Listwise Collaborative Filtering with High-Rating-Based Similarity and Simple Missing Value Estimation

Yoshiki TSUCHIYA* · Hajime NOBUHARA*

In this paper, we make two proposals. The first aims to accelerate similarity calculations by only using a subset of the rating information (namely the highest ratings), while the second attempts to improve the accuracy of listwise collaborative filtering using a simple missing value estimation process. Experiments using the MovieLens 1M (6,040 users, 3,952 items and 1,000,209 ratings), 10M (71,567 users, 10,681 items and 10,000,054 ratings) and Jester (48,483 users, 100 items and 3,519,448 ratings) datasets demonstrate that these proposals can considerably reduce the computation time (by a factor of up to 50) and improve the normalized discounted cumulative gain value by up to 0.02 compared with ListCF, a well-known listwise collaborative filtering algorithm.

Keywords: recommender system, ranking-oriented collaborative filtering, high-rating-based similarity, missing value estimation

1. Introduction

In recent years, as the Web has developed, recommender systems have become increasingly important and are being used for a wide range of applications, leading many researchers to focus on recommendation technologies and systems [1–5]. Collaborative filtering (CF), a widely used recommendation algorithm, is based on assessing the similarity of users or items, calculated using a user-rating matrix. Various CF algorithms have been proposed, which can be divided into two types: rating-oriented [2, 4] and ranking-oriented [1, 3, 5], as shown in Fig. 1.

Rating-oriented algorithms, such as item-based CF [4], predict ratings for items that users have not evaluated and use these to make recommendations. In contrast, ranking-oriented CF predicts item rankings based on user similarity and makes recommendations accordingly. We will focus on the latter method here due to its higher performance [1].

Ranking-oriented CF algorithms can be further divided into two types: pairwise [3, 5] and listwise [1] approaches. Pairwise algorithms predict the ordering of item pairs but have high computational costs. In contrast, listwise methods predict the ordering of the complete item list. Although these are more accurate than typical pairwise CF algorithms, calculating the required similarities is still computationally intensive and there is scope to improve their ranking accuracy. In this paper, we propose an efficient listwise ranking-oriented CF algorithm that is both faster and more accurate than current approaches.

The proposed method implements two improvements. First, when calculating user similarities, it only considers their highest-rated items, greatly speeding up the calculations. Second, it

* Department of Intelligent Interaction Technologies, University of Tsukuba

Fig. 1 Collaborative filtering algorithm types

creates simple estimates of missing values when making ranking predictions.

Experimental comparisons between the proposed methods and a conventional CF algorithm using the MovieLens 1M (6,040 users, 3,952 movies, 1,000,209 ratings), 10M (71,567 users, 10,681 movies, 10,000,054 ratings), and Jester (48,483 users, 100 items, 3,519,448 ratings) datasets confirm that they reduce the similarity computation time by a factor of about 50 and improve the ranking accuracy by 0.02.

This paper is an extended version of a previous study [6] that conducted experiments using only the MovieLens 1M and 10M datasets. Here, we also consider the Jester dataset, which is of a different type to the MovieLens datasets in that it is dense and includes a wide range of rating scores.

This paper is organized as follows. Section 2 reviews ListCF, the well-known ranking-oriented listwise CF algorithm, as a related work. Section 3 describes the high-rating-based similarity calculation and simple missing value estimation processes for listwise CF. Section 4 describes the experiments and discusses the weaknesses they identified in our methods. Finally, Section 5 presents our conclusions.
Step 1. Input user and rating matrix

| User | Item1 | Item2 | Item3 |
|------|-------|-------|-------|
| u    | 4     | 3     | 5     |
| v    | 2     | 3     | 4     |

Step 2. Calculate probability distribution

Probability model for top-k permutations calculated based on the probability distribution of permutations of items calculated from the user and rating matrix.

