of the executions, the people who executed the functions and frequencies of the executions. Next, the system displays to each user a summary of the other users' usage histories. Each user is therefore aware of what functions have been used by other users.

The above systems for software users collect usage histories from the users to make users aware of the each others' behaviors. Software function recommender systems based on the proposed framework are similar in approach to the above systems; however, the above systems are information sharing systems rather than recommender systems because these work as a bridge to the users' behaviors and knowledge. The above systems just bring each user the other users' usage histories. Users of these systems have to choose preferable information from the displayed usage histories by themselves. However, when the systems are operated by a small number of users in small closed community, it is possible that these systems cannot be used as open systems to serve a large number of users because the users are given a large amount of information at a time by these systems. On the other hand, the systems based on the proposed framework generate personalized information to individual users using usage histories, and also recommend information. The proposed framework can be operated as an open system because the users are given a small amount of individual information made especially for each user.

8.2 User Assistant for High-Functionality Applications

In order to make users aware of useful previously unknown software functions, an HFA *user assistant* that suggests useful functions has been developed. According to a method for predicting useful unaware functions, these conventional user assistants can be classified into the following three types:

- *User assistant without prediction method*: This type of assistant does not predict, but just displays certain functions pre-specified by developers of HFA. This type of assistant can be implemented as a stand-alone. The accuracy of the prediction is liable to be poor.
- *User assistant with a method based on user stereotypes*: This type of assistant predicts useful functions based on user stereotypes (the so-called *Standard User Models*) predefined by developers of HFA. This type of assistant can be implemented as a stand-alone. The accuracy of the prediction depends on the defined user stereotypes.
- *User assistant with a method based on CF*: This type of assistant predicts useful functions based on CF. This type of assistant cannot be implemented as a stand-alone. The accuracy of the prediction depends on the data collected from the users.

Owen [37] proposed the *DYK (Did You Know)* system and Horvitz [21] mentioned the idea of a *Tip of the Day*. These are user assistants that have no prediction method. These assistants display a previously unknown function randomly when the user launches the HFA. In contrast to an online help assistant or an information retrieval assistant, the user does not have to choose a query to get the functions. If a chance should serve, however, the user can be made aware of useful previously unknown functions. In theory, NDPM of these assistants' recommendation (i.e. randomly displayed function) must be 0.5. Although the accuracy of the prediction is not high, Owen and Horvitz provided the basic idea to support software function discovery. On the other hand, the recommender systems based on the proposed framework can provide more accurate recommendations which achieves NDPMs smaller than 0.5.

Fischer et al. [10] proposed *Active Help*; Winkels et al. proposed *EURO-HELP* [54]; and Horvitz et al. [22] proposed *Microsoft Office Assistant*. These assistants collect usage histories from each HFA user, and construct a *user model* from the collected usage histories. A user model is an inner data representation of the user's behaviors [12]. These assistants compare the constructed user model to the user stereotypes to find a similar stereotype. User stereotypes (the so-called *Standard User Models*) are predefined by the HFA developers and represent typical users' behaviors and objectives [38]. Generally, the user stereotypes are defined by observing the behaviors of the user representatives and giving them questionnaires [3]. The assistants infer the user's objective based on similar stereotypes with the current user's model. Then, the assistants recommend the functions which are helpful to complete the objective. These assistants are very effective when the developers can define accurate and numerous kinds of stereotypes; however, if the number of functions of the HFA is extremely large, defining accurate and numerous kinds of stereotypes is difficult because the number of potential stereotypes is liable to be extremely large [11]. On the other hand, the recommender systems based on the proposed framework do not use any user stereotypes to predict useful previously unknown functions. Thus, these recommender systems can provide accurate recommendations regardless of the number of functions of the HFA.

To date, no user assistant with a method based on CF has been developed. This dissertation proposes a novel framework for providing systems which are classified as this type of user assistants. As previously discussed in this dissertation, one of the big advantages of this type of user assistants is that recommendations without the user stereotypes can be made. The developers do not have to spend time and effort to define the stereotypes, so they can cut cost. In addition, developers do not have to challenge many difficult issues [11] related to the accuracy of the user stereotypes. One of the different points of this type of user assistants from the other type of assistants is that in this case, the users' computers need to be connected to the server via a network, while the other type of assistants can make the users' computers stand-alone. However, this different point is not a disadvantage because, in using HFA, the users' computers must be connected anyway to use other features, such as product activation or automated software deployment [51].

9. Conclusion

This dissertation proposed a framework for software function recommendation based on CF to allow users to discover useful, but previously unknown functions efficiently. The proposed framework included a concept design and the architectural design of system which makes software function recommendations. In order to produce accurate recommendations to a target HFA user, this dissertation described three simple prediction algorithms (lacking similarity computation), ten similarity computation algorithms including two new algorithms, and seven prediction algorithms. Prediction accuracies of these CF algorithms were empirically evaluated by using usage histories collected from actual Microsoft Office Application users. The result of the experiments suggested that the proposed framework can be effective for discovering useful functions of HFAs. In particular, a combination of the Rank Correlation and the Weighted Sum is a suitable algorithm in the context of software function recommendation because this combination has enough accuracy to generate software function recommendations for practical use.

Other CF algorithms such as Item-based CF algorithms [46] and Model-based CF algorithms [4] exist. Furthermore, some enhancement methods of the CF algorithm such as Inverse User Frequency and Case Amplification [4] are available. Investigating the effectiveness of these other CF algorithms and the enhancement methods in context of software function recommendation is a project for the future. Currently, Item-based CF algorithms are used on various Websites such as Amazon.com and CDNow [47]. These Item-based algorithms have a high likelihood to be able to improve the accuracy of the recommendations. Nonetheless, several challenging issues in actually implementing the system based on the proposed framework still exist. One of these issues is the scalability of the system. Typically, a CF-based recommender system has thousands of users, and should provide recommendations quickly as a reply to the users' requests. In order to achieve accurate and faster services, implementation efforts in an open source project, *NAIST Collaborative Filtering Engines*, will be ongoing [34].

Traditional CF systems have been employed for accelerating knowledge sharing in the entertainment environments. Today, some working environments such as software engineering need to share knowledge in organizations such as companies. For instance, Ohsugi et al. [33] proposed an estimation method of software development effort using CF-based knowledge sharing. CF has the possibility to conquer strong obstacles in today's working environments. When researchers begin to research these methods, this dissertation will provide one of the early experiences in applying CF to the issues in the working environments.

Acknowledgements

I would especially like to thank my supervisor Professor Ken-ichi Matsumoto for his continuous support, encouragement, and guidance for this work.

I am also very grateful to the members of my dissertation review committee: Professor Masaki Koyama, Professor Hiroyuki Seki, and Associate Professor Hajimu Iida for their invaluable comments and helpful criticism of this dissertation.

I also would like to thank Dr. Dee A. Worman of Edit Science, Inc. She edited this dissertation, and gave me valuable comments and suggestions to improve the quality of this dissertation.

Many of the courses that I have taken during my graduate career have been helpful in preparing this dissertation. I would especially like to acknowledge the guidance of Assistant Professor Akito Monden and Assistant Professor Masahide Nakamura.

I would like to express my gratitude to President Koji Torii of Nara Institute of Science and Technology, Professor Katsuro Inoue of Osaka University, Professor Kumiyo Nakakoji of University of Tokyo, Professor Hideo Kudo of Osaka Seikei University, Associate Professor Yasuhiro Takemura of Osaka University of Arts, Associate Professor Kazuyuki Shima of Hiroshima City University and Dr. Kazuaki Yamada of the University of Tokyo. They gave me valuable suggestions and provided stimulating discussions throughout my graduate career.

I would also like to acknowledge the valuable comments of Dr. Makoto Sakai of Software Research Associates Key Technology Laboratory, Inc., Mr. Koji Kondo and Mr. Yasushi Tanaka of Sony Corporation. I also acknowledge the helpful support of Dr. Toshihiro Kamiya of PRESTO Japan Science and Technology Corporation, Dr. Shuji Morisaki of Internet Initiative Japan, Inc. and Dr. Masatake Yamato of Red Hat, Inc.

Special thanks are due to many friends in the Graduate School of Information Science, Nara Institute of Science and Technology. I especially would like to thank Ms. Junko Inui, Mr. Yuichiro Kanzaki, Mr. Masateru Tsunoda, and Mr. Takeshi Kakimoto who gave me invaluable assistance throughout my graduate career.

Finally, special thanks go to my grandfather Eitaro Ohsugi and Tsunesaburo Kawakami, my grandmother Yoshiko Ohsugi and Yoshie Kawakami, my father Eiichi Ohsugi, my mother Hatsuko Ohsugi, and my sister Chinatsu Ohsugi. Without their help, I would not have finished this dissertation. They have given me love and power to live any time. All of my work is built upon my family's contributions.

References

[1] Balabanovic, M., and Shoham, Y., “Fab: Content-based Collaborative Recommendation,” *Communications of the ACM*, Vol.40, No.3, pp.66-72, 1997.

[2] Balabanovic, M., “An Adaptive Web Page Recommendation Service,” *Proceedings of the First International Conference on Autonomous Agents (Agents’97)*, pp. 378-385, 1997.

[3] Brajnik, G., and Tasso, C., “A Shell for Developing Non-monotonic User Modeling Systems. International Journal of Human-Computer Studies,” Vol.40, pp.31-62, 1994.

[4] Breese, J. S., Heckerman, D., and Kadie, C., “Empirical Analysis of Predictive Algorithms for Collaborative Filtering,” *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp.43-52, 1998.

[5] Carroll, J. M., and Rosson, M. B., “Paradox of the Active User,” Interfacing Thought: Cognitive Aspects of Human-Computer Interaction, MIT Press, 1987.

[6] Conte, S. D., Dunsmore, H. E., and Shen, V. Y., “Software Engineering Metrics and Models,” The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1986.

[7] Cypher, A., “Eager: Programming Repetitive Tasks by Example,” *Proceedings of the CHI’91 Conference on Human Factors in Computing Systems*, pp.33-39, 1991.

[8] Efron, B., “The Jackknife, the Bootstrap, and Other Resampling Plans,” Philadelphia: SIAM, 1982.

[9] Finlay, J., and Harrison, M., “Pattern Recognition and Interaction Models,” *Proceedings of the 3rd IFIP International Conference on Human Computer Interaction (INTERACT’90)*, pp.149-154, 1990.

[10] Fischer, G., Lemke, A. C., and Schwab, T., “Knowledge-based Help Systems,” *Proceedings of the CHI’85 Conference on Human Factors in Computing Systems*, pp.161-167, 1985.

[11] Fischer, G., “User Modeling: the Long and Winding Road,” Proceedings of the User

Modeling Conference ’99, pp.349-355, 1999.

[12] Fischer, G., “User Modeling in Human-Computer Interaction,” *User Modeling and User-Adapted Interaction (UMUAI)*, Vol.11, No.1/2, pp 65-86, 2001.

[13] Good, N., Schafer, J., Konstan, J., Borchers, A., Sarwar, B., Herlocker, J., and Riedl, J., “Combining Collaborative Filtering with Personal Agents for Better Recommendations,” *Proceedings of the 6th National Conference on Artificial Intelligence (AAAI-99)*, pp.439-446, 1999.

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

[15] Hawkins, S., He, H., Williams, G., and Baxter, R., “Outlier Detection Using Replicator Neural Networks,” *Proceedings of the 4th International Conference, Data Warehousing and Knowledge Discovery (DaWaK2002)*, pp.170-180, 2002.

[16] Hilbert D. M., and Redmiles, D. F., “An Approach to Large-Scale Collection of Application Usage Data Over the Internet,” *Proceedings of 20th International Conference on Software Engineering (ICSE’98)*, pp.136-145, 1998.

[17] Hilbert D. M., and Redmiles, D. F., “Collecting Usage Data and User Feedback on a Large Scale to Inform Software Development,” *Technical Report UCI-ICS-99-41*, Department of Information and Computer Science, University of California, Irvine, 1999.

[18] Hilbert, D. M., and Redmiles, D. F., “Extracting Usability Information from User Interface Events,” *ACM Computing Surveys*, Vol.32, No.4, pp.384-421, 2000.

[19] Hill, W., Stead, L., Rosenstein, M., and Furnas, G., “Recommending and Evaluating Choices in a Virtual Community of Use,” *Proceedings of the 1995 Conference on Human Factors in Computing Systems (CHI’95),* pp.194-201, 1995.

[20] Hogg, R.V., and Craig, A.T., “Introduction to Mathematical Statistics,” 5th edition, New York: Macmillan, 1995.

[21] Horvitz, E., “Agents with Beliefs: Reflections on Bayesian Methods for User Modeling,” *Proceedings of the 16th International Conference on User Modeling*

(UM'97), pp.441-442, 1997.

[22] Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K., "The Lumière Project: Bayesian User Modeling for Inferring the Goals and Needs of Software Users," *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp.256-265, 1998.

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

[24] Kautz, H., Selman, B., and Shah, M., "Referral Web: Combining Social Networks and Collaborative Filtering," *Communications of the ACM*, Vol.40, No.3, pp.63-65, 1997.

[25] Khoshgoftaar, T. M., Munson, J. C., Bhattacharya, B. B., and Richardson, G D. "Predictive Modeling Techniques of Software Quality from Software Measures," *IEEE Transaction on Software Engineering,* Vol.18, Bo.1, pp.979-987, 1992.

[26] Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L., and Riedl, J., "Grouplens: Applying Collaborative Filtering to Usenet News," *Communications of the ACM*, Vol.40 pp.77–87, 1997.

[27] Mathe, N., and Chen, J., "A User-Centered Approach to Adaptive Hypertext based on an Information Relevance Model," *Proceedings of the 4th International Conference on User Modeling (UM'94)*, pp. 107-114, 1994.

[28] Morisaki, S., Monden, A., Matsumoto, K., Inoue, K., and Torii, K., "Sharing Usage Knowledge for Application Software Using Function Execution History," *IPSJ Journal*, Vol.41, No.10, pp.2770-2781, 2000 (in Japanese).

[29] Morisaki, S., Shiraishi, Y., Yamato, M., Monden, A., Matsumoto, K., and Torii, K., "A Support System for Software Function Discovery Using Histories of Function Executions," *Transactions of the Institute of Electronics, Information and Communication Engineers, D-I*, Vol.J84-D-I, No.6, pp.755-767, 2001 (in Japanese).

[30] Ohsugi, N., Monden, A., and Matsumoto, K., "A Recommendation System for Software Function Discovery," *Proceedings of the 9th Asia Pacific Software Engineering Conference (APSEC2002),* pp.248-257, 2002.

[31] Ohsugi, N., Monden, A., and Morisaki, S., “Collaborative Filtering Approach for Software Function Discovery,” *Proceedings of the 2002 International Symposium on Empirical Software Engineering (ISESE2002)*, Vol.2, pp.29-30, 2002.

[32] Ohsugi, N., Monden, A., Morisaki, S., and Matsumoto, K., “Software Function Recommender System Based on Collaborative Filtering,” *Journal of Information Processing Society of Japan*, Vol.45, No.1, pp.267-278, January 2004 (in Japanese).

[33] Ohsugi, N., Tsunoda, M., Monden, A., and Matsumoto, K., “Effort Estimation Based on Collaborative Filtering,” *Proceedings of the 5th International Conference on Product Focused Software Process Improvement (PROFES2004)*, pp.274-286, 2004.

[34] Ohsugi, N., “Naist Collaborative Filtering Engines”, Open Source Development Network Japan, https://sourceforge.jp/projects/ncfe/, 2004.

[35] O’Sullivan, D., Wilson, D., and Smyth, B., “Improving Case-Based Recommendation: A Collaborative Filtering Approach,” *Proceedings of the 6th European Conference on Case Based Reasoning (ECCBR2002)*, pp.278-291, 2002.

[36] O’Sullivan D, Smyth, B., Wilson D. C., Mcdonald K., and Smeaton, A., “Improving the Quality of the Personalized Electronic Program Guide,” *User Modeling and User-Adapted Interaction (UMUAI)*, Vol.14, No.1, pp.5-36, 2004.

[37] Owen, D., “Answers First, Then Questions,” *User-Centered System Design, New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Inc., pp.361-375, 1986.

[38] Paliouras, G., Karkaletsis, V., Papatheodorou, C., and Spyropoulos, C. D., “Exploiting Learning Techniques for the Acquisition of User Stereotypes and Communities,” *Proceedings of the 17th International Conference on User Modeling (UM’99)*, pp.169-178, 1999.

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

[40] Pretschner, A., and Gauch, S., “Ontology Based Personalized Search,” *Proceedings of the 11th IEEE International Conference on Tools with Artificial Intelligence*, pp.391-398, 1999.

[41] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J., “GroupLens: An Open Architecture for Collaborative Filtering of Netnews,” *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work (CSCW’94)* pp.175-186, 1994.

[42] Robert G, and Dawson, T.B., “Basic & Clinical Biostatistics,” McGraw-Hill, 3rd Edition, 2001.

[43] Rucker, J., and Marcos, J. P., “Siteseer: Personalized Navigation for the Web,” *Communications of the ACM*, Vol.40, No.3, pp.73-75, 1997.

[44] Salton, G., and McGill, M., “Introduction to Modern Information Retrieval,” McGraw-Hill, New York, 1983.

[45] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., “Analysis of Recommendation Algorithms for E-Commerce,” *Proceedings of the ACM Conference on ECommerce (EC00)*, pp.158-167, 2000.

[46] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J., “Item-based Collaborative Filtering Recommendation Algorithms,” *Proceedings of the 10th International World Wide Web Conference (WWW10)*, pp.285-295, 2001.

[47] Schaefer, J. B., Konstan, J., and Riedl, J., “Recommender Systems in E-Commerce,” *Proceedings of the 1st ACM Conference on Electronic Commerce*, pp.158-166, 1999.

[48] Shardanand, U., and Maes, P., “Social Information Filtering: Algorithms for Automating ‘Word of Mouth’,” *Proceedings of the CHI’95 Conference on Human Factors in Computing Systems*, pp.210-217, 1995.

[49] Smyth, B., and Cotter, P., “Personalized Electronic Programme Guides,” *Artificial Intelligence Magazine*, Vol.21, 2001.

[50] Strike, K., El Eman, K., and Madhavji, N., “Software Cost Estimation with Incomplete Data,” *IEEE Transaction on Software Engineering*, Vol.27, No.10,

pp.890-908, 2001.

[51] Tallman, O. H., “Project Gabriel: Automated Software Deployment in a Large Commercial Network,” *Digital Technical Journal*, Vol.7, No.2, pp.56-70, 1995.

[52] Terveen, L., Hill, W., McDonald, D., and Creter, J., “PHOAKS: A System for Sharing Recommendations,” *Communications of the ACM*, Vol.40, No.3, pp.59-62, 1997.

[53] Wilensky, R., Arens, Y., and Chin, D., “Talking to UNIX in English: An Overview of UC,” *Communications of the ACM*, Vol.27, No.6, pp.574-593, 1984.

[54] Winkels, R., Breuker, J., and den Haan, N., “Principles and Practice of Knowledge Representation in EURO-HELP,” *Proceedings of the International Conference on the Learning Sciences*, pp.442-448, 1991.

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

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

[57] Ye, Y., Fischer, G., “Information Delivery in Support of Learning Reusable Software Components on Demand”, *Proceedings of 2002 International Conference on Intelligent User Interface (IUI’02)*, pp.159-166, 2002.

Appendix

A. Abbreviations of CF Algorithm Names

In the figures and the tables for summarizing evaluation results, each algorithm name is described with the following abbreviations for convenience of description.

