Bookshelf Problem: A Human-in-the-Loop Approach for Data Grouping without Complete Information

Emi Sakurai[1], Atsuyuki Morishima[1], Kosetsu Ikeda[1], Nobutaka Suzuki[1]
[1]University of Tsukuba, Japan

Abstract

The problem of dividing a given set of data items into groups in the situation that the given input is not su cient to solve it has a wide range of applications. However, the problem cannot be solved by computers alone. This paper defines the Bookshelf problem to deal with such a problem and discusses how to solve the problem with the help of humans. Intuitively, the Bookshelf problem is as follows. Given a set of books with tags and a book cabinet with *N* shelves, we need to construct *N* groups of books s.t. all books in each group share at least one common tag. However, the given tags and their connections to books may not be su cient to make groups, and we have to find the missing tags and connections. This paper proposes a systematic human-in-the-loop method that uses two types of microtasks to solve the problem, and experimentally shows that human intelligence is e ective to avoid the worst-case search.

Keywords: crowdsourcing; human computation; data management

doi: 10.9776/16194

Acknowledgements: The authors are grateful to the contributors to Crowd4U, whose names are partially listed at http://crowd4u.org. This research was partially supported by the Grant-in-Aid for Scientific Research (#25240012) from MEXT, Japan.

Contact: emi.sakurai.2013b@mlab.info

1 Introduction

This paper defines the Bookshelf problem to model a problem of dividing a given set of data items into groups without information su cient to solve the problem at the beginning, and proposes a human-in-the-loop method that uses microtasks. The Bookshelf problem is easy to understand with the following example. We have a set of four books that are tagged as in Figure 1 and are going to put the four books into a book cabinet with two shelves. Since it has two shelves, we need to divide them into two groups. For easy access, books on the same shelf have to have at least one common tag and the number of the books on a shelf needs to be balanced. Notice that there is no group that satisfies all the conditions above with the given tags. Usually, a person who is putting the books on the cabinet implicitly adds tags to them so that she can obtain an intended result (Figure 2).

The Bookshelf problem is defined as follows. Given a set *I* of books (or items), a set *T* of tags, a relation *R* that connects items in *I* to tags in *T* , and the number *N* of shelves, construct a set *G* of groups satisfying the following three conditions. (1) $|G| = N$. (2) Each group consists of at most $\lceil |I|/N \rceil$ items. (3) Items in a group share at least one common tag. An important characteristic of the Bookshelf problem is that there may not be such a *G* with the given *T* and *R* because they are incomplete and we have to find the missing tags and connections.

The Bookshelf problem is interesting for the following three reasons. First, it has a wide range of applications. For example, making a session program of a large academic conference is an application of the Bookshelf problem. Here, *N* is the prefixed number of sessions, and papers presented in the same session must share some common interests. Another example is to make a set of groups of participants in a travel tour. In this case, *N* is the number of groups, and it is preferable for people who act together to have some common properties.

Second, the Bookshelf problem cannot be solved by computers alone. The problem assumes that there is no solution with the given *T* and *R* since they are not complete and we need to complement the tags and connect them to items. In addition, it is often the case that we cannot compute such new tags from book contents. “My favorite” is an example of such a tag. Therefore, solving the problem requires the human power anyway.

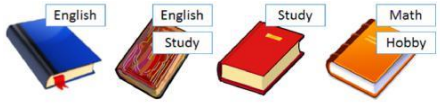

Figure 1: Example of tagged books

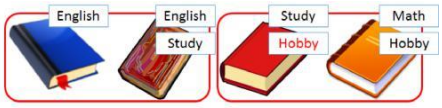

Figure 2: Creating two groups with an added tag

Third, how to solve the Bookshelf problem is not trivial. The following simple two methods are not practical.

Enumeration Method We enumerate all possible sets of groups (i.e., solution candidates) and ask humans to judge whether each set is a solution that satisfies the constraints posed by the Bookshelf problem.

Given I and N, the number of all possible sets is more than $\prod_{k=0}^{N-2}((|I|-k\cdot n)C_n)$ where $n = \lceil |I|/N \rceil$. If I is large, it is not realistic for humans to judge all solution candidates.

Two-Phase Method The method consists of two phases. In Phase 1, we ask humans to add tags and connect them to items so that they are su cient to construct a solution. In Phase 2, we use the new T and R to compute a solution for the Bookshelf problem. However, as we explain later, deciding whether T and R are su cient to construct a solution is NP-complete.

In this paper, we propose a complete, human-in-the-loop method that uses two types of microtasks to solve the Bookshelf problem. By using the two types of microtasks, we use human intelligence to (1) reduce the search space, and (2) supply new tags and connect them to items. The method is complete, in the sense that the method always gives a solution, if any, for the Bookshelf problem.