Step 3. Calculate KL-divergence

\[ D_{KL}(P_u\|P_v) = \sum_{g_k \in g_k^{u,v}} P_u(g) \log \left( \frac{P_u(g)}{P_v(g)} \right). \] 

The KL divergence is asymmetric, but ListCF defines the similarity \( s(u,v) \) to be symmetric, as follows:

\[ s(u,v) = 1 - \frac{1}{2} [D_{KL}(P_u\|P_v) + D_{KL}(P_v\|P_u)]. \] 

2.2 Similarity Calculation of ListCF

In ListCF, the similarity of a pair of users \( u \) and \( v \) is calculated based on item permutation probability distributions for each user, calculated using the Plackett–Luce model [7], which is a standard permutation probability model. Fig. 2 shows how the similarities are calculated. Let \( I = \{i_1, i_2, \ldots, i_n\} \) be the complete set of items and \( \pi' = (\pi'_1, \pi'_2, \ldots, \pi'_n) \) be an ordered list of items, where the cardinal number of item set \( I \) should be equal to the number of elements of ordered list \( \pi' \), that is, \( n = m \). \( \pi'_j \in I \) and \( \pi'_j \neq \pi'_j \) if \( j \neq k \), and let \( \Omega' (\subset \Omega^I) \) be the set of all possible permutations of \( I \). Given a list of item rating scores \( \{r_{\pi'_1}, r_{\pi'_2}, \ldots, r_{\pi'_m}\} \) (e.g., integers between 1 and 5 for MovieLens 1M), where \( r_{\pi'_j} \) is the score for \( \pi'_j \), the probability of \( \pi' \), \( P(\pi') \), is defined using an increasing and strictly positive function \( \Phi(\cdot) \geq 0 \) as follows:

\[ P(\pi') = \prod_{j=1}^{n} \phi(r_{\pi'_j}) \in [0,1]. \tag{1} \]

Here, we have chosen to define this function as \( \Phi(r) = e^r \).

One issue with this approach is that it requires \( n! \) different permutations of the \( n \) items to be considered, which would take a long time to compute. To speed up this process, the top-k probability model \( g_k \) [1] is introduced, as follows:

\[ g_k (i_1, i_2, \ldots, i_k) = \left\{ \pi' \mid \pi' \in \Omega', \pi'_j = i_j, j = 1, \ldots, k, l = 1, \ldots, n \right\} \]

and the probabilities of the top-k permutations are calculated as

\[ P(g_k (i_1, i_2, \ldots, i_k)) = \prod_{j=1}^{k} \phi(r_{\pi'_j}) \in [0,1]. \]

\[
\forall j = 1, \ldots, k : \pi'_j = i_j.
\]

A previous experiment [1] used \( k = 1 \). Let \( g_k^u \ (\subset \Omega^I) \) be the set of top-k permutations of \( I \), and let the probabilities of these permutations form the probability distribution. Then, define \( I_{u,v} \ (\subset I) \) as the set of items rated by both users \( u \) and \( v \), and \( P_u \) and \( P_v \ (\in [0,1]) \) as the probability distributions over \( g_k^{u,v} \ (\subset \Omega^u,v) \) calculated by Eq. (3) based on the users' rating scores. The similarity score is now obtained from the Kullback–Leibler (KL) divergence [9] of \( P_u \) and \( P_v \), which is calculated as

\[ D_{KL}(P_u\|P_v) = \sum_{g_k \in g_k^{u,v}} P_u(g) \log \left( \frac{P_u(g)}{P_v(g)} \right). \tag{4} \]

The KL divergence is asymmetric, but ListCF defines the similarity \( s(u,v) \) to be symmetric, as follows:

\[ s(u,v) = 1 - \frac{1}{2} [D_{KL}(P_u\|P_v) + D_{KL}(P_v\|P_u)]. \tag{5} \]

If the set \( I_{u,v} \) only includes a few items, the similarity will inevitably be high, so this is compensated for by multiplying the similarity function by \( \min[I_{u,v} \cap I_u, 1] \), where \( c \) is a threshold. Each user's neighborhood users can then be found from the similarities calculated using Eqs. (3)–(5).

2.3 Ranking Prediction of ListCF

The flow of ListCF's ranking prediction process is shown on the left-hand side of Fig. 4. Let \( B \) be the set of users, \( N_u (\subset U) \) be the set of user \( u \)'s neighborhood users, and \( T_{u,v} (\subset I \cap I_u) \) be the set of items whose ranks are to be predicted. Let \( P_{u,v} (\in [0,1]) \) be the probability distribution of the top-k permutations \( g_k^{u,v} (\subset \Omega^{u,v}) \) of \( T_{u,v} \), defined as

\[ P_{u,v}(g) = \frac{\varphi_{u,g}}{\sum_{g' \in g_k^{u,v}} \varphi_{u,g'}}. \tag{6} \]

where \( \varphi_{u,g} \mid \forall g \in g_k^{u,v} \) are unknown variables assigned to the top-k permutations.