Table 5. Abbreviations of CF Algorithm Names

Algorithm Type	Abbreviation	Formal Nomenclature
Simple Prediction Algorithms	**Random**	Random Algorithm
	UC	User Count Algorithm
	UA	User Average Algorithm
Similarity Computation Algorithms	**CS**	Cosine Similarity
	ACSA	Adjusted Cosine Similarity with Average
	ACSM	Adjusted Cosine Similarity with Median
	CC	Correlation Coefficient
	CCM	Correlation Coefficient with Median
	BCS	Binary Cosine Similarity
	RC	Rank Correlation
	MR	Magnitude Relation
	ASD	Average Squared Difference
	MSD	Median of Squared Difference
	CS	Cosine Similarity
	ACSA	Adjusted Cosine Similarity with Average
Prediction Algorithms	**WS**	Weighted Sum
	AWSAC	Adjusted Weighted Sum with Average of Column
	AWSMC	Adjusted Weighted Sum with Median of Column
	AWSAN	Adjusted Weighted Sum with Average of Neighbors
	AWSMN	Adjusted Weighted Sum with Median of Neighbors
	AWSAR	Adjusted Weighted Sum with Average of Row
	AWSMR	Adjusted Weighted Sum with Median of Row

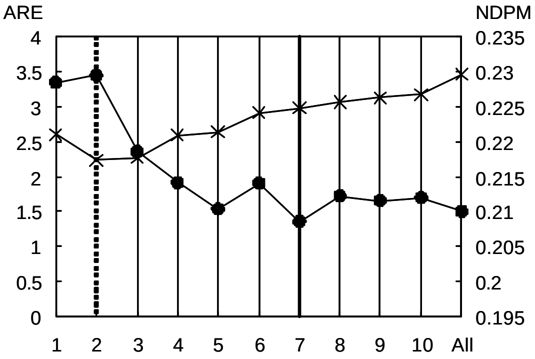
B. Evaluation Results for Selecting Appropriate Neighborhood Sizes

B.1 Evaluation Results with Usage Histories of MS Word 2002

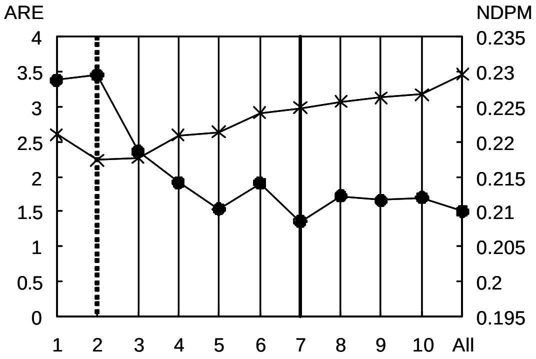
The following graphs show evaluation results for selecting appropriate neighborhood size for each CF algorithm when the evaluation was executed with the usage histories of MS Word 2002. Each graph shows the evaluation results of each CF algorithm composed by certain combinations of a similarity computation algorithm and a prediction algorithm. Recommendation was generated by using each CF algorithm with 1 to 10 nearest neighbors and all users. In other words, neighborhood size was moved from 1 up to 10 and |*U*| (the number of users). Next, the AREs and NDPMs were calculated with the generated recommendations. In the graph, the vertical axis indicates ARE and NDPM, and their scales are written in the left side and the right side of the graph, respectively. The horizontal axis indicates each neighborhood size of 1 to 10 or All (i.e. the number of users |*U*|). Each ARE is marked with a symbol "×", while each NDPM is marked with a symbol "●", at the corresponding neighborhood size. As described in 6.2, smaller ARE and smaller NDPM (i.e. the symbols closer to the bottom of the graph) indicate higher accuracy. Appropriate neighborhood sizes which achieving the highest accuracy, were chosen regarding each CF algorithm for ARE and NDPM, respectively. In the graph, the broken vertical line and the heavy vertical line indicate selected neighborhood sizes for ARE and NDPM, respectively.

┆ Selected Neighborhood Size for ARE | Selected Neighborhood Size for NDPM

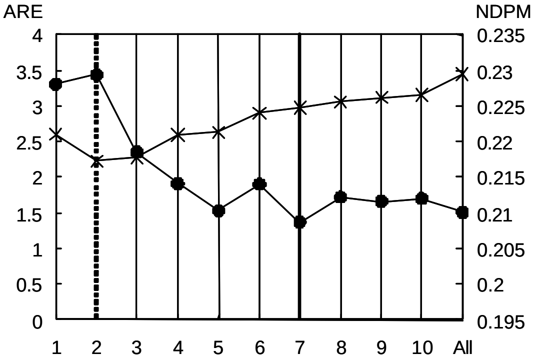
—✕— ARE
—●— NDPM



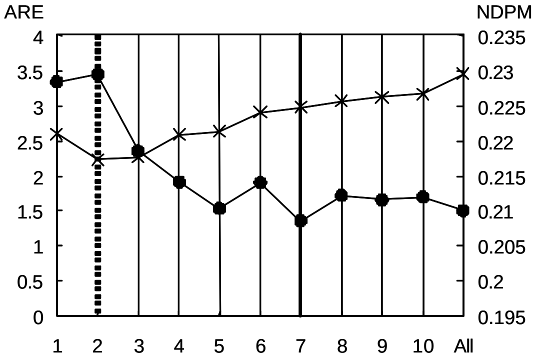
B.1. 1. CS & WS



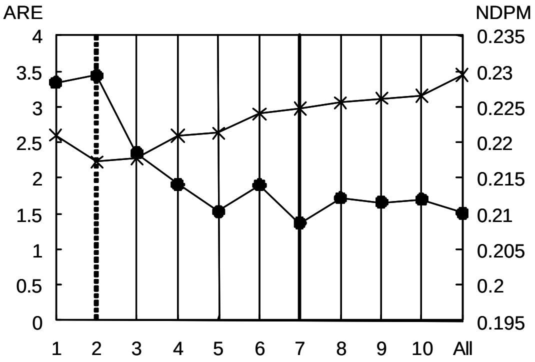
B.1. 2. CS & AWSAC



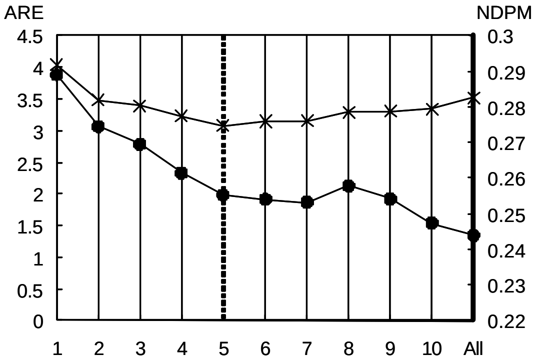
B.1. 3. CS & AWSMC



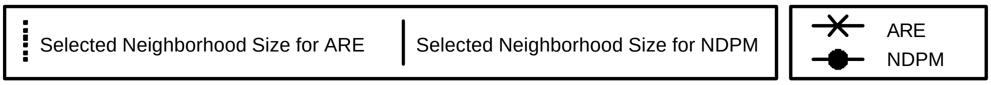
B.1. 4. CS & AWSAN

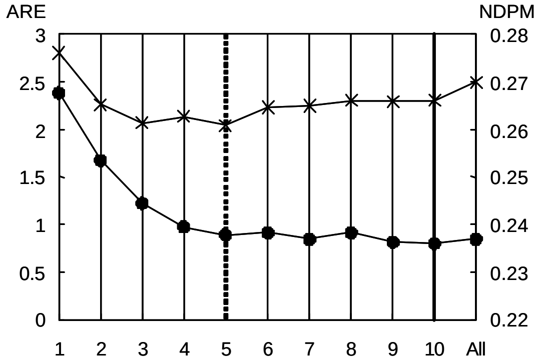


B.1. 5. CS & AWSMN

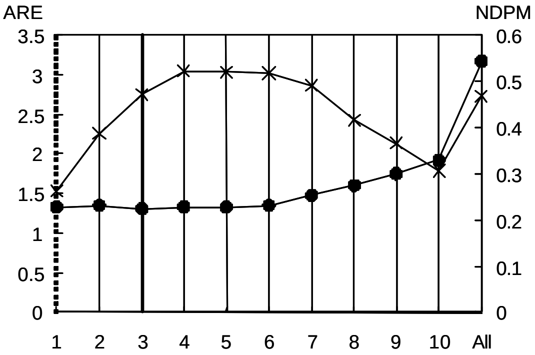


B.1. 6. CS & AWSAR

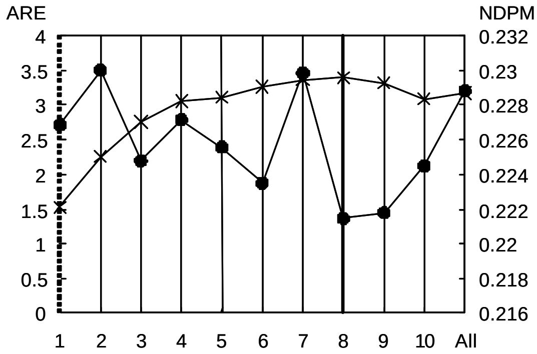




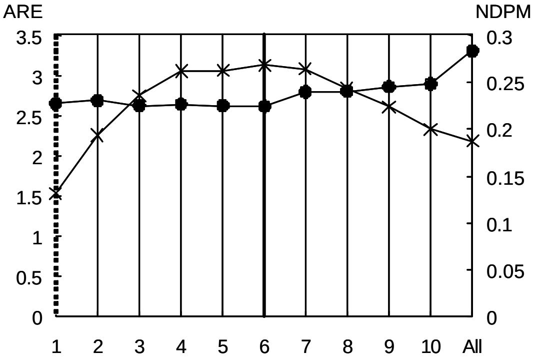
B.1. 7. CS & AWSMR



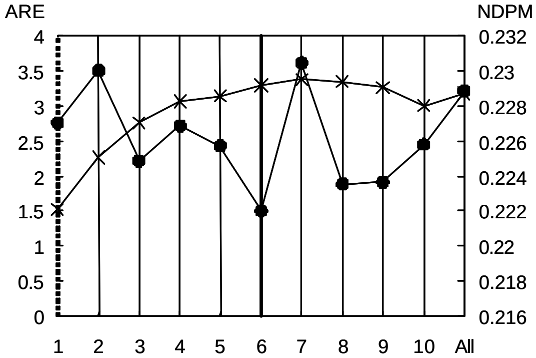
B.1. 8. ACSA & WS



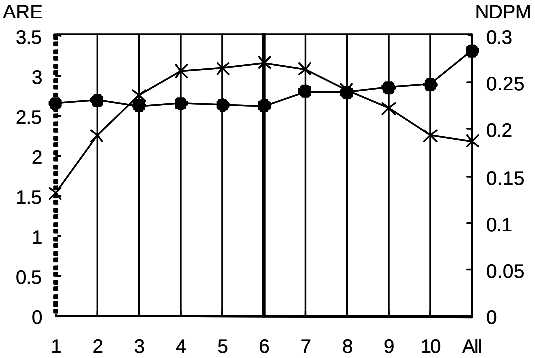
B.1. 9. ACSA & AWSAC



B.1. 10. ACSA & AWSMC

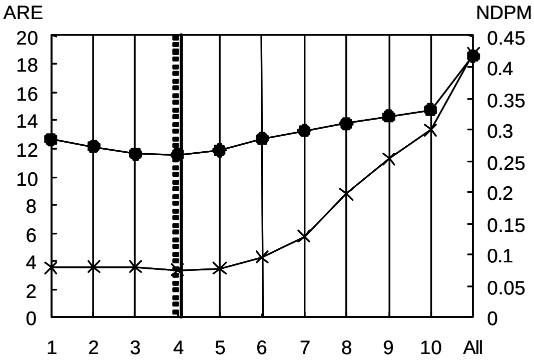


B.1. 11. ACSA & AWSAN

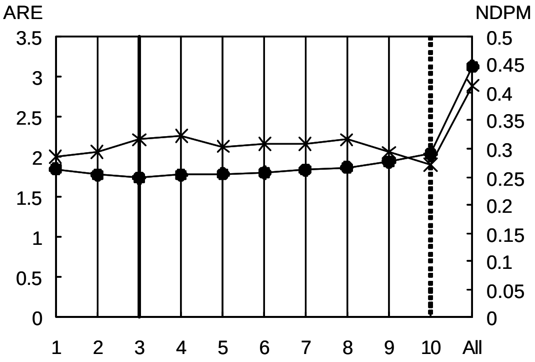


B.1. 12. ACSA & AWSMN

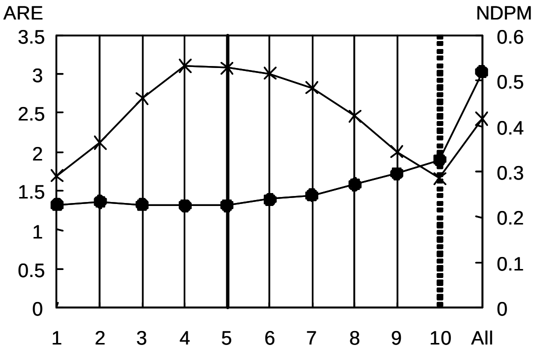
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



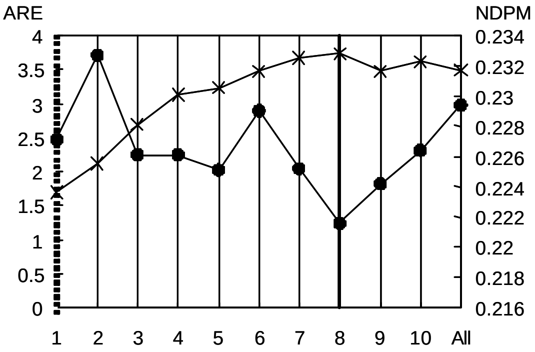
B.1. 13. ACSA & AWSAR



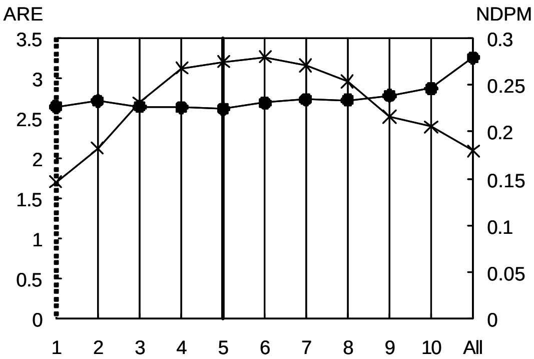
B.1. 14. ACSA & AWSMR



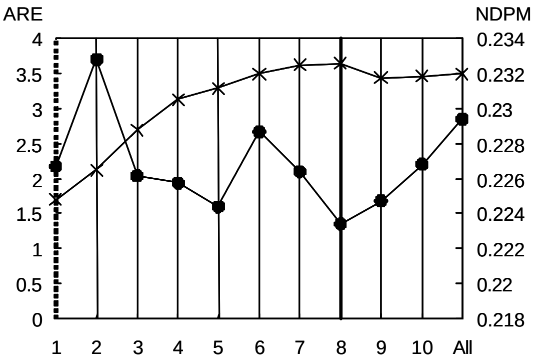
B.1. 15. ACSM & WS



B.1. 16. ACSM & AWSAC

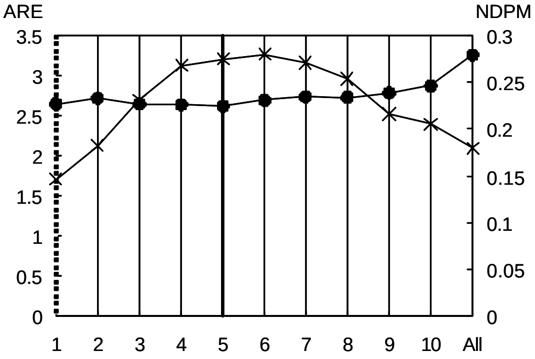


B.1. 17. ACSM & AWSMC

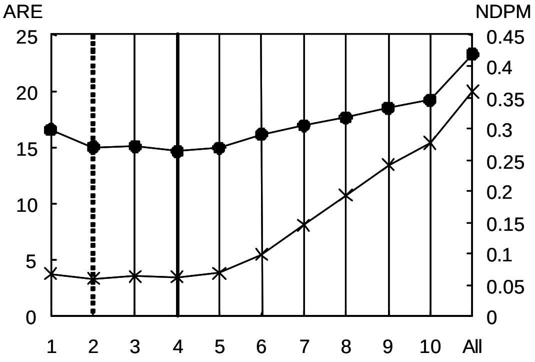


B.1. 18. ACSM & AWSAN

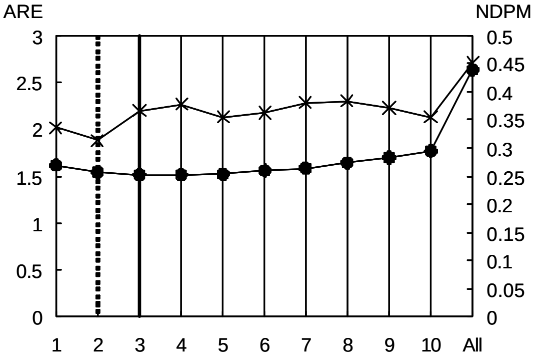
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