In ListCF, the cross entropy is used as a loss function for prediction. Consider a target user \( u \), for whom we want to rank a set of items \( T_{u,v} \), and a neighborhood user \( v \in N_u \). Let the set of items rated by \( v \) be \( I_v \), with \( T_{u,v} \cap I_v \) and \( g_k^{u,v} (\subset \Omega^{u,v}) \) be the set of top-k permutations of \( T_{u,v} \). The cross entropy \( E \) is calculated using the probability distributions \( P_{u,v} \) and \( P' \) over \( g_k^{u,v} \), as follows:

\[ E \left( P_{u,v}, P' \right) = - \sum_{g \in g_k^{u,v}} P'_u(g) \log P'_{u,v}(g). \tag{7} \]

ListCF makes predictions by minimizing the following weighted cross entropy sum:
arg min \sum_{v \in N_u} s(u, v) \cdot E \left( P_u', P_v' \right), \tag{8}
\text{s.t.} \forall g' \in g_{k_u}^{T_u} : \varphi_{u,g} \geq 0.

The objective function \( F(\varphi_u) \) in Eq. (8) is then transformed as follows, using Eqs. (6) and (7):
\[
\begin{align*}
F(\varphi_u) &= \sum_{v \in N_u} s(u, v) \cdot E \left( P_u', P_v' \right) \\
&= -\sum_{v \in N_u} s(u, v) \sum_{g \in g_{k_u}^{T_u}} P_v'(g) \log_2 \left( P_u'(g) \right) \\
&= -\sum_{v \in N_u} s(u, v) \sum_{g \in g_{k_u}^{T_u}} P_v'(g) \log_2 \left( \frac{\varphi_{u,g}}{\sum_{g' \in g_{k_u}^{T_u,v}} \varphi_{u,g'}} \right) \\
&= \sum_{v \in N_u} \left( s(u, v) \sum_{g \in g_{k_u}^{T_u,v}} P_v'(g) \right) \log_2 \left( \sum_{g' \in g_{k_u}^{T_u,v}} \varphi_{u,g'} \right) - \sum_{v \in N_u} s(u, v) P_v'(g) \log_2 \left( \frac{\varphi_{u,g}}{\sum_{g' \in g_{k_u}^{T_u,v}} \varphi_{u,g'}} \right).
\end{align*}
\] (9)

Equation (9) is then optimized using the gradient descent method. Partially differentiating \( F \) with respect to \( \varphi_{u,g} \), we obtain
\[
\frac{\partial F}{\partial \varphi_{u,g}} = \sum_{v \in N_u} \left( \frac{s(u, v) \sum_{g \in g_{k_u}^{T_u,v}} P_v'(g)}{\ln 2 \cdot \sum_{g \in g_{k_u}^{T_u,v}} \varphi_{u,g}} \right) - \frac{s(u, v) P_v'(g)}{\ln 2 \cdot \varphi_{u,g}}.
\] (10)

For a given learning rate \( \eta \), \( \varphi_{u,g} \) is updated as follows:
\[
\varphi_{u,g}^{t+1} = \varphi_{u,g}^t - \eta \frac{\partial F}{\partial \varphi_{u,g}}.
\] (11)

### 3. Proposed Method

#### 3.1 Rapid High-Rating-Based Similarity Calculation

Since ListCF calculates the similarity based on permutations and all the rating information, it is very computationally intensive. Since the purpose of recommender systems is to recommend items that users will rate highly, we propose to address this issue by instead focusing only on the similarity of highly rated items to achieve this goal.

Fig. 3 shows the proposed similarity calculation procedure. Let \( H_u \subset I_u \) be the set of items that the target user \( u \) has rated highly, and \( H_{uv} \subset I_{uv} \) be the set of items that both \( u \) and \( v \) have rated highly. We then define the similarity between \( u \) and \( v \) as follows:
\[
similarity(u, v) = \frac{|H_{uv}|}{|H_u|} \quad (\in [0, 1]).
\] (12)

Since \( \similarity(u, v) \) and \( \similarity(v, u) \) have different denominators, the similarity matrix is asymmetric. In addition, we define \( \similarity(uv) = 0 \) if \( u = v \) so as not to add users to their own neighborhoods during ranking prediction. The choice of high rating threshold has a significant impact on the ranking accuracy, as discussed in Section 4.2. If this threshold is set too high, then the sets of highly rated items produced will often be empty, meaning the user similarities cannot be calculated accurately.

---

**Fig. 3** High-rating-based similarity calculation procedure

**Fig. 4** Example of unhelpful optimization by ListCF

#### 3.2 Simple Missing Value Estimation

We also propose to improve ListCF’s ranking accuracy using simple estimates of missing values. In ListCF, the cross entropy (Eq. (7)) is calculated from the probability distribution of \( g_{k_u}^{T_u,v} \), the set of permutations of \( T_{uv} \). However, it may fail to optimize the objective function if \( T_{uv} \) has too few elements, e.g., if it only includes two items, both of which received low ratings from the neighborhood user \( v \) (Fig. 4). In this case, despite the items’ low ratings, the optimization process generates high item ranks, which are unhelpful for user \( u \). In Fig. 4, although the neighborhood user gave Item 2 a low rating, after optimization that item is very likely to be ranked first.

To deal with this issue, we propose to first give temporary estimated ratings to items that the neighborhood user \( v \) has not rated before making ranking predictions. Let \( r_{u,i} \) be the rating given by user \( u \) to item \( i \), and let unrated items have a score of zero. For a given neighborhood user \( v \) in \( N_u \), define \( v \)'s temporary ratings for the items not rated by \( u \) (i.e., the items in \( T_u \)) as
\[
temporary rating (v, t_j) = \begin{cases} r_{v,t_j} & \text{if } r_{v,t_j} \neq 0 \\ \sum_{\nu' \in N_{v,t_j}} r_{v',t_j} & \text{if } r_{v,t_j} = 0, \quad (j = 1, \ldots, p), \end{cases}
\] (13)

where the set \( N_{v,t_j} \) represents \( v \)'s neighborhood users \( v' \in N_v \) who have rated item \( t_j \). If none of \( v \)'s neighborhood users have rated \( t_j \), the temporary rating will still be zero.

In this case, we obtain a nonzero temporary rating by calculating temporary rating \( (v', t_j) \) for each neighborhood user \( v' \) of \( v \).
If we calculate the cross entropy (Eq. (7)) using these temporary ratings (Eq. (13)) and replace \( s(u, v) \) (Eq. (5)) by similarity \( \text{similarity}(u, v) \) (Eq. (12)), we can then make ranking predictions in the same way as for ListCF by using Eqs. (8)–(11). Adding temporary ratings makes the neighborhood users’ probability distributions more reasonable, leading to fewer undesirable parameter updates and hence (hopefully) improved ranking accuracy. Fig. 6 compares the procedures for ListCF and the proposed method.

4. Experimental Evaluation

4.1 Overview of the Experiment

To confirm the effectiveness of the two proposed changes, we compared the proposed method with ListCF experimentally in terms of computation time and ranking accuracy using the MovieLens (1M and 10M) and Jester datasets, details of which are shown in Table 1. Here, the sparsity is calculated as follows:

\[
\text{Sparsity} = 1 - \frac{\text{No. of ratings}}{\text{Users} \times \text{Items}}
\] (14)

As can be seen from the Tables 1, both MovieLens datasets are very sparse, while the Jester dataset is very dense and has a wide range of rating scores. Figs. 7-9 show the distributions of ratings for each data set.

The holdout approach was already used in the experiment of the conventional method [1], therefore we also employ holdout approach in the experiment. For each dataset, we selected 10 ratings from each user to create the test dataset and used the remainder as training data. Since it has been shown that increasing the value of \( k \), where the parameter \( k \) means top-\( k \) ranking of recommendation, does not improve the ranking accuracy significantly [1], we chose the top-1 probability model for our ListCF implementation with a similarity threshold \( c \) explained in Section 2.2 of 300. We conducted preliminary experiments with \( c \) at several values and used the most accurate values for comparison with the proposed

| Table 1 | Details of datasets |
|---------|---------------------|
| MovieLens 1M | MovieLens 10M | Jester |
| Users | 6,040 | 71,567 | 48,483 |
| Items | 3,952 | 10,681 | 100 |
| Ratings | 1,000,209 | 10,000,054 | 3,519,448 |
| Rating | Integer | Real with increments of 0.5 | Real |
| range | (1 to 5) | (-10 to 10) |
| Sparsity | 0.958 | 0.987 | 0.274 |

Fig. 5 Missing value estimation procedure

Fig. 6 Procedures for ListCF and the proposed method

Fig. 7 The distribution of rating value of MovieLens 1M

Fig. 8 The distribution of rating value of MovieLens 10M

Fig. 9 The distribution of rating value of Jester
method.