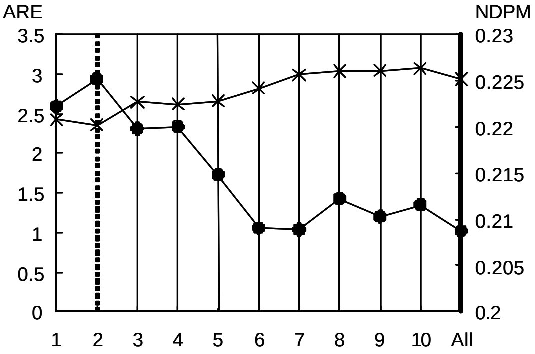
B.1. 19. ACSM & AWSMN



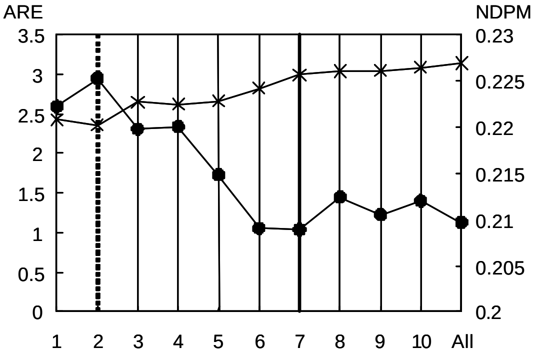
B.1. 20. ACSM & AWSAR



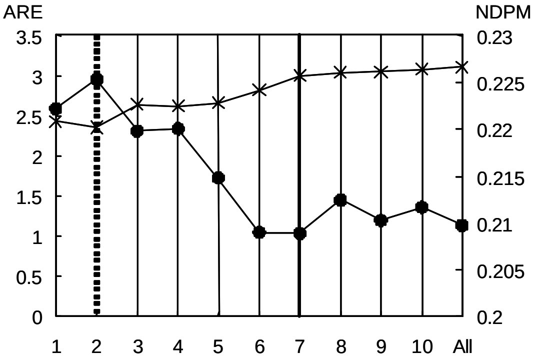
B.1. 21. ACSM & AWSMR



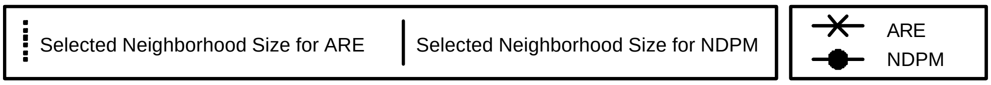
B.1. 22. CC & WS

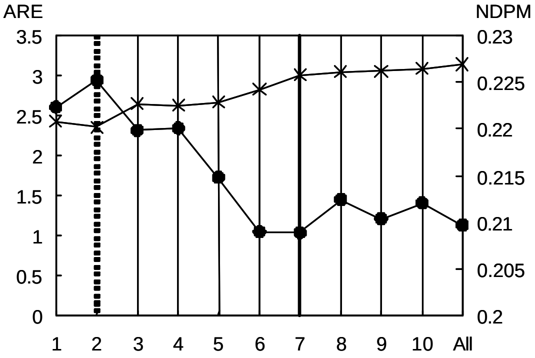


B.1. 23. CC & AWSAC

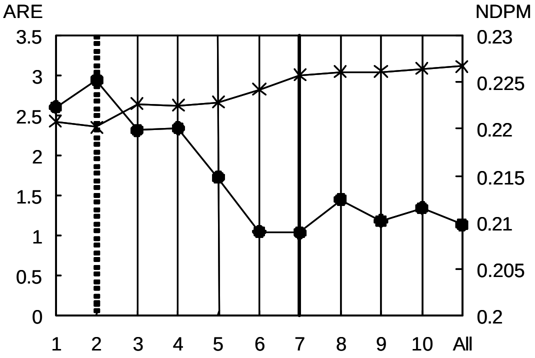


B.1. 24. CC & AWSMC

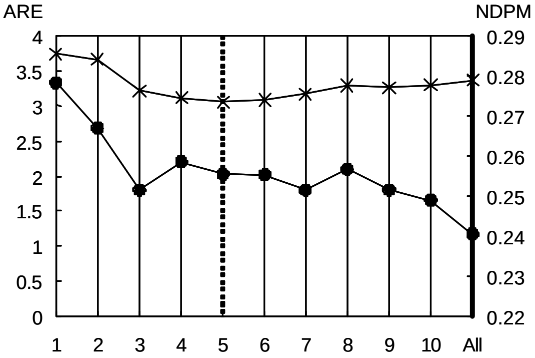

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



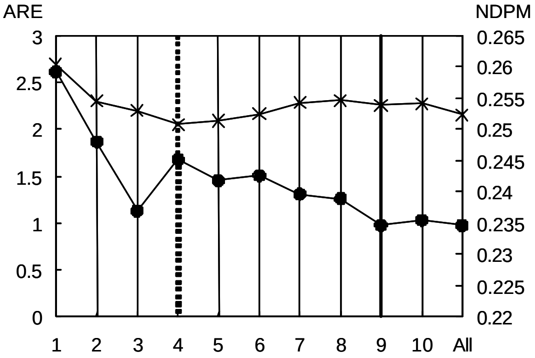
B.1. 25. CC & AWSAN



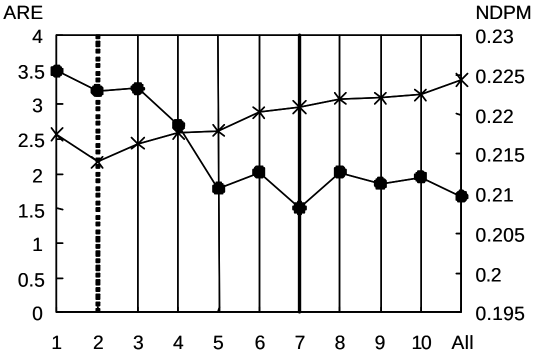
B.1. 26. CC & AWSMN



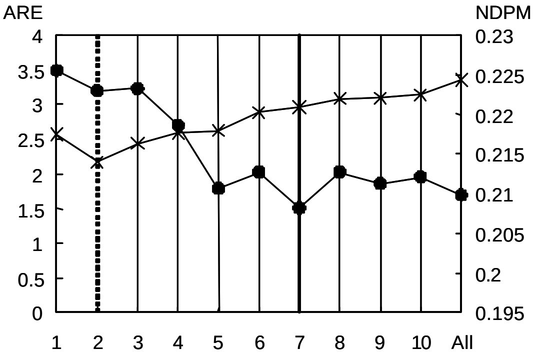
B.1. 27. CC & AWSAR



B.1. 28. CC & AWSMR

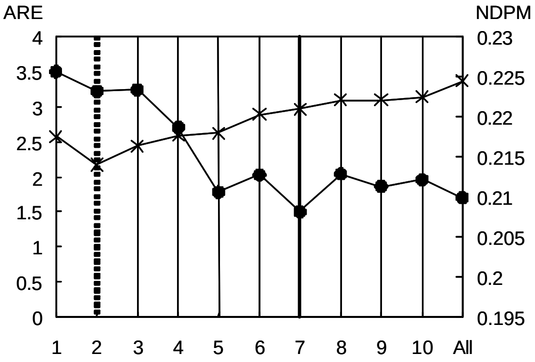


B.1. 29. CCM & WS

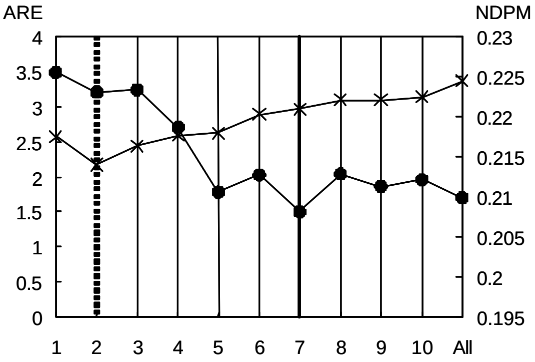


B.1. 30. CCM & AWSAC

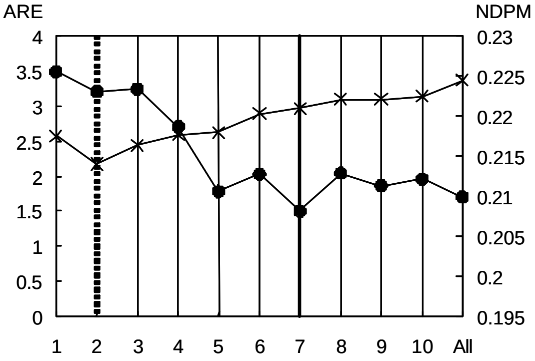
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



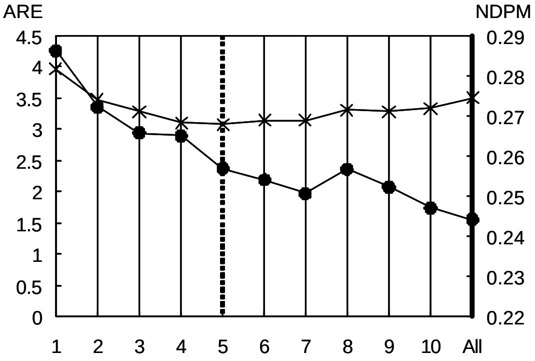
B.1. 31. CCM & AWSMC



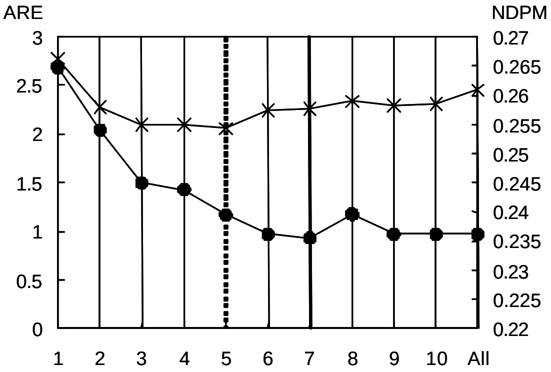
B.1. 32. CCM & AWSAN



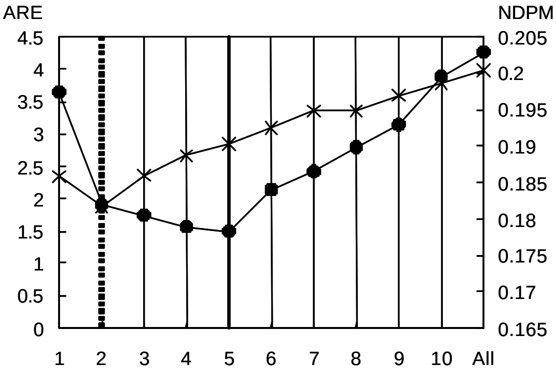
B.1. 33. CCM & AWSMN



B.1. 34. CCM & AWSAR



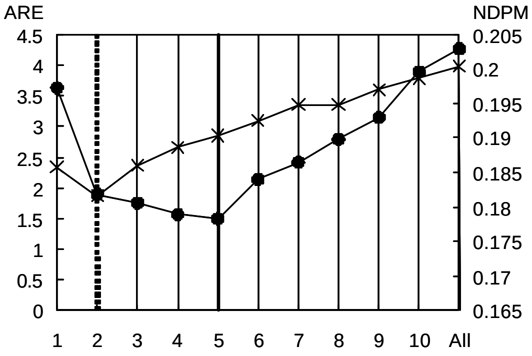
B.1. 35. CCM & AWSMR



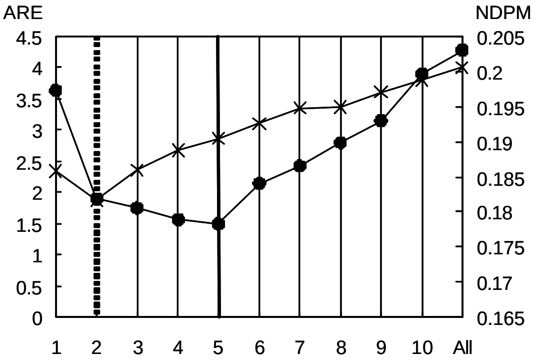
B.1. 36. RC & WS

┊ Selected Neighborhood Size for ARE | Selected Neighborhood Size for NDPM

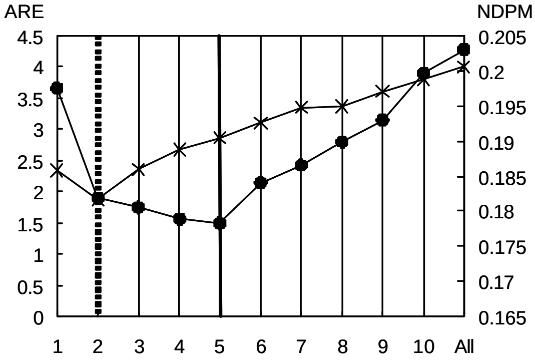
✕ ARE
● NDPM



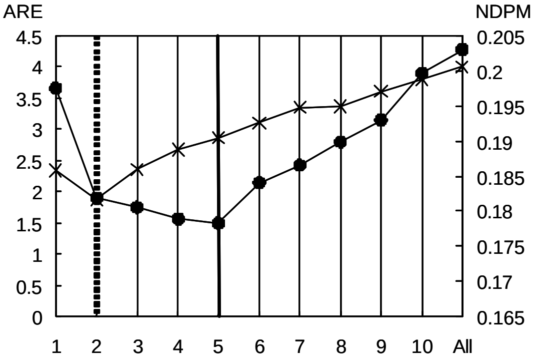
B.1. 37. RC & AWSAC



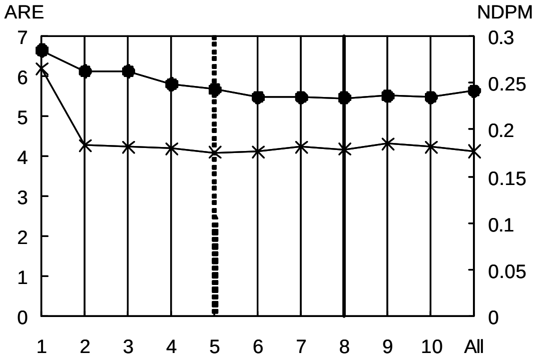
B.1. 38. RC & AWSMC



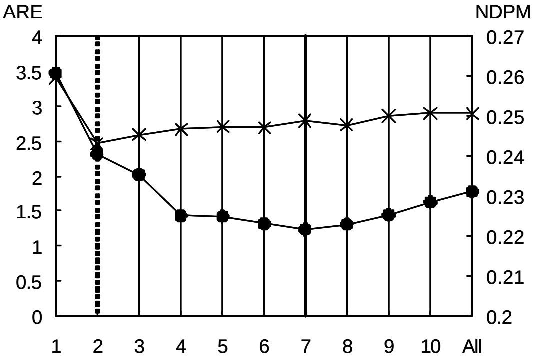
B.1. 39. RC & AWSAN



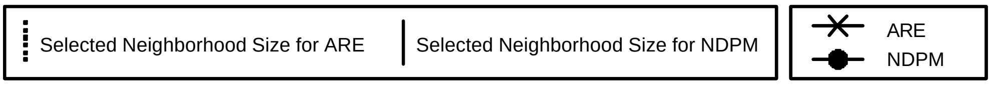
B.1. 40. RC & AWSMN

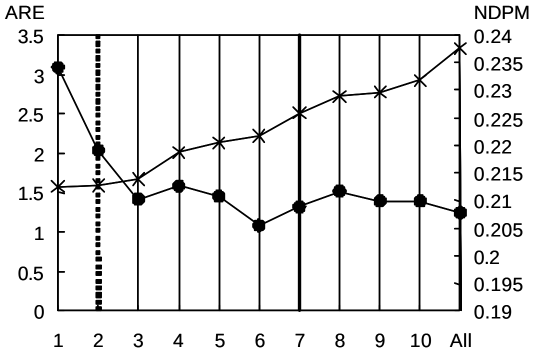


B.1. 41. RC & AWSAR

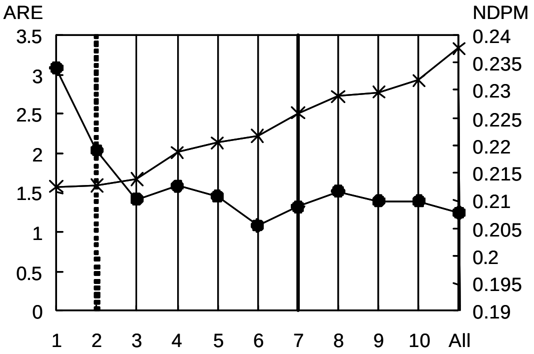


B.1. 42. RC & AWSMR

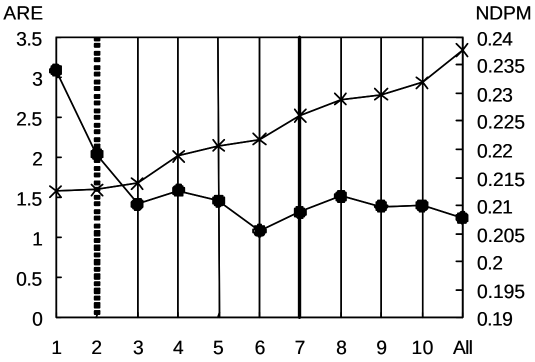




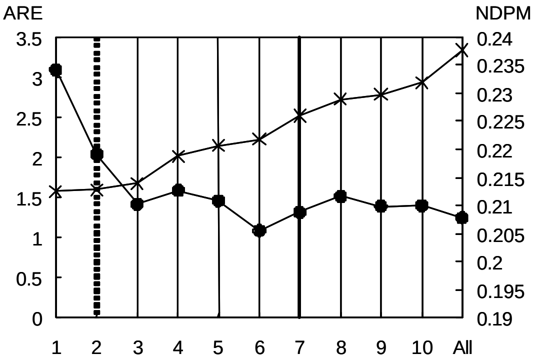
B.1. 43. DSA & WS



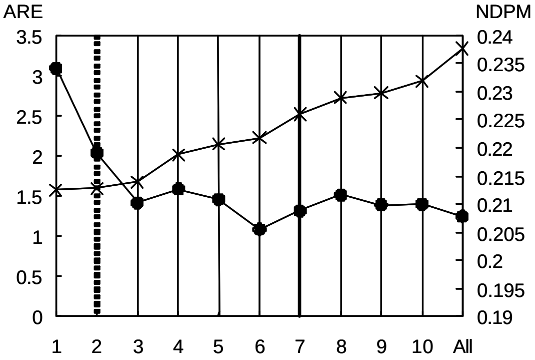
B.1. 44. DSA & AWSAC



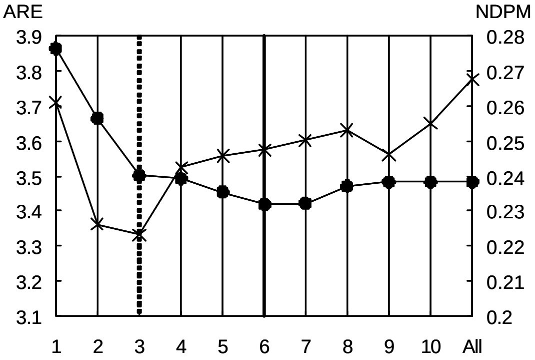
B.1. 45. DSA & AWSMC



B.1. 46. DSA & AWSAN

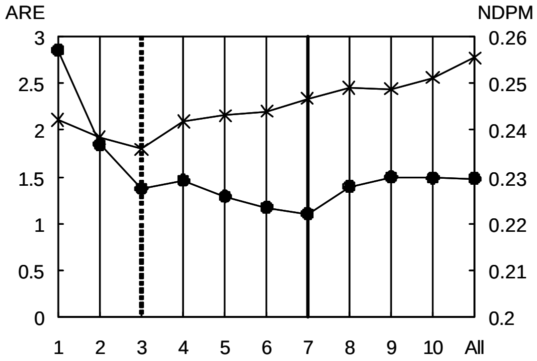


B.1. 47. DSA & AWSMN

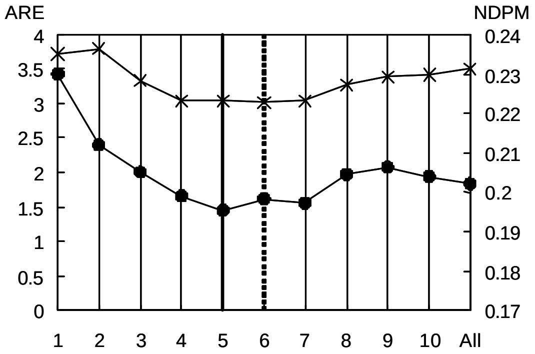


B.1. 48. DSA & AWSAR

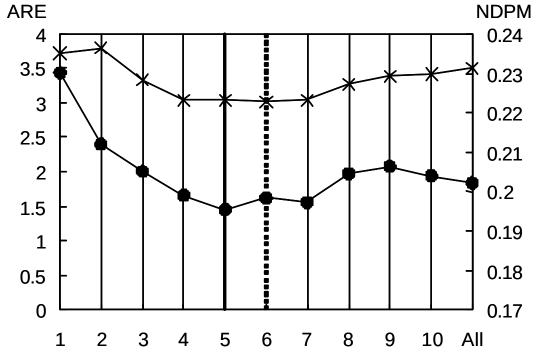
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