We used the normalized discounted cumulative gain (NDCG) [10, 11] metric to assess ranking prediction accuracy. The NDCG, based on the top n predicted rankings for user u, is defined as follows:

$$\text{NDCG}_u @ n = \sum_{p=1}^{n} \frac{2^{r_u^p} - 1}{\log_2 (1 + p)}$$

where $Z_u$ is a normalization term that ensures the correct ranking has an NDCG value of 1 and $r_u^p$ is the rating of the pth-ranked item for user u. For a given set of users U, the overall NDCG@n score is

$$\text{NDCG}@n = \frac{1}{|U|} \sum_{u=1}^{n} \text{NDCG}_u @ n.$$  

Since the Jester dataset includes negative ratings, we added 10 to all the Jester ratings when carrying out the NDCG calculations.

In this paper, the conventional method and the proposed method have been implemented in C++ language. Also, all experiments described in this paper have been performed on a computer with 4 core 3.60 GHz CPU and 32 GB RAM.

### 4.2 Similarity Computation Time Comparison

In this experiment, we compared the time taken to calculate the user similarities using both the proposed and ListCF methods, and then compared the accuracy of the ListCF ranking predictions made using these similarities.

For the MovieLens 1M and 10M datasets, we chose a high rating threshold of 5 because, after testing with thresholds ranging between 1 and 5, this gave the most accurate results. For the Jester dataset, we chose a high rating threshold of 0.0 because higher values caused a sharp increase in the number of cases where the user similarity could not be calculated.

Figs. 10(a)–(c) show the computation times for the MovieLens (1M and 10M) and Jester datasets, respectively, while Figs. 11(a)–(c) show the corresponding ranking accuracy results. The horizontal axes indicate the similarity calculation method, while the vertical axes indicate the computation time (Fig. 10) or NDCG@5 score (Fig. 11). These show that the proposed method reduced the computation time considerably, by factors of about 15, 50, and 4 for MovieLens 1M, MovieLens 10M, and Jester, respectively, while achieving similar ranking accuracies. In other words, the proposed method was able to substantially reduce the computation time while maintaining accuracy.

### 4.3 Comparison of Ranking Accuracy

Next, we conducted experiments to examine the effect of our simple missing value estimation method on ranking accuracy. We compared the accuracy of the ranking predictions made by both the proposed and ListCF prediction methods, based on the similarities calculated by both methods in Section 4.2. We used an initial $q_{u,g}$ value of 10 and 100 neighborhood users for all datasets, together with learning rates of 0.025, 0.01, and 0.1 for MovieLens 1M, MovieLens 10M, and Jester respectively. In the reference [6] (actually, this reference is conference version of this manuscript), the results of experiments with $n = 1, 3,$ and 5 have been shown, and we recognized all results are almost same. So we omitted the experiment results other than $n = 5$.

Figs. 12(a)–(c) show ranking accuracy comparisons for the proposed and ListCF methods on all three datasets. The horizontal axes indicate the similarity calculation method, while the vertical axes indicate the NDCG@5 score. In addition, the red and blue bars indicate the ranking predictions made by the ListCF and proposed methods, respectively. These charts show that the proposed method improved the ranking accuracy in almost all cases, achieving a maximum improvement of 0.02.

That said, applying the proposed ranking prediction method to the results of the proposed similarity calculation approach in Jester dataset caused the accuracy to decrease by 0.003. In this case the missing value estimation process was unstable because only a small amount of data was used for the similarity calculations and the Jester dataset’s rating values covered a wide range.

### 5. Discussion

These experimental results confirm the effectiveness of the two proposed methods. At the same time, however, they also highlight several of their weaknesses.

First, with the high-rating-based similarity approach, setting the threshold of high rating to too high value makes it impossible to calculate many similarities, as increasing the threshold reduces the size of the resulting item set. Since, as can be seen from Eq. (12), the number of items in these sets is used by the proposed method to calculate the similarities, empty item sets mean the similarities cannot be calculated. Selecting an appropriate threshold is particularly important for datasets with wide rating score ranges (such as the Jester dataset). However, since there are only a modest number of possible thresholds, good values can quickly be found by experiment.