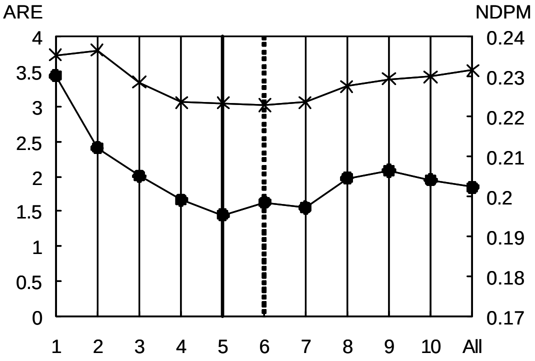
B.1. 49. DSA & AWSMR



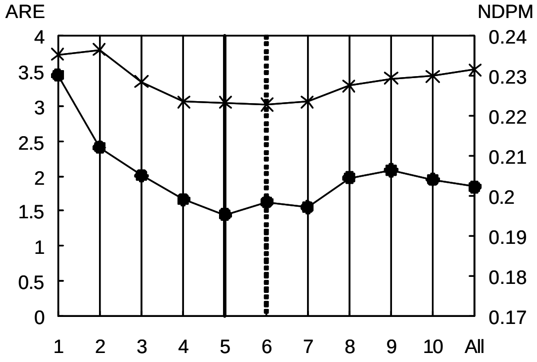
B.1. 50. DSM & WS



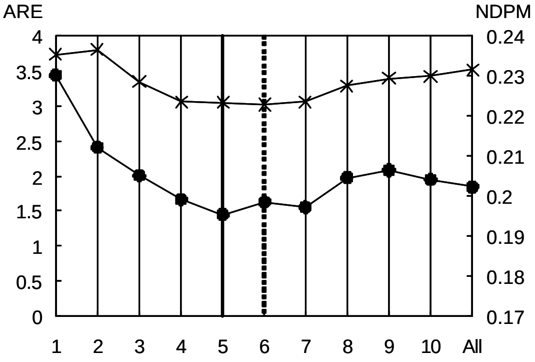
B.1. 51. DSM & AWSAC



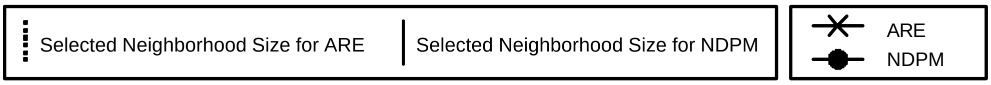
B.1. 52. DSM & AWSMC

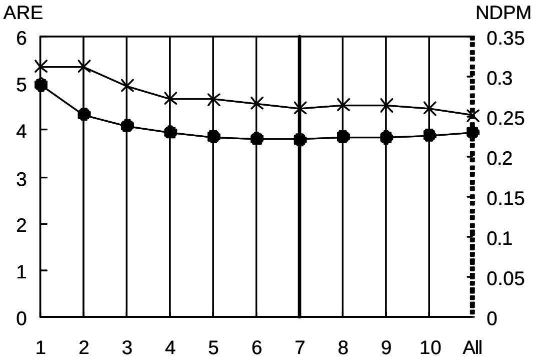


B.1. 53. DSM & AWSAN

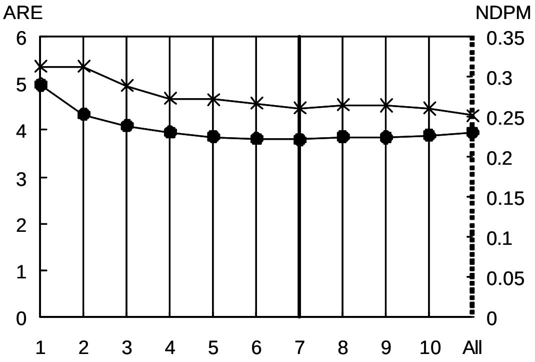


B.1. 54. DSM & AWSMN


Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



B.1. 55. DSM & AWSAR



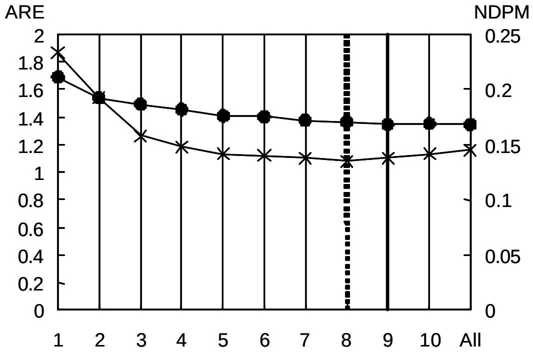
B.1. 56. DSM & AWSMR

B.2 Evaluation Results with Usage Histories of MS PowerPoint 2002

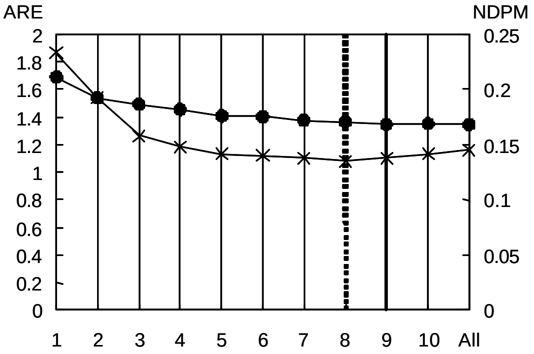
The following graphs show the evaluation results for selecting appropriate neighborhood size for each CF algorithm when the evaluation was executed with the usage histories of MS PowerPoint 2002. The following graphs are written in the same manner as the graphs written in Appendix B.1.

⋮ Selected Neighborhood Size for ARE | Selected Neighborhood Size for NDPM

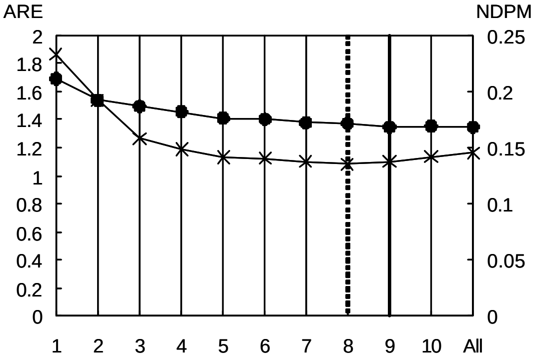
—×— ARE
—●— NDPM



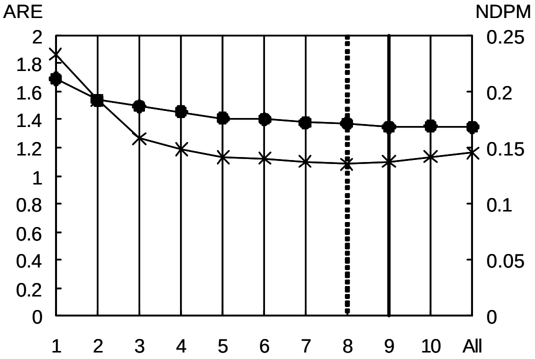
B.2. 1. CS & WS



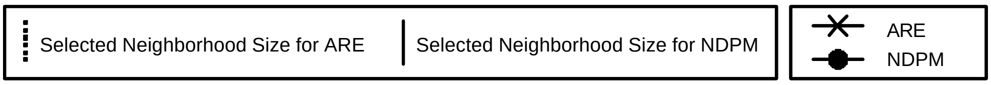
B.2. 2. CS & AWSAC

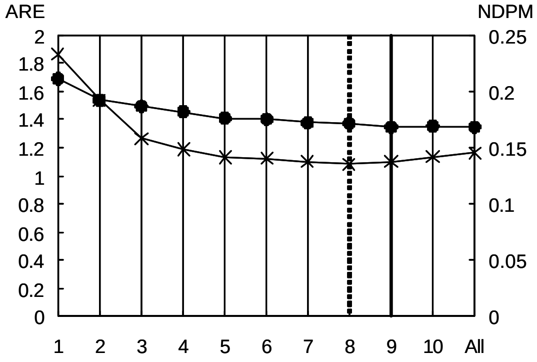


B.2. 3. CS & AWSMC

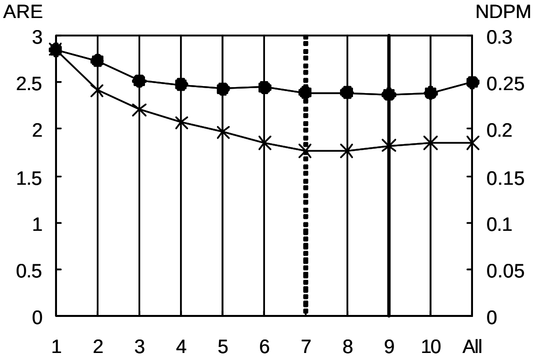


B.2. 4. CS & AWSAN

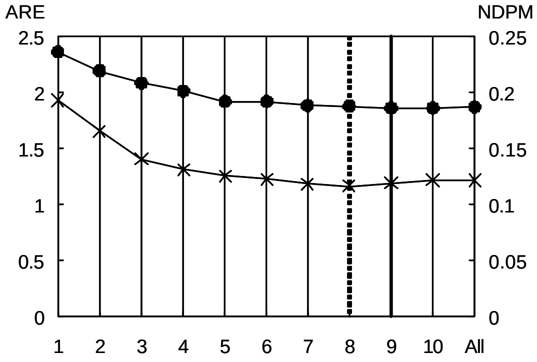

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



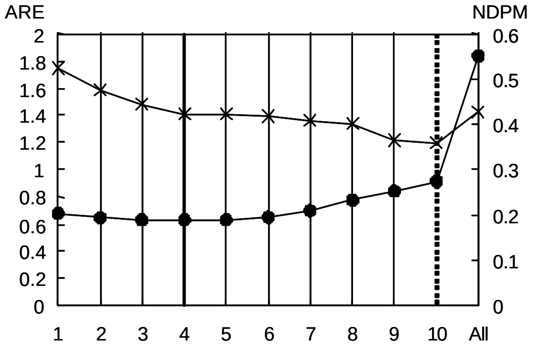
B.2. 5. CS & AWSMN



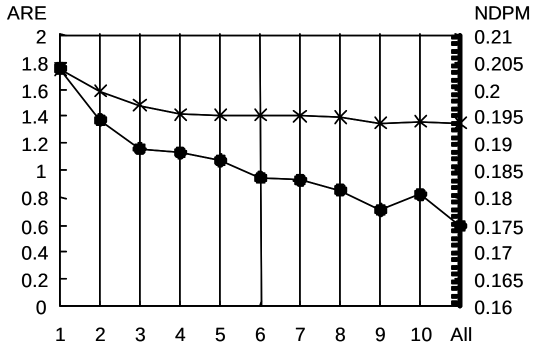
B.2. 6. CS & AWSAR



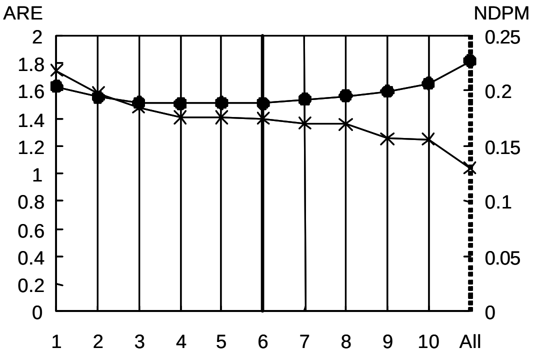
B.2. 7. CS & AWSMR



B.2. 8. ACSA & WS

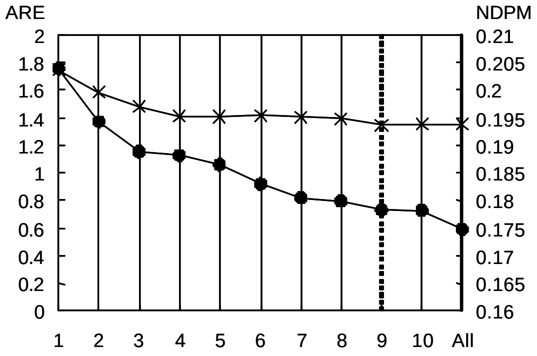


B.2. 9. ACSA & AWSAC

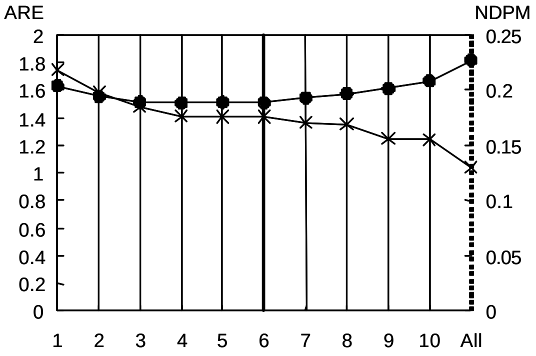


B.2. 10. ACSA & AWSMC

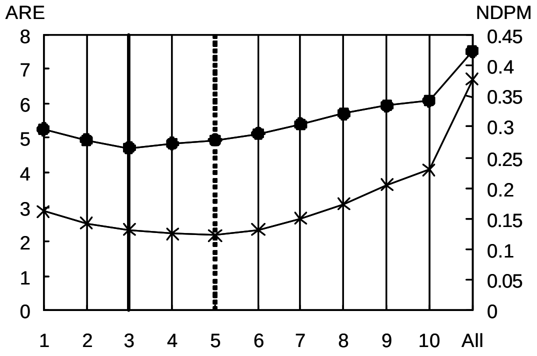
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



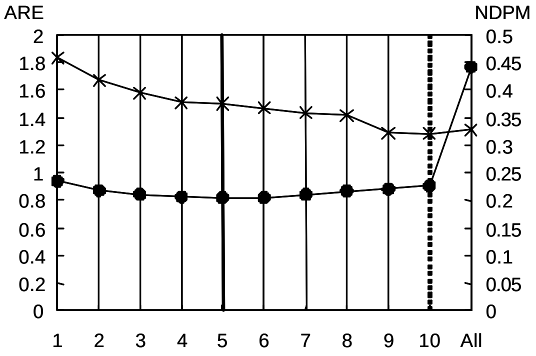
B.2. 11. ACSA & AWSAN



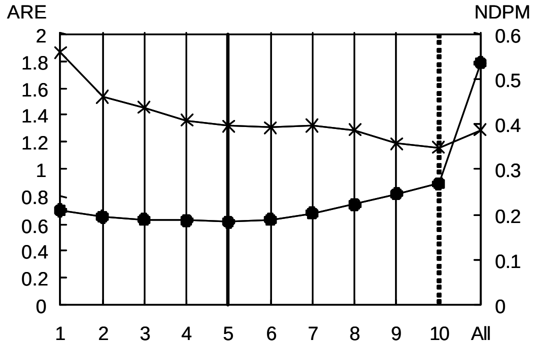
B.2. 12. ACSA & AWSMN



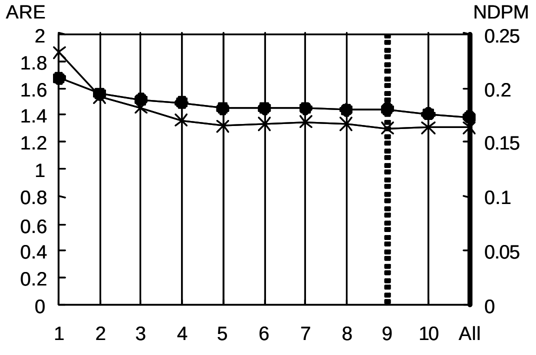
B.2. 13. ACSA & AWSAR



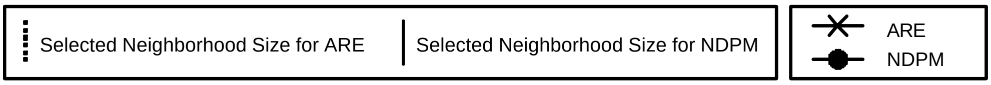
B.2. 14. ACSA & AWSMR

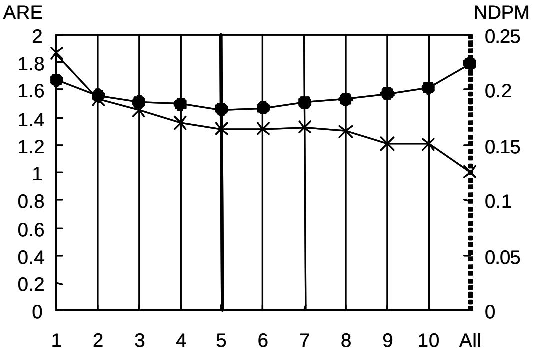


B.2. 15. ACSM & WS

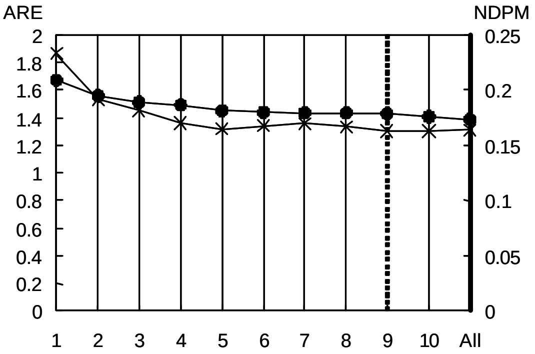


B.2. 16. ACSM & AWSAC

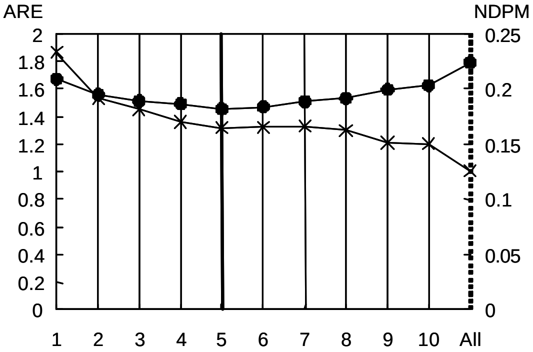




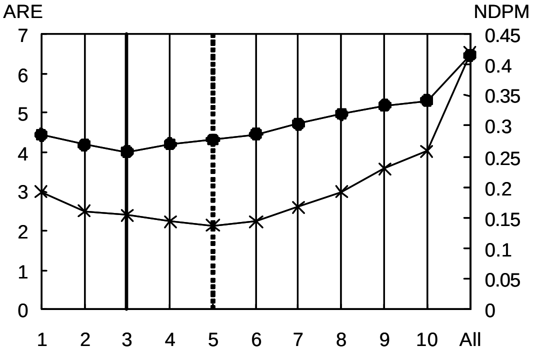
B.2. 17. ACSM & AWSMC



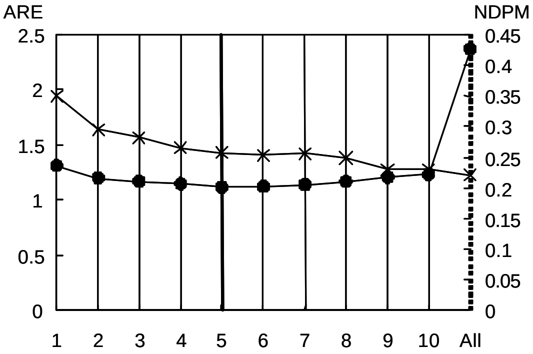
B.2. 18. ACSM & AWSAN



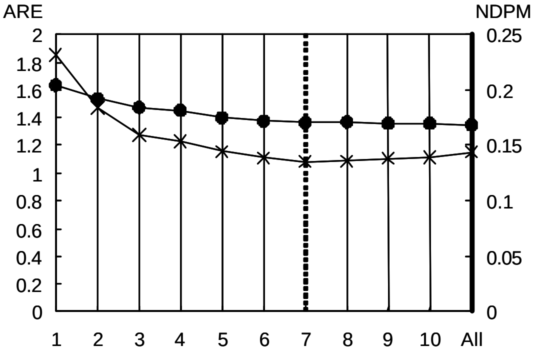
B.2. 19. ACSM & AWSMN



B.2. 20. ACSM & AWSAR



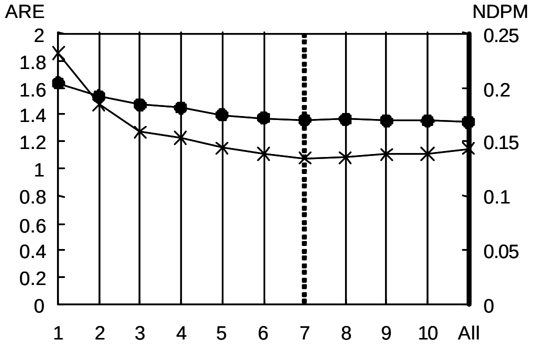
B.2. 21. ACSM & AWSMR



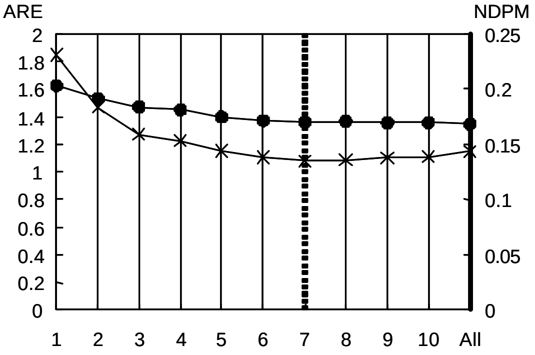
B.2. 22. CC & WS

┆ Selected Neighborhood Size for ARE | Selected Neighborhood Size for NDPM

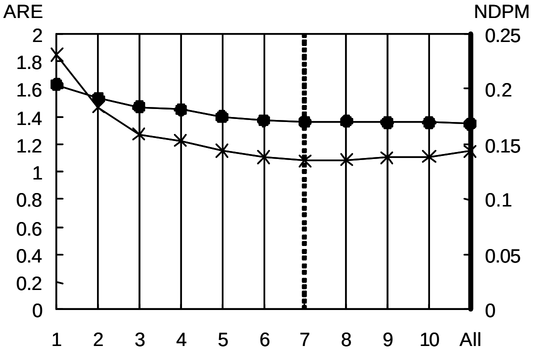
—✕— ARE
—●— NDPM



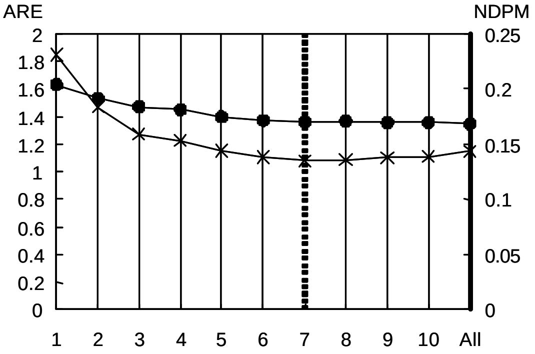
B.2. 23. CC & AWSAC



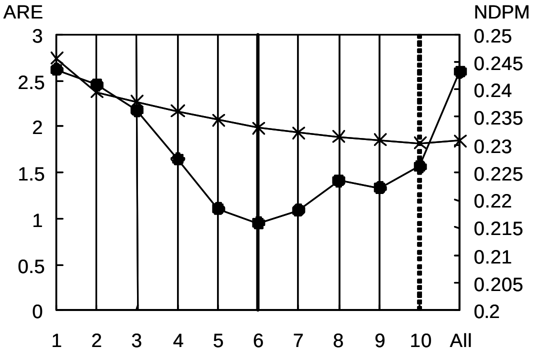
B.2. 24. CC & AWSMC



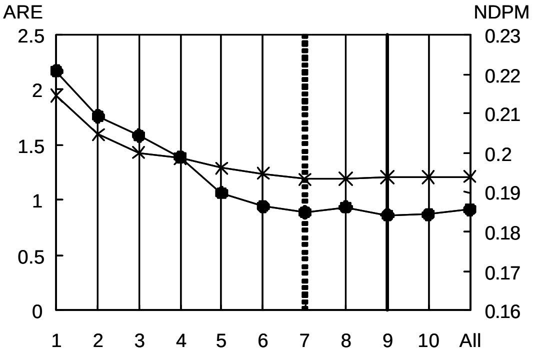
B.2. 25. CC & AWSAN



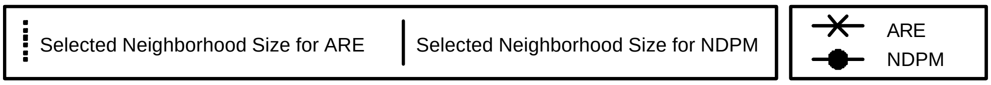
B.2. 26. CC & AWSMN

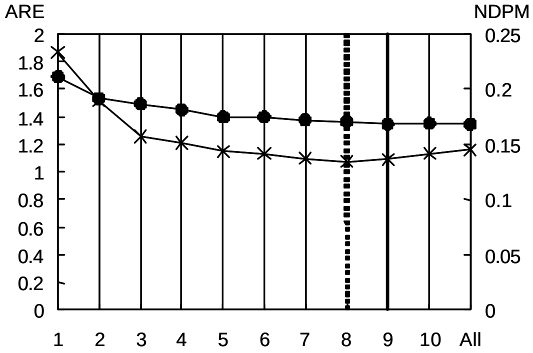


B.2. 27. CC & AWSAR

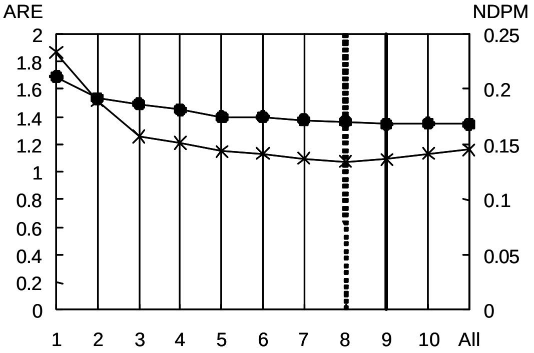


B.2. 28. CC & AWSMR

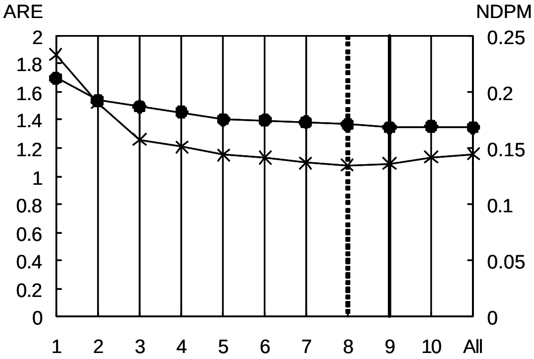

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



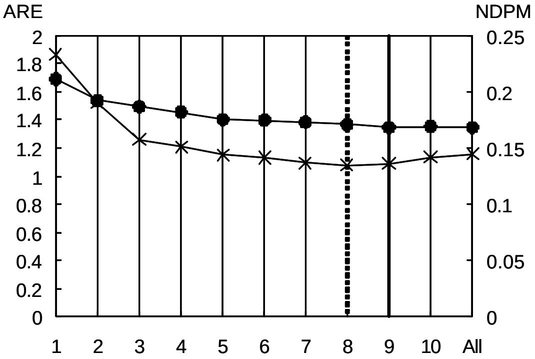
B.2. 29. CCM & WS



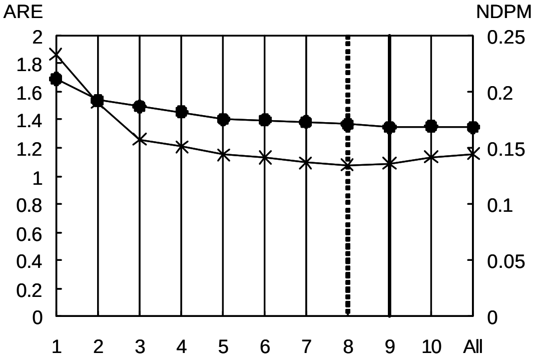
B.2. 30. CCM & AWSAC



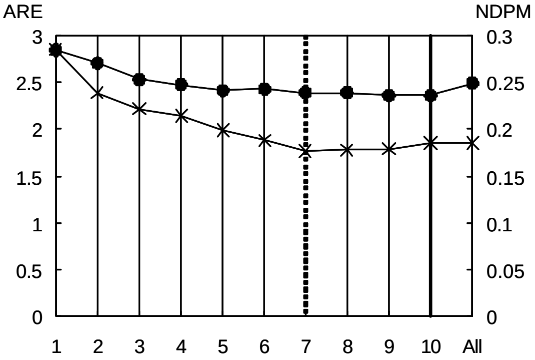
B.2. 31. CCM & AWSMC



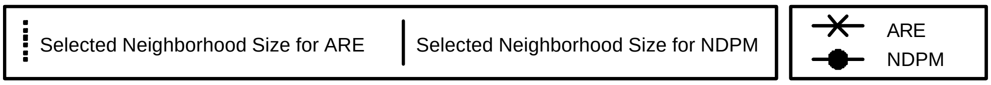
B.2. 32. CCM & AWSAN

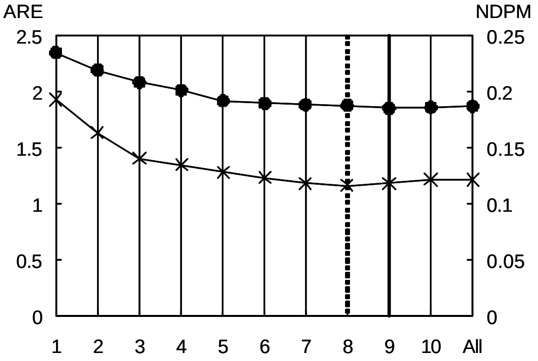


B.2. 33. CCM & AWSMN

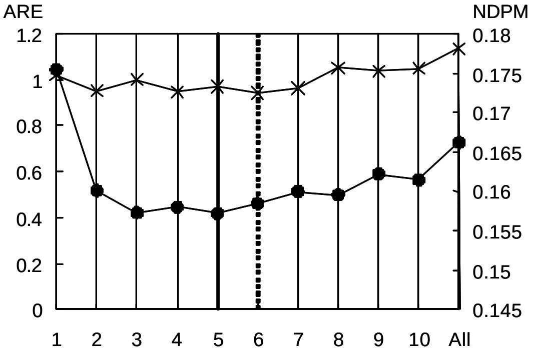


B.2. 34. CCM & AWSAR

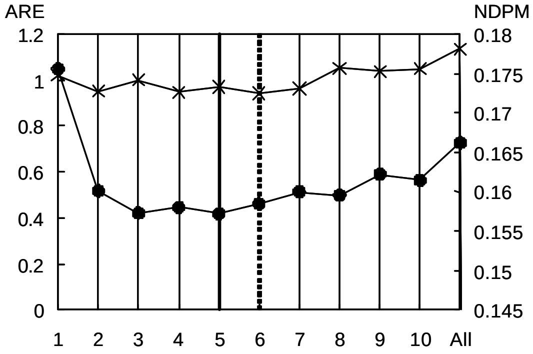




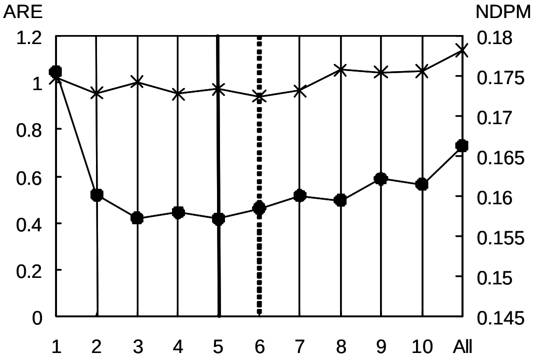
B.2. 35. CCM & AWSMR



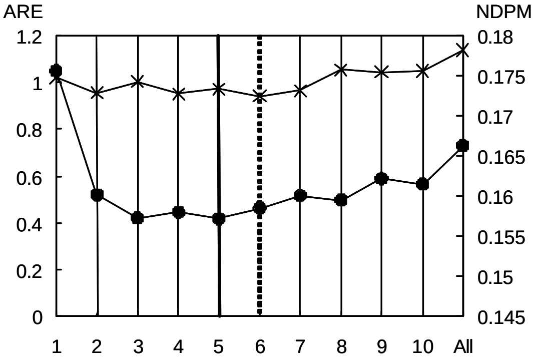
B.2. 36. RC & WS



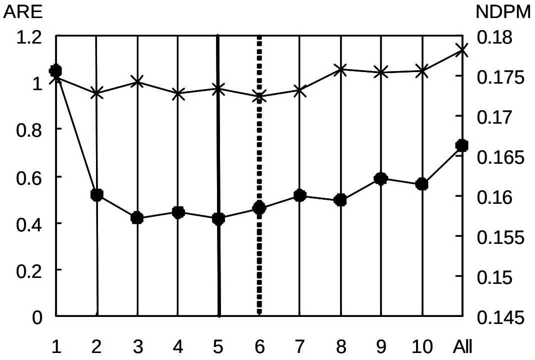
B.2. 37. RC & AWSAC



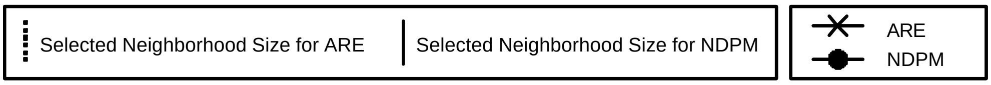
B.2. 38. RC & AWSMC

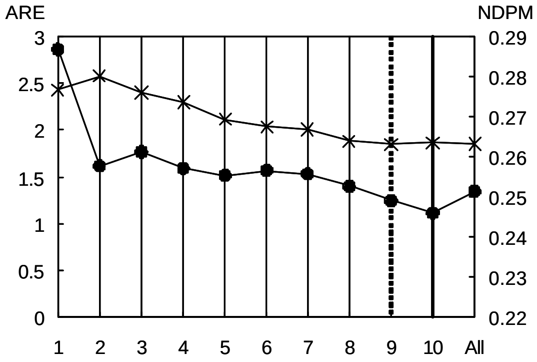


B.2. 39. RC & AWSAN

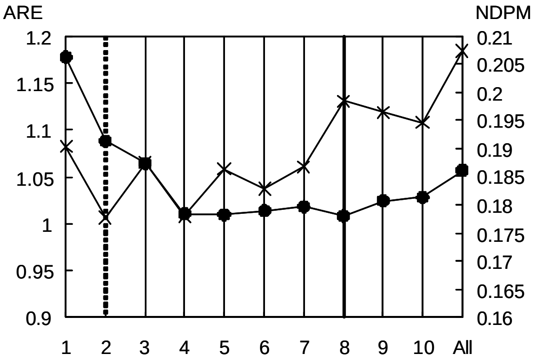


B.2. 40. RC & AWSMN

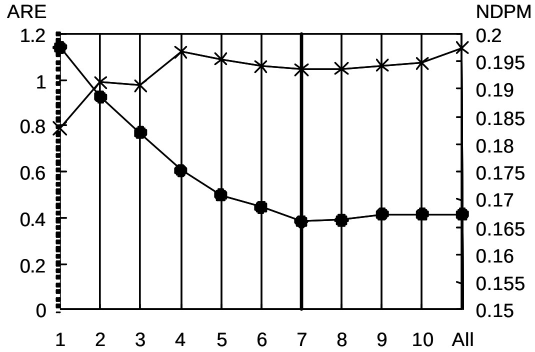

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



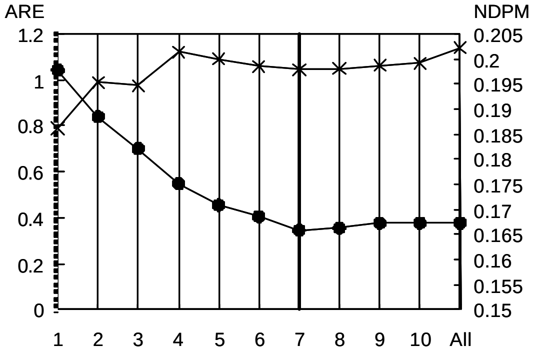
B.2. 41. RC & AWSAR



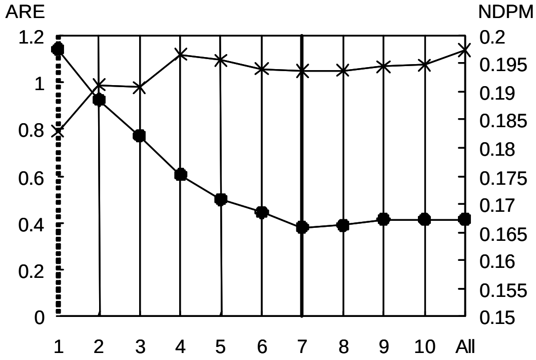
B.2. 42. RC & AWSMR



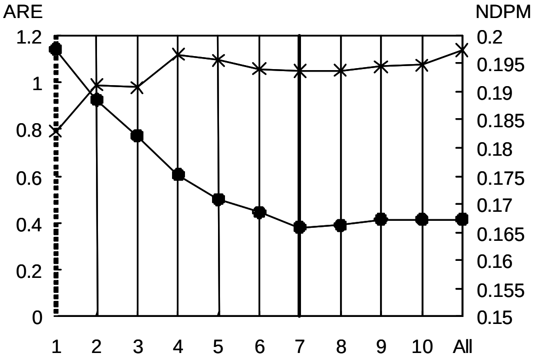
B.2. 43. DSA & WS



B.2. 44. DSA & AWSAC

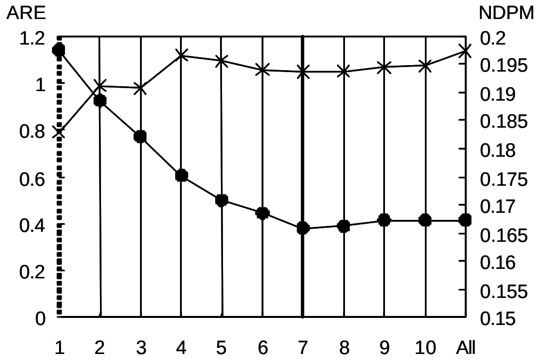


B.2. 45. DSA & AWSMC

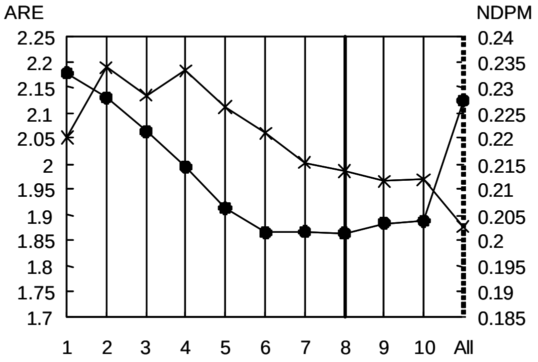


B.2. 46. DSA & AWSAN

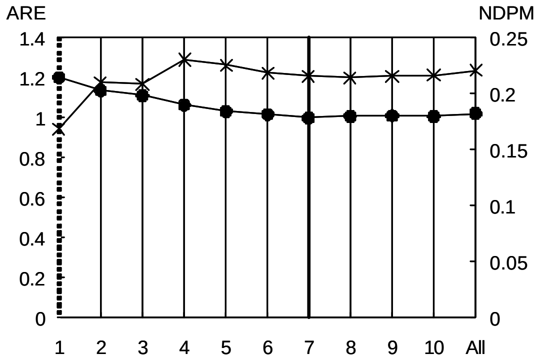
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