In addition, as mentioned in Section 4.3, the simple missing value estimation process becomes more difficult for wide rating score ranges, lowering the ranking accuracy in some cases. We will consider more robust estimation methods in our future work.

### 6. Conclusion

In this paper, we have made two proposals. The first is to accelerate user similarity score computation by using only high ratings, instead of all ratings. The second is to improve ranking accuracy by adding simple estimates for missing values, i.e., items that have not been rated by neighborhood users.

We then conducted experiments using the MovieLens (1M and 10M) and Jester datasets to compare the computation time and ranking accuracy of both proposals with those of conventional ListCF. These demonstrated that high-rating-based similarity could considerably reduce the computation time without loss of ranking accuracy, while the missing value estimates generally improved the ranking accuracy.

In the future, we plan to improve the way neighborhood users are selected, search for better objective functions, and consider more robust estimation methods.
The comparison of similarity calculation time of MovieLens 1M (a), MovieLens 10M (b) and Jester (c)

The comparison of ranking accuracy of MovieLens 1M (a), MovieLens 10M (b) and Jester (c)

References

[1] S. Huang, S. Wang, Tie-Yan Liu, J. Ma, Z. Chen, and J. Veijalainen: “Listwise Collaborative Filtering”, Proc. of the 38th Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, Aug. 09-13, Santiago, pp. 343-352, 2015.

[2] G. Linden, B. Smith, and J. York: “Amazon.com recommendations: item-to-item collaborative filtering”, IEEE Internet Computing, Vol.7, No.1, pp. 76-80, 2003.

[3] N. N. Liu and Q. Yang: “EigenRank: a ranking-oriented approach to collaborative filtering”, Proc. of the 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, July 20-24, Singapore, pp. 83-90, 2008.

[4] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl: “Item-based collaborative filtering recommendation algorithms”, Proc. of the 10th Int. conf. on World Wide Web, May 01-05, Hong Kong, pp. 285-295, 2001.

[5] S. Wang, J. Sun, B. J. Gao, and J. Ma, “VSRank: A Novel Framework for Ranking-Based Collaborative Filtering”, ACM Trans. Intell. Syst. Technol., Vol.5, No.3, Article No. 51, 2014.

[6] Y. Tsuchiya and H. Nobuhara: “Improved Listwise Collaborative Filtering with High-Rating-Based Similarity and Temporary Ratings,” Proc. of the ACM IUI Workshops, Mar. 11, Tokyo, 2018.

[7] J. I. Marden: Analyzing and modeling rank data, CRC Press, 1996.

[8] Z. Cao, T. Qin, Tie-Yan Liu, Ming-Feng Tsai, and H. Li: “Learning to rank: from pairwise approach to listwise approach,” Proc. of the 24th Int. Conf. on Machine learning, June 20-24, Corvallis, pp. 129-136, 2007.

[9] S. Kullback: Information Theory and Statistics, Courier Corporation, 1997.
Listwise Collaborative Filtering with High-Rating-Based Similarity and Simple Missing Value Estimation

[10] K. Järvelin and J. Kekäläinen: “IR evaluation methods for retrieving highly relevant documents,” Proc. of the 23rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, July 24-28, Athens, pp. 41-48, 2000.

[11] K. Järvelin and J. Kekäläinen: “Cumulated gain-based evaluation of IR techniques,” ACM Trans. Inf. Syst. Vol.20, No.4, pp. 422-446, 2002.

(2018年4月15日受付)
(2018年6月21日採録)

[Contact Address]
Department of Intelligent Interaction Technologies, University of Tsukuba
1-1-1 Tennoudai, Tsukuba-City, Ibaraki 305-8573, Japan
Hajime NOBUHARA
E-mail: nobuhara@cmu.iit.tsukuba.ac.jp

Information about Author

Yoshiki TSUCHIYA [non-member]
Yoshiki Tsuchiya graduated from Faculty of Engineering at Shibaura Institute of Technology in 2017. He is currently studying in Master’s Program in Intelligent Interaction Technologies at University of Tsukuba.

Hajime NOBUHARA [non-member]
Hajime Nobuhara graduated from Tokyo Institute of Technology in 2002. He is currently Associate Professor, Intelligent Interaction Technologies at University of Tsukuba.