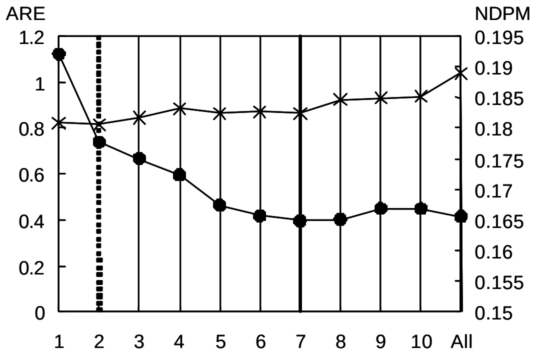
B.2. 47. DSA & AWSMN



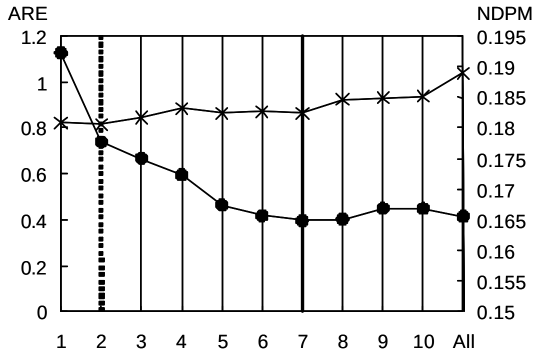
B.2. 48. DSA & AWSAR



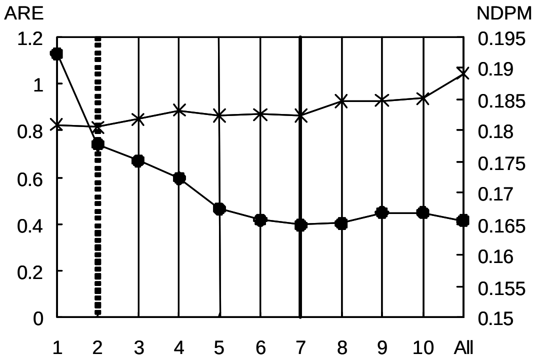
B.2. 49. DSA & AWSMR



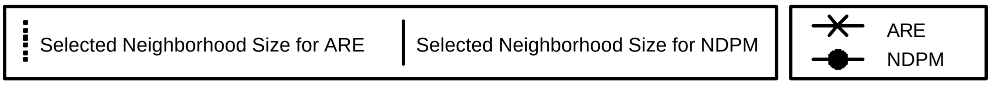
B.2. 50. DSM & WS

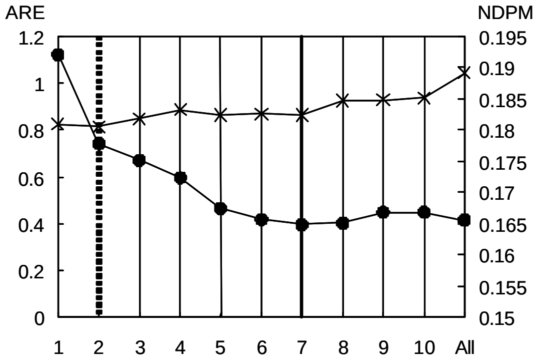


B.2. 51. DSM & AWSAC

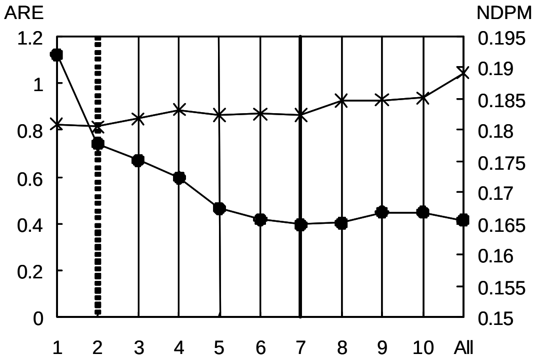


B.2. 52. DSM & AWSMC

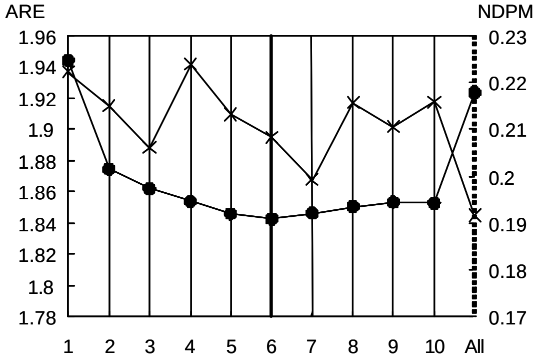

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



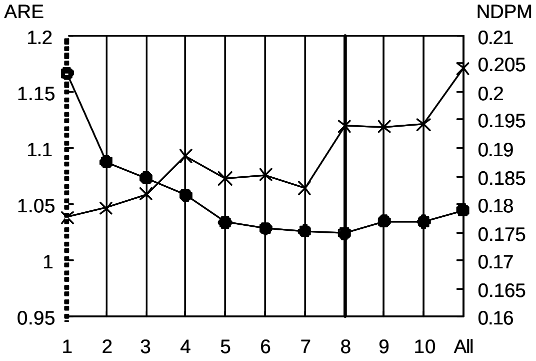
B.2. 53. DSM & AWSAN



B.2. 54. DSM & AWSMN



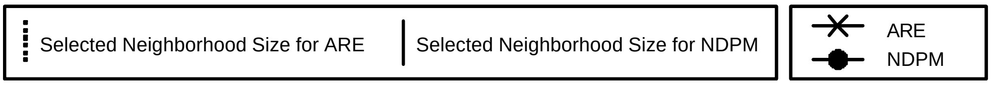
B.2. 55. DSM & AWSAR

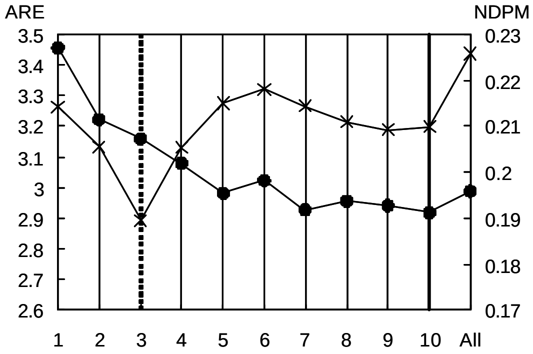


B.2. 56. DSM & AWSMR

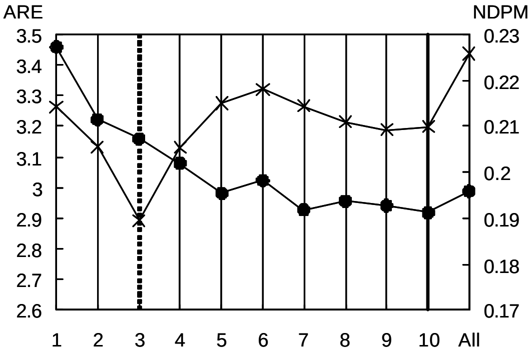
B.3 Evaluation Results with Usage Histories of MS Excel 2002

The following graphs show the evaluation results for selecting appropriate neighborhood size for each CF algorithm when the evaluation was executed with the usage histories of MS Excel 2002. The following graphs are written in the same manner as the graphs written in Appendix B.1 and B.2.

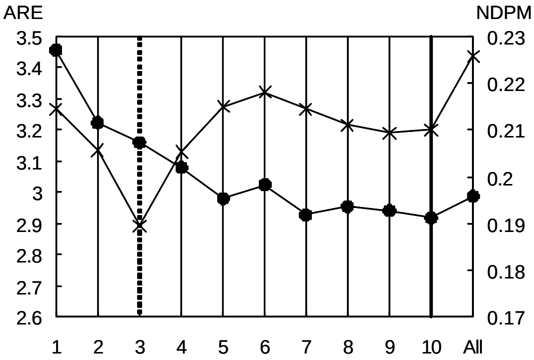

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



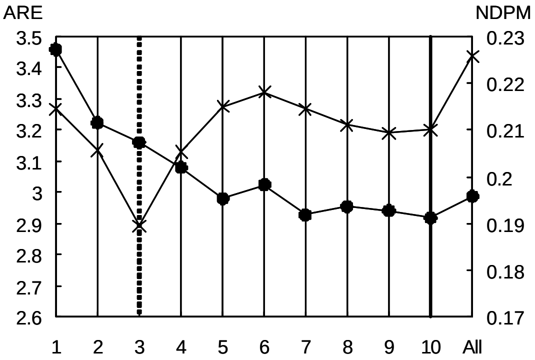
B.3. 1. CS & WS



B.3. 2. CS & AWSAC

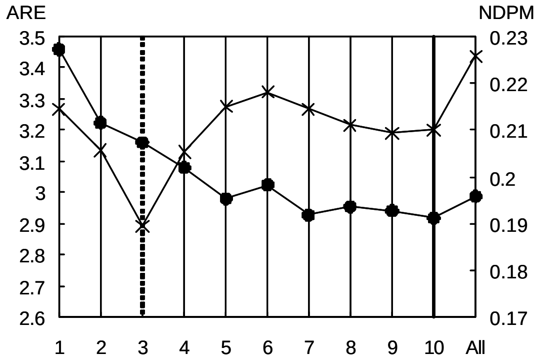


B.3. 3. CS & AWSMC

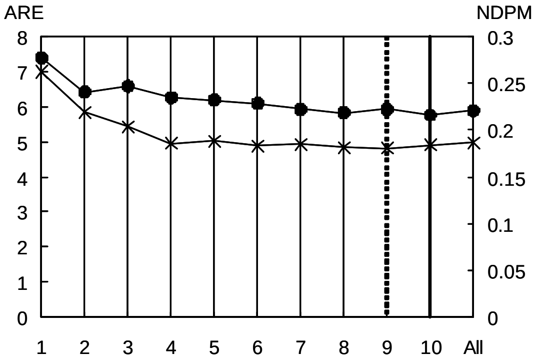


B.3. 4. CS & AWSAN

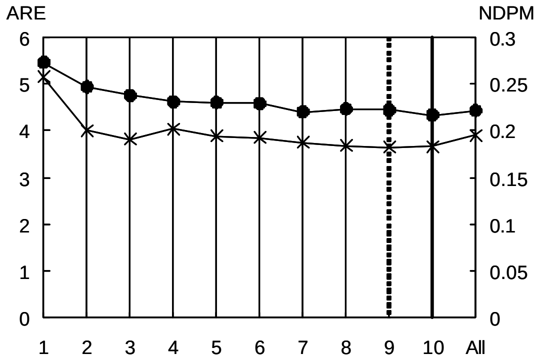
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



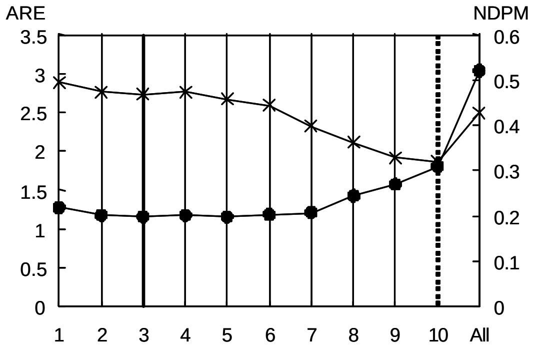
B.3. 5. CS & AWSMN



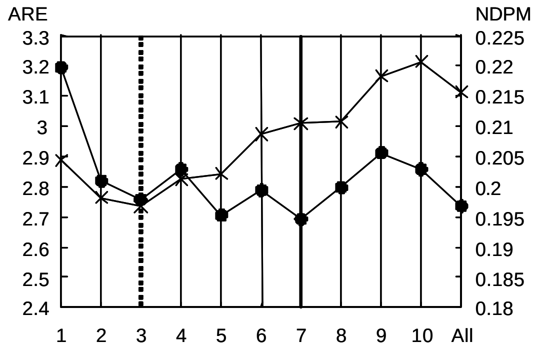
B.3. 6. CS & AWSAR



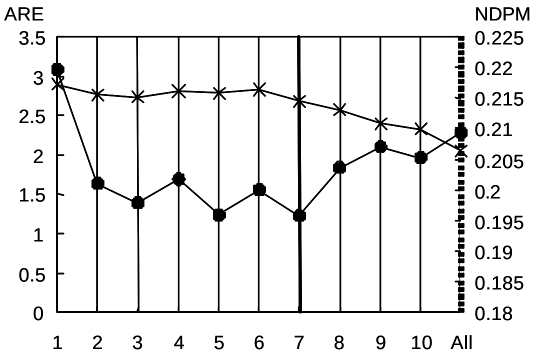
B.3. 7. CS & AWSMR



B.3. 8. ACSA & WS

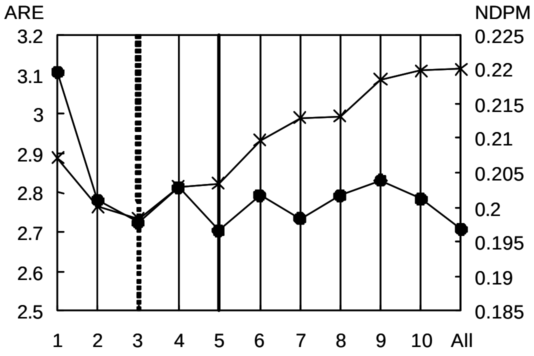


B.3. 9. ACSA & AWSAC

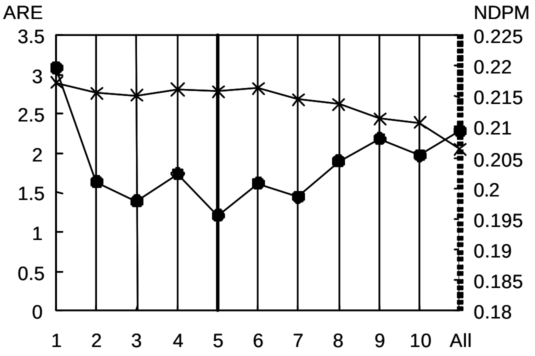


B.3. 10. ACSA & AWSMC

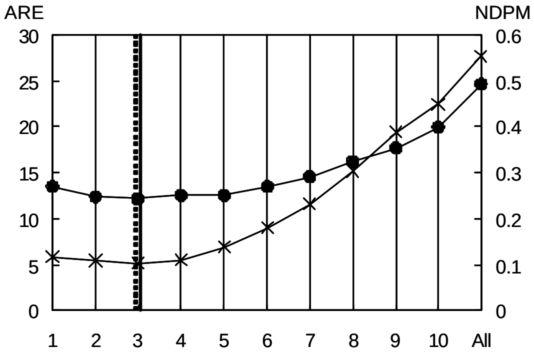
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



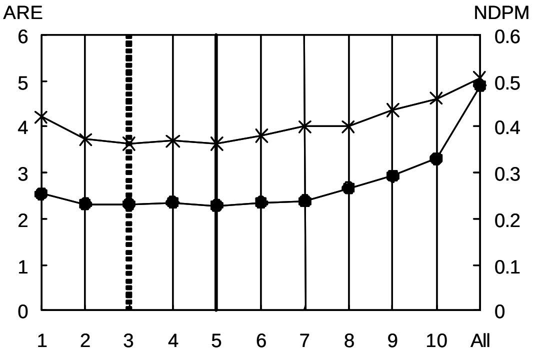
B.3. 11. ACSA & AWSAN



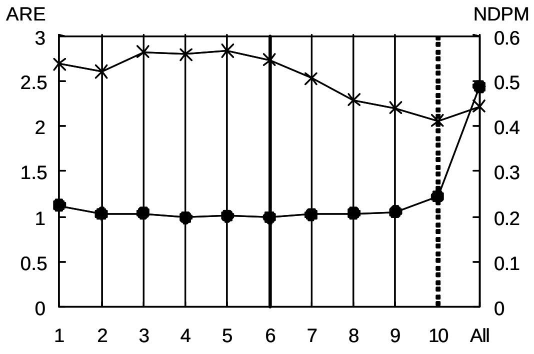
B.3. 12. ACSA & AWSMN



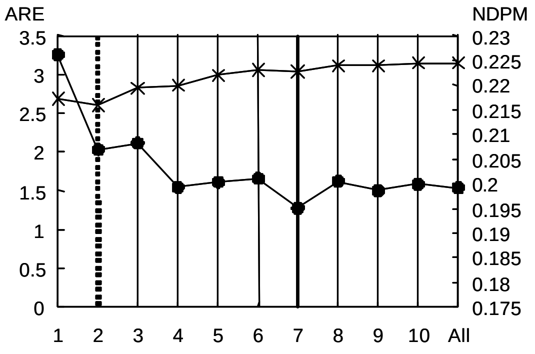
B.3. 13. ACSA & AWSAR



B.3. 14. ACSA & AWSMR

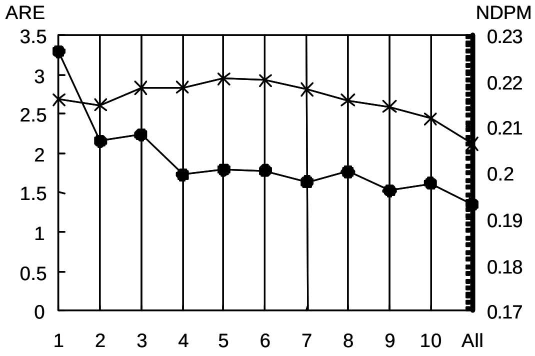


B.3. 15. ACSM & WS

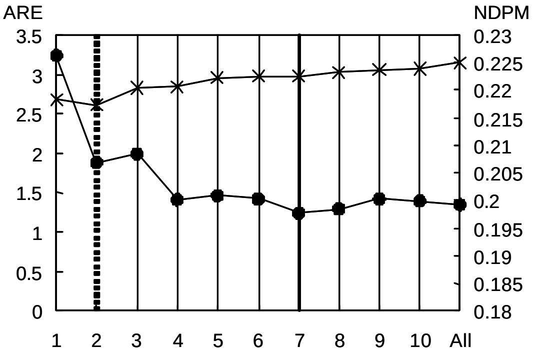


B.3. 16. ACSM & AWSAC

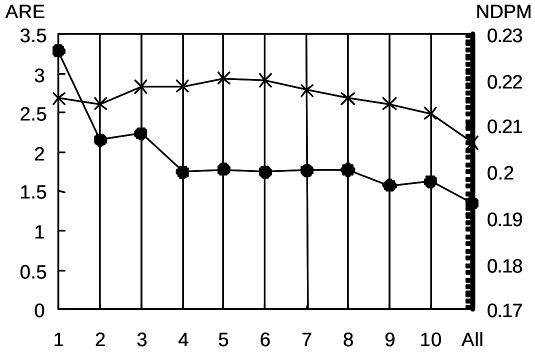
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



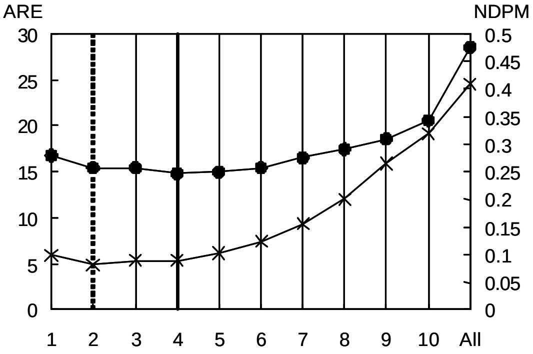
B.3. 17. ACSM & AWSMC



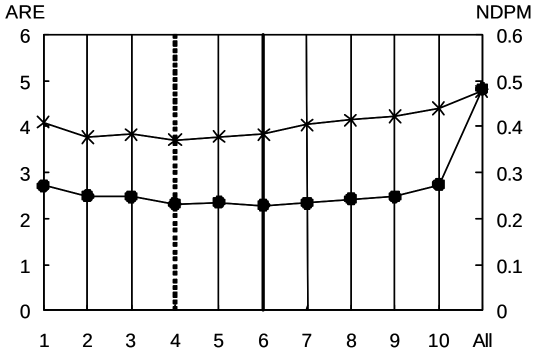
B.3. 18. ACSM & AWSAN



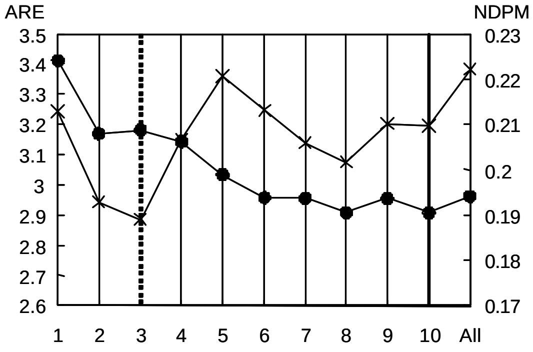
B.3. 19. ACSM & AWSMN



B.3. 20. ACSM & AWSAR

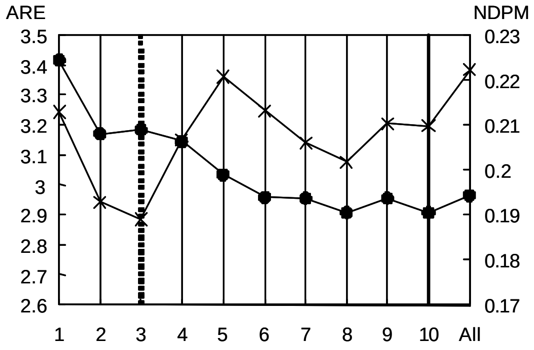


B.3. 21. ACSM & AWSMR

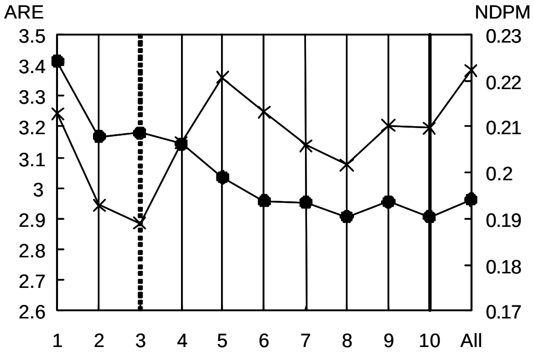


B.3. 22. CC & WS

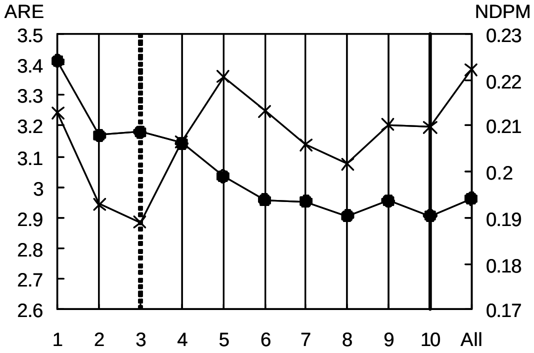
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



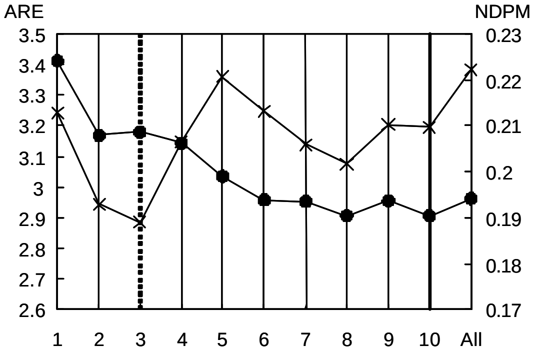
B.3. 23. CC & AWSAC



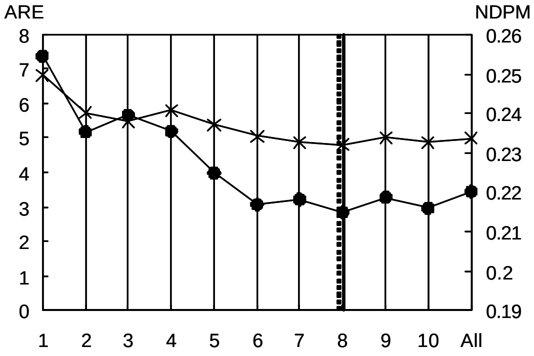
B.3. 24. CC & AWSMC



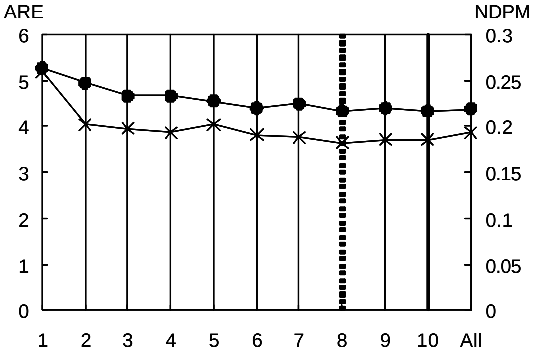
B.3. 25. CC & AWSAN



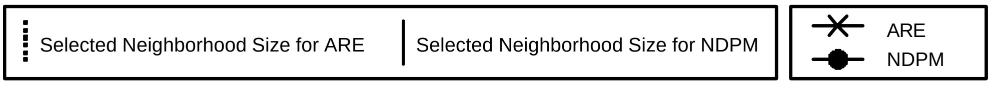
B.3. 26. CC & AWSMN

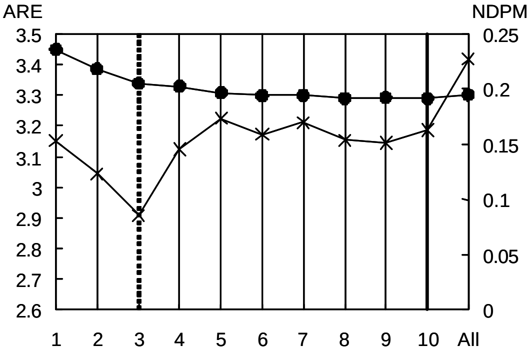


B.3. 27. CC & AWSAR

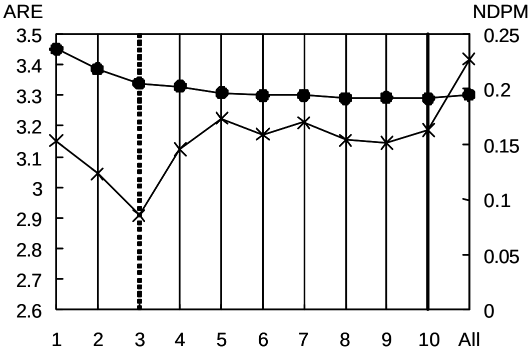


B.3. 28. CC & AWSMR

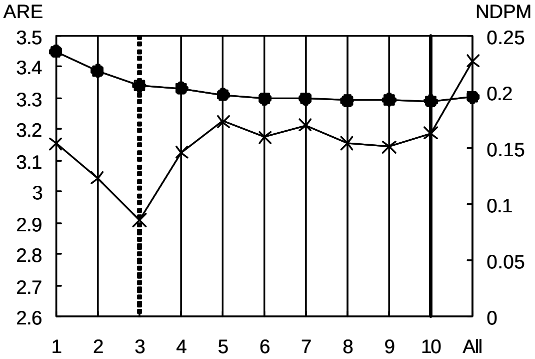

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



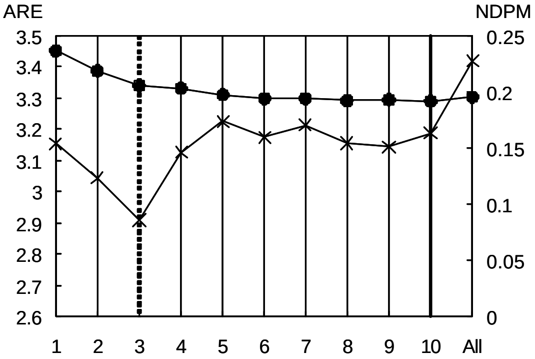
B.3. 29. CCM & WS



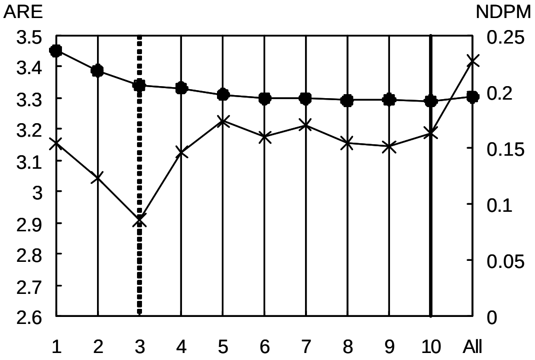
B.3. 30. CCM & AWSAC



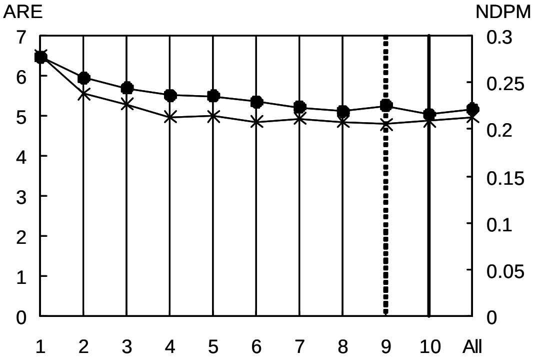
B.3. 31. CCM & AWSMC



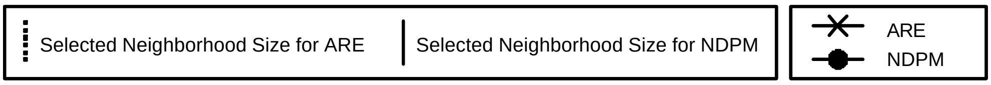
B.3. 32. CCM & AWSAN

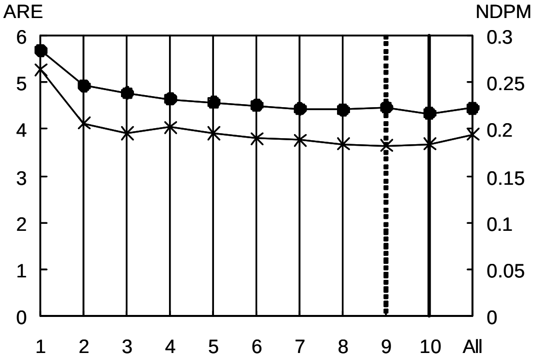


B.3. 33. CCM & AWSMN

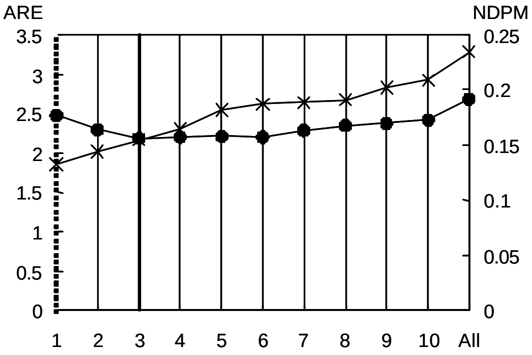


B.3. 34. CCM & AWSAR

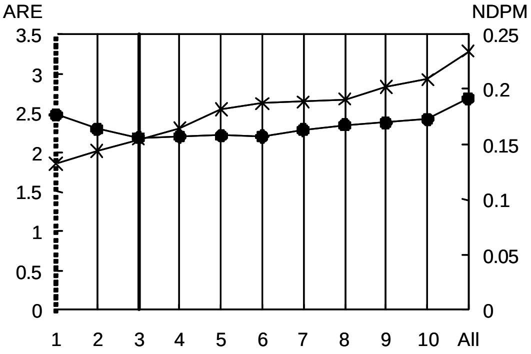

Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



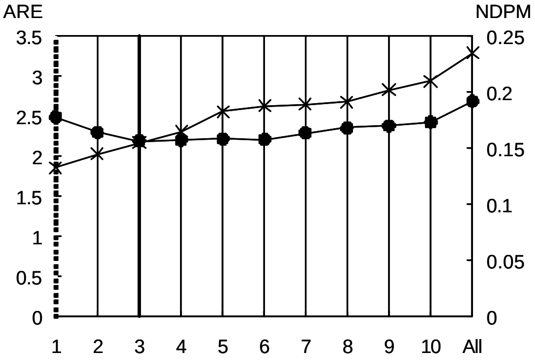
B.3. 35. CCM & AWSMR



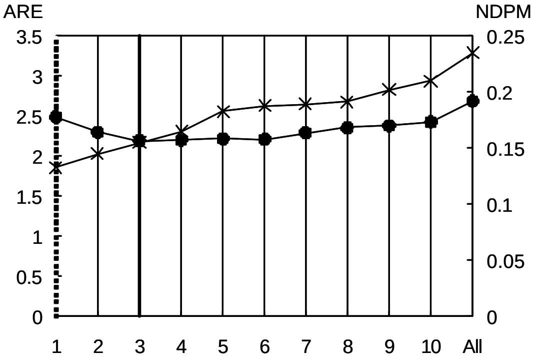
B.3. 36. RC & WS



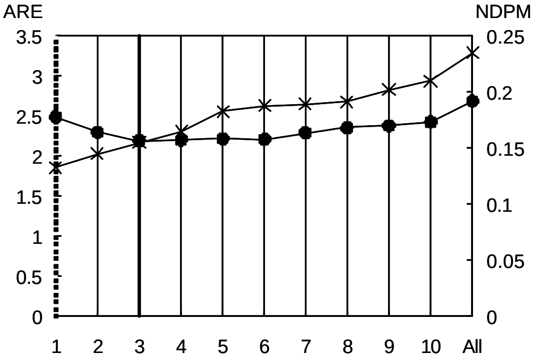
B.3. 37. RC & AWSAC



B.3. 38. RC & AWSMC

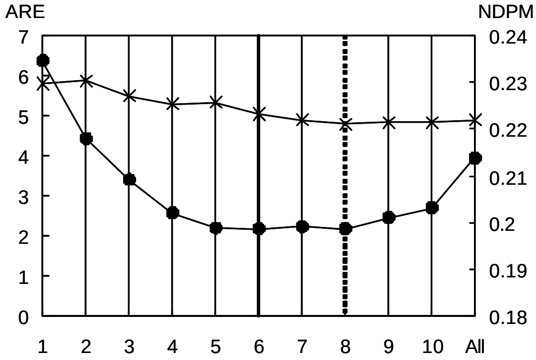


B.3. 39. RC & AWSAN

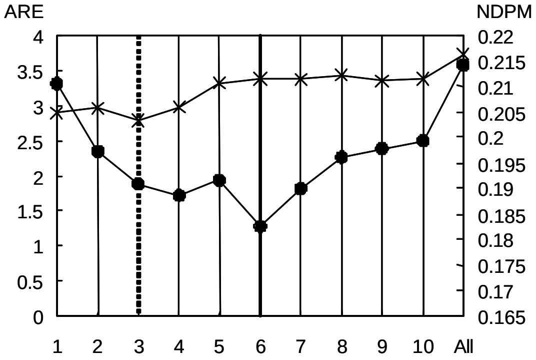


B.3. 40. RC & AWSMN

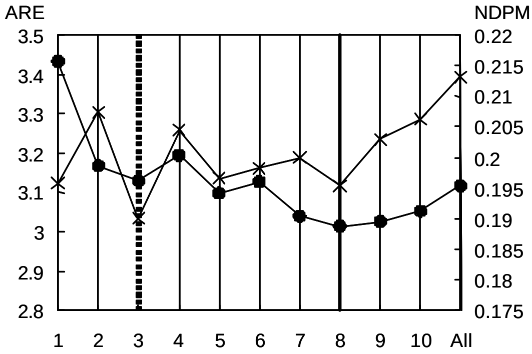
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



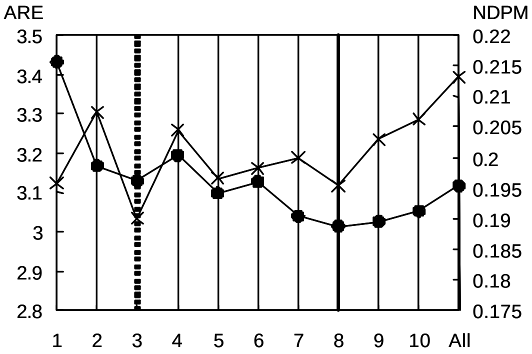
B.3. 41. RC & AWSAR



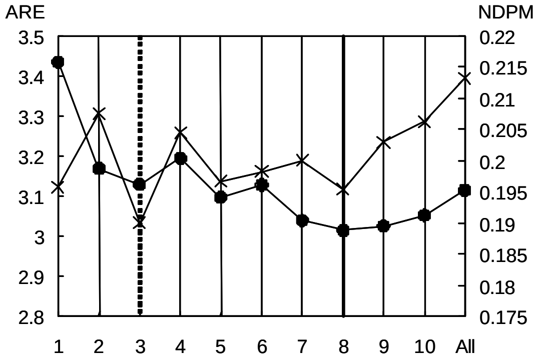
B.3. 42. RC & AWSMR



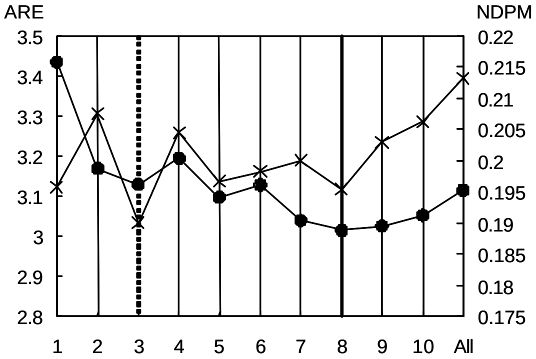
B.3. 43. DSA & WS



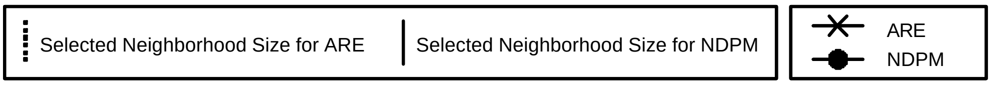
B.3. 44. DSA & AWSAC

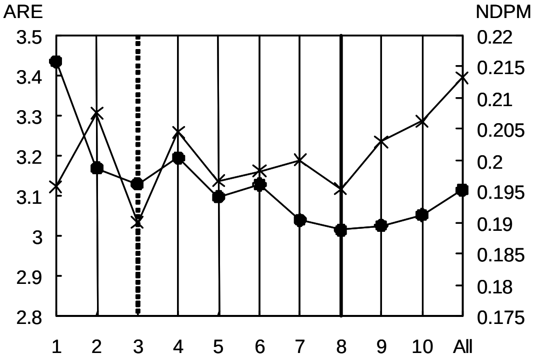


B.3. 45. DSA & AWSMC

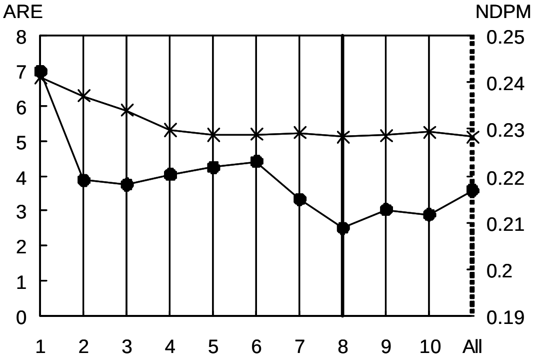


B.3. 46. DSA & AWSAN

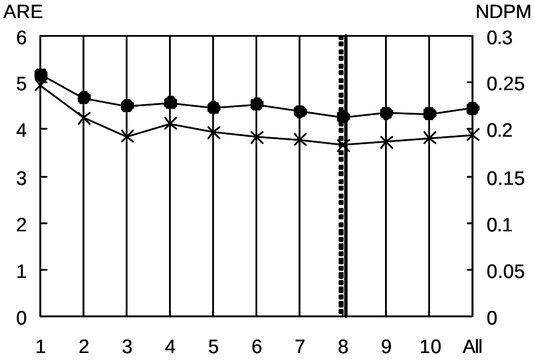




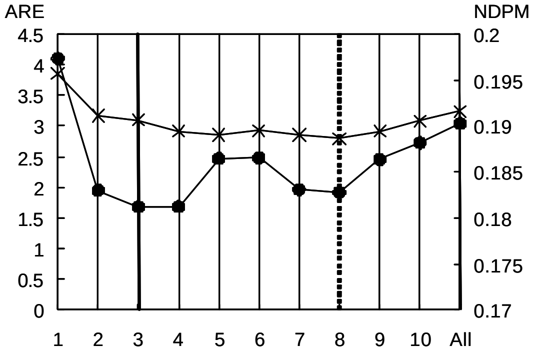
B.3. 47. DSA & AWSMN



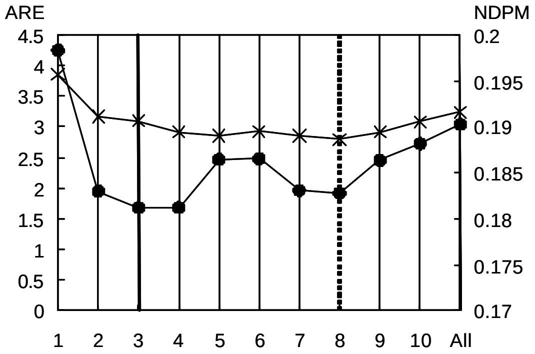
B.3. 48. DSA & AWSAR



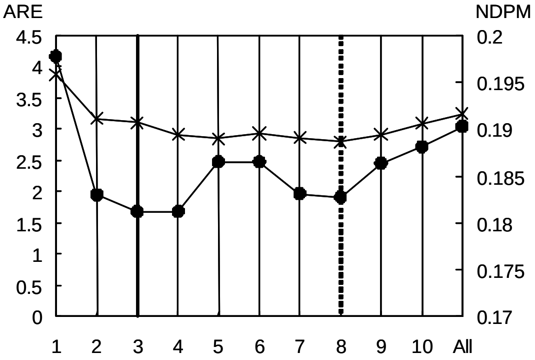
B.3. 49. DSA & AWSMR



B.3. 50. DSM & WS

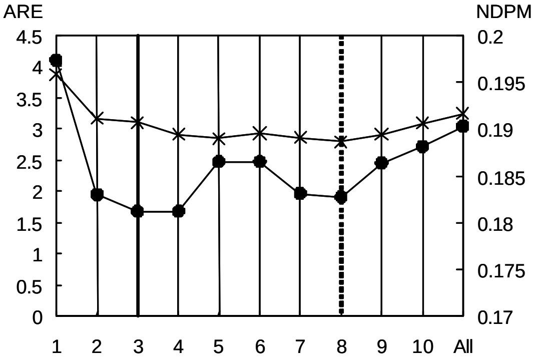


B.3. 51. DSM & AWSAC

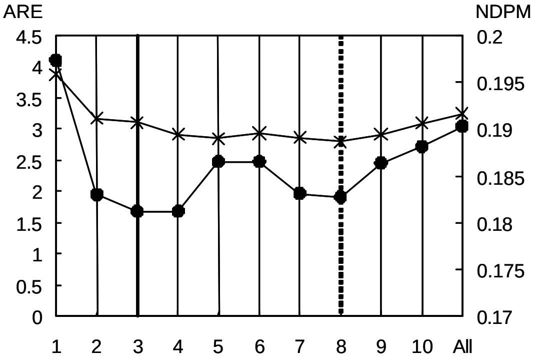


B.3. 52. DSM & AWSMC

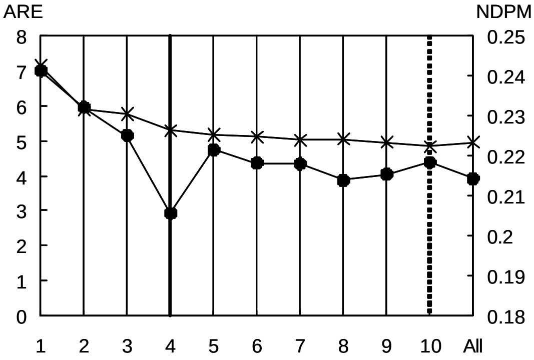
Selected Neighborhood Size for ARE
Selected Neighborhood Size for NDPM
ARE
NDPM



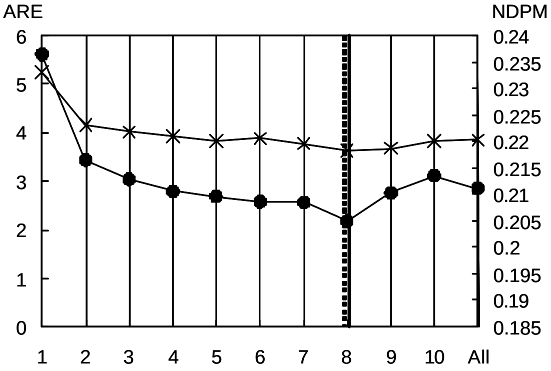
B.3. 53. DSM & AWSAN



B.3. 54. DSM & AWSMN



B.3. 55. DSM & AWSAR



B.3. 56. DSM & AWSMR

C. Detailed Evaluation Results

C.1 Detailed Result of Experiment 1: NDPM

Table 6 shows the result of Experiment 1. Each entry of the table contains a value of NDPM indicating the accuracy of recommendation which was made for the user of the corresponding column by the algorithm of the corresponding row. The rightmost column indicates the average NDPM among six users in each algorithm. The algorithms in the rows are sorted in ascending order by the average NDPMs from top to the bottom in the table. The average NDPM written with emphasized letters (italic underlined boldface) indicates the paired Student's *t*-test statistically observed significant difference to the other value written below, at the level of $p < 0.05$. In other words, the emphasized value indicates that the corresponding row's algorithm outperformed the algorithms written below.

C.2 Detailed Result of Experiment 2: ARE

Table 7 shows the calculated AREs of Experiment 2. Each entry of the table contains the average ARE indicating the accuracy of prediction made for the users of HFA of the corresponding column by the algorithm of the corresponding row. The algorithms in the rows are sorted in ascending order by the average AREs of the row from the top to the bottom of the table. The average ARE written with emphasized letters

Table 6. Detailed Result of Experiment 1: NDPM

Algorithm	NDPM of Each User						Avg. NDPM
	#1	#2	#3	#4	#5	#6	
MR & AWSAR	0.2811	0.4268	0.3373	0.3873	0.3454	0.3497	0.3546
CC & AWSAR	0.2654	0.5553	0.3288	0.4179	0.3533	0.3778	***<u>0.3831</u>***
UA	0.2702	0.5868	0.3371	0.4401	0.3532	0.3894	0.3961
UC & AWSAR	0.2701	0.5868	0.3358	0.4388	0.3664	0.4243	0.4037
Random	0.5241	0.5058	0.5244	0.5103	0.4918	0.5258	0.5137

indicates the paired Student's *t*-test statistically observed significant difference to the other value written below, at the level of $p < 0.05$.

C.3 Detailed Result of Experiment 2: NDPM

Table 8 shows the calculated NDPMs of Experiment 2. This table is written in the same manner as the table written in Appendix C.2 where the calculated NDPMs instead of the AREs are written.

C.4 Detailed Result of Experiment 3: ARE

Table 9 shows the calculated AREs of Experiment 3. Each row indicates a CF algorithm composed by a combination of a similarity computation algorithm and a prediction algorithm written in the leftmost two columns. The other right columns contain the average AREs of the corresponding row's CF algorithm where such CF algorithm used the corresponding columns' HFA usage histories. In the left of each

Table 7. Detailed Result of Experiment 2: ARE

Algorithm	ARE of Each HFA			Avg. ARE
	Word 2002	PowerPoint 2002	Excel2002	
UA	***2.3978***	***2.0812***	***1.6232***	***2.0341***
BC & AWSAR	18.1429	17.5288	10.2658	15.3125
CC & AWSAR	***18.5338***	***17.8290***	***10.5655***	***15.6428***
RC & AWSAR	23.3142	19.5541	12.0022	18.2901
MR & AWSAR	***23.6369***	***19.4104***	***12.1713***	***18.4062***
UC	614.4649	1260.2588	236.3690	703.6975

Table 8. Detailed Result of Experiment 2: NDPM

Algorithm	NDPM of Each HFA			Avg. NDPM
	Word 2002	PowerPoint 2002	Excel2002	
RC & AWSAR	***0.2842***	0.2345	0.2618	***0.2602***
MR & AWSAR	0.2858	***0.2345***	0.2619	***0.2607***
BC & AWSAR	0.2883	***0.2366***	***0.2638***	***0.2629***
UA	0.2891	0.2374	0.2666	0.2644
CC & AWSAR	***0.291***	***0.2376***	***0.2664***	***0.2650***
UC	0.3109	0.2758	0.2912	0.2926

average ARE, neighborhood sizes (*k*) selected in calculating the ARE are written. For example, the top-and-rightmost cell containing the value “1.871800913” indicates the average ARE of the combination of the Adjusted Cosine Similarity with Average and the Weighted Sum, when using the usage histories of Excel 2002. The neighborhood size *k* was 10, which is written in the left cell of “1.871800913”. The sorting and the emphasizing manner of each ARE is the same as the table written in Appendix C.2.

C.5 Detailed Result of Experiment 3: NDPM

Table 10 shows the calculated NDPMs of Experiment 3. This table is written in the same manner as the table written in Appendix C.4e same manner as the table written in Appendix C.4 where the calculated NDPMs are written instead of the AREs. For example, the top-and-rightmost cell containing the value “0.156034739” indicates the average NDPM of the combination of the Rank Correlation and the Weighted Sum, when using the usage histories of Excel 2002. The neighborhood size k was 3, which is written in the left cell of “0.156034739”.

Table 9. Detailed Result of Experiment 3: ARE

Similarity Computation Algorithm	Prediction Algorithm	Word 2002		Power Point 2002		Excel2002		Avg. ARE
		k	ARE	*k*	ARE	*k*	ARE	
ACSA	WS	1	1.527299844	10	1.197950996	10	1.871800913	1.532351
ACSA	AWSMC	1	1.527299844	0	1.03841677	0	2.052638333	1.539452
ACSA	AWSMN	1	1.527299844	0	1.03841677	0	2.052638333	1.539452
RC	WS	2	1.871011255	6	0.940172895	1	1.853342471	1.554842
RC	AWSAC	2	1.871011255	6	0.940172895	1	1.853342471	1.554842
RC	AWSMC	2	1.871011255	6	0.940172895	1	1.853342471	1.554842
RC	AWSAN	2	1.871011255	6	0.940172895	1	1.853342471	1.554842
RC	AWSMN	2	1.871011255	6	0.940172895	1	1.853342471	1.554842
ACSM	AWSMC	1	1.697091183	0	1.002602929	0	2.113788872	1.604494
ACSM	AWSMN	1	1.697091183	0	***1.002602929***	0	2.113788872	***1.604494***
ACSM	WS	10	1.659033776	10	1.16327919	10	2.056881268	1.626398
ASD	AWSAC	1	1.571011397	1	0.788076189	3	3.034641375	1.79791
ASD	AWSAN	1	1.571011397	1	0.788076189	3	3.034641375	1.79791
ASD	AWSMN	1	1.571011397	1	0.788076189	3	3.034641375	1.79791
ASD	WS	1	1.571011397	1	0.788076189	3	3.034641375	1.79791
ASD	AWSMC	1	1.571011397	1	0.788076189	3	3.034641375	1.79791
ACSM	AWSAN	1	1.697091183	9	1.301888086	2	2.604740186	1.867906
ACSM	AWSAC	1	1.697091183	9	1.302705597	2	2.604740186	1.868179
ACSA	AWSAN	1	1.527299844	9	1.34563553	3	2.733597381	1.868844
ACSA	AWSAC	1	***1.527299844***	0	1.348265681	3	2.733002907	***1.869523***
CCM	WS	2	2.166792582	8	1.075290188	3	2.908669402	2.050251
CCM	AWSMC	2	2.166792582	8	1.075290188	3	2.908669402	2.050251
CCM	AWSAN	2	2.166792582	8	1.075290188	3	2.908669402	2.050251
CCM	AWSMN	2	2.166792582	8	1.075290188	3	2.908669402	2.050251
CCM	AWSAC	2	2.166792582	8	1.075290188	3	2.908669402	2.050251
CS	WS	2	2.226395848	8	1.079794754	3	2.892912748	2.066368
CS	AWSAC	2	2.226395848	8	1.079794754	3	2.892912748	2.066368
CS	AWSMC	2	2.226395848	8	1.079794754	3	2.892912748	2.066368
CS	AWSAN	2	2.226395848	8	1.079794754	3	2.892912748	2.066368
CS	AWSMN	2	2.226395848	8	1.079794754	3	2.892912748	2.066368
RC	AWSMR	2	2.455606019	2	1.006964551	3	2.780492463	2.081021
CC	AWSAC	2	2.353294107	7	1.080598324	3	2.8850008	2.106298
CC	AWSMC	2	2.353294107	7	1.080598324	3	2.8850008	2.106298
CC	AWSAN	2	2.353294107	7	1.080598324	3	2.8850008	2.106298
CC	AWSMN	2	2.353294107	7	1.080598324	3	2.8850008	2.106298
CC	WS	2	2.353294107	7	1.080598324	3	2.8850008	2.106298
ASD	AWSMR	3	1.794129162	1	0.942044553	8	3.65039598	2.128857
MSD	WS	6	3.009644569	2	0.814343325	8	2.793375757	2.205788
MSD	AWSAC	6	3.009644569	2	0.814343325	8	2.793375757	2.205788
MSD	AWSMC	6	3.009644569	2	0.814343325	8	2.793375757	2.205788
MSD	AWSAN	6	3.009644569	2	0.814343325	8	2.793375757	2.205788
MSD	AWSMN	6	3.009644569	2	0.814343325	8	2.793375757	2.205788
ACSM	AWSMR	2	1.877346099	0	1.226419591	4	3.703519889	2.269095
ACSA	AWSMR	10	1.901390083	10	1.283558463	3	3.629309573	2.271419
CS	AWSMR	5	2.045663731	8	1.165067875	9	3.637096479	2.282609
CCM	AWSMR	5	2.059370841	8	1.163040415	9	3.635689017	2.286033
CC	AWSMR	4	2.046978972	7	1.190034576	8	3.647900115	2.294971
MSD	AWSMR	0	3.171170293	1	1.038370746	8	3.64351212	2.617684
–	UA	–	3.323785323	–	2.468225192	–	***2.696767408***	***2.829593***
CCM	AWSAR	5	3.067383114	7	1.761033888	9	4.781150052	3.203189
CS	AWSAR	5	3.078850363	7	1.767189202	9	4.812251478	3.21943
CC	AWSAR	5	3.044225633	10	1.822700153	8	4.798880066	3.221935
ASD	AWSAR	3	3.332343064	0	1.877218246	0	5.085645564	3.431736
ACSM	AWSAR	2	3.350644172	5	2.152147036	2	4.926534028	3.476442
ACSA	AWSAR	4	3.374610122	5	2.184205949	3	5.121541469	3.560119
RC	AWSAR	5	4.085148681	9	1.843856098	8	4.783344952	3.570783
MSD	AWSAR	0	***4.289858113***	0	***1.844689871***	10	***4.826765367***	***3.653771***
–	Random	–	***639.9408194***	–	1319.160835	–	***305.120526***	754.7407
–	UC	–	795.2954119	–	994.5097597	–	499.4964732	763.1005

Table 10. Detailed Result of Experiment 3: NDPM

Similarity Computation Algorithm	Prediction Algorithm		Word 2002		Power Point 2002		Excel2002	
		k	NDPM	*k*	NDPM	*k*	NDPM	Avg. NDPM
RC	WS	5	0.178345952	5	0.157157881	3	0.156034739	0.163846
RC	AWSAC	5	0.178345952	5	0.157157881	3	0.156034739	0.163846
RC	AWSMC	5	0.178345952	5	0.157157881	3	0.156034739	0.163846
RC	AWSAN	5	0.178345952	5	0.157157881	3	0.156034739	0.163846
RC	AWSMN	5	0.178345952	5	***0.157157881***	3	***0.156034739***	***0.163846***
MSD	WS	5	0.195382302	7	0.164847524	3	0.18118463	0.180471
MSD	AWSAC	5	0.195382302	7	0.164847524	3	0.18118463	0.180471
MSD	AWSMC	5	0.195382302	7	0.164847524	3	0.18118463	0.180471
MSD	AWSAN	5	0.195382302	7	0.164847524	3	0.18118463	0.180471
MSD	AWSMN	5	0.195382302	7	0.164847524	3	0.18118463	0.180471
ASD	WS	6	0.205436187	7	0.165914561	8	0.18876585	0.186706
ASD	AWSAC	6	0.205436187	7	0.165914561	8	0.18876585	0.186706
ASD	AWSMC	6	0.205436187	7	0.165914561	8	0.18876585	0.186706
ASD	AWSAN	6	0.205436187	7	0.165914561	8	0.18876585	0.186706
ASD	AWSMN	6	0.205436187	7	0.165914561	8	0.18876585	0.186706
CC	WS	0	0.208730345	0	0.168026978	10	0.190454586	0.189071
CC	AWSAC	7	0.208887856	0	0.168026978	10	0.190454586	0.189123
CC	AWSMC	7	0.208887856	0	0.168026978	10	0.190454586	0.189123
CC	AWSAN	7	0.208887856	0	0.168026978	10	0.190454586	0.189123
CC	AWSMN	7	0.208887856	0	0.168026978	10	0.190454586	0.189123
CCM	WS	7	0.20815032	9	0.168233325	10	0.191309623	0.189231
CCM	AWSAC	7	0.20815032	9	0.168233325	10	0.191309623	0.189231
CCM	AWSMC	7	0.20815032	9	0.168233325	10	0.191309623	0.189231
CCM	AWSAN	7	0.20815032	9	0.168233325	10	0.191309623	0.189231
CCM	AWSMN	7	0.20815032	9	0.168233325	10	0.191309623	0.189231
CS	WS	7	0.208655822	9	0.168081938	10	0.191199341	0.189312
CS	AWSAC	7	0.208655822	9	0.168081938	10	0.191199341	0.189312
CS	AWSMC	7	0.208655822	9	0.168081938	10	0.191199341	0.189312
CS	AWSAN	7	0.208655822	9	0.168081938	10	0.191199341	0.189312
CS	AWSMN	7	0.208655822	9	0.168081938	10	0.191199341	0.189312
RC	AWSMR	7	0.221721456	8	0.178050824	6	***0.182641971***	***0.194138***
MSD	AWSMR	6	0.208975944	8	0.174892109	8	0.204977856	0.196282
ACSM	AWSAC	8	0.221593687	0	0.173231161	7	0.195132286	0.196652
ACSA	AWSAC	8	0.221428627	0	0.174799059	7	0.194681001	0.19697
ACSA	AWSAN	6	0.222001074	0	0.174799059	5	0.196657715	0.197819
ACSM	AWSAN	8	0.223381372	0	***0.173231161***	7	0.197674231	***0.198096***
ACSM	AWSMC	5	0.223092828	5	0.182281224	0	0.193186435	0.19952
ACSM	AWSMN	5	0.223978394	5	0.181829388	0	0.193186435	0.199665
ACSM	WS	5	0.225246164	5	0.183950057	6	0.198610257	0.202602
ACSA	AWSMC	6	0.223771973	6	0.188397964	7	0.195824642	0.202665
ACSA	AWSMN	6	0.224587394	6	0.188182553	5	0.195496944	0.202756
ACSA	WS	3	0.224807105	4	0.188619023	3	0.197825729	0.203751
ASD	AWSMR	7	0.222121772	7	0.177661519	8	0.212296164	0.204026
MSD	AWSAR	7	0.220849201	6	0.190895427	4	0.205519532	0.205755
CC	AWSMR	9	0.234564807	9	0.184095277	10	0.216417721	0.211693
CCM	AWSMR	7	0.235414788	9	0.185088841	10	0.216151782	0.212218
CS	AWSMR	10	0.235912426	9	***0.185250606***	10	0.21600504	***0.212389***
ASD	AWSAR	6	0.231862404	8	0.201455765	8	0.208831199	0.21405
CC	AWSAR	0	0.240647534	6	***0.215805847***	8	0.214825541	0.22376
RC	AWSAR	8	***0.233534192***	10	0.245972244	6	***0.198731862***	***0.226079***
ACSM	AWSMR	3	0.252151783	5	0.201003323	6	0.228354416	0.22717
ACSA	AWSMR	3	0.248292742	5	0.205041777	5	0.229246043	0.227527
CCM	AWSAR	0	0.244116879	10	0.23536655	10	0.215861842	0.231782
CS	AWSAR	0	0.243973137	9	0.235963669	10	0.215783083	0.231907
–	UA	–	0.271476443	–	0.223442063	–	***0.209384784***	***0.234768***
ACSM	AWSAR	4	0.263874148	3	0.25741506	4	0.246929283	0.256073
ACSA	AWSAR	4	***0.259560093***	3	0.265621045	3	0.243441405	0.256208
–	UC	–	***0.337214222***	–	***0.266874878***	–	***0.255487634***	***0.286526***
–	Random	–	0.517314685	–	0.499935814	–	0.503179714	0.50681