The contributions of the paper are as follows.

Microtask-based solution for a common problem. To the best of our knowledge, this is the first paper to propose a microtask-based method to solve the problem of grouping items with incomplete information at the beginning. Our method defines microtasks to be performed by humans and how to integrate the results to construct a solution. We believe that this is a significant contribution, because our method allows the people, who know the domain well but do not know how to solve the problem, to join the process to solve the problem. For example, in constructing a session program for a large academic conference, the speakers of the conference, who are expert in the domain but do not have the skill to make the program, can help construct the program. In addition, although we assume workers are reliable in this paper, the proposed method gives us an opportunity to use microtask-based crowdsourcing platform such as Amazon Mechanical Turk to solve our problem. This is an interesting issue, but we need to incorporate techniques for improving data quality in crowdsourcing (such as voting) into our algorithm and is out of the scope of this paper.

Experiment with a real set of data. We conducted an experiment using a real dataset on a large academic conference. We used the results of our microtasks to estimate the total number of tasks required to produce the solution that is the same as the actual sessions of the conference. We found that the human intelligence is e ective to avoid the worst case scenarios where the number of required microtasks is too large.

The reminder of this paper is as follows. Section 2 gives related work. Section 3 defines the Bookshelf problem. Section 4 explains our proposed method. Section 5 explains an experiment and its results. Section 6 is the conclusion.

2 Related Work

Data Classification and Clustering involving the Crowd. There are di erent researches on classifi-cation and clustering involving the crowd in some way. For example, crowdsourcing has been used for data classification. A typical approach is to give labels to data items to construct a set of training data supplied to classifiers (Vijayanarasimhan & Grauman, 2011) (Imran, Castillo, Lucas, Meier, & Vieweg, 2014) (Imran, Castillo, Lucas, Patrick, & Rogstadius, 2014). Another approach is to allow workers to evaluate the results of classifiers to improve them (Sun, Rampalli, Yang, & Doan, 2014). Although the Bookshelf problem can not be solved by such classifiers alone, we can use them to compute some of missing tags from the content

of data items. Note, however, that the tags in our context are not necessarily computed by the content of data items (such as “Favorite”). Other related researches include semi-supervised learning and label propa-gation (Chapelle, Schölkopf, Zien, et al., 2006) (Zhu & Ghahramani, 2002), recommender systems (Resnick & Varian, 1997) (Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994), community detection (Fortunato, 2009), and crowdsourced taxonomy creation (Chilton, Little, Edge, Weld, & Landay, 2013). However, none of them can compute solutions that satisfy the constraints posed by the Bookshelf problem.

Crowd solutions for making conference sessions. Making conference sessions is one of the ap-plications of our proposed method. There are several researches on human-in-the-loop approaches for making conference sessions. Cobi (Zhang et al., 2013) is a tool to allow the expected participants to collaboratively revise the shown sessions so that the program can satisfy session constraints and participants' preferences. An example of participants' preference is “Sessions A and B should not be scheduled at the same because I am interested in both sessions.” Our proposed method can be used to make the first version of the session program. Frenzy (Chilton et al., 2014) is another tool that breaks the session making into two sub-problems: metadata elicitation and global constraint satisfaction. In the second phase, a small number of volunteers who are co-located and can communicate easily with one another solve the global constraints. Our approach is unique in that we break the problem of making sessions into self-contained small microtasks and do not require workers to communicate easily to each other.

Solving Di cult Problems by Human Computation. There have been researches on human-powered approaches for solving many other problems that are di cult to solve with computers alone. Crowd-Planr (Lotosh, Milo, & Novgorodov, 2013) proposes a crowdsourcing approach to assist in solving planning problems such as a vacation trip planning, where the goal of the problem is hard to formalize. On the other hand, we deal with a problem with a clear goal, but the first input is incomplete. To the best of our knowledge, our paper is the first to address such a problem.

3 Bookshelf Problem

Definition 1 (Bookshelf problem) *Assume that there is a model M that defines* $R_M : I_M \times T_M$ *where* I_M *and* T_M *are the complete set of items and the complete set of tags in the world, respectively. Assume that we are given M, a set* $I \subseteq I_M$ *of items, a set* $T \subseteq T_M$ *of tags, a relation* $R \subseteq I \times T$, *and the number N of groups. Then, a solution of the Bookshelf problem, denoted by* $(R', G) = Bookshelf(M, I, T, R, N)$, *is defined as follows.*

- R' *is a relation s.t.* $R \subseteq R' \subseteq R_M$, *and*
- $G = \{g_1, \ldots, g_N\}$ *is a set of groups that satisfies all the following conditions.*
  1. $g_1 \oplus g_2 \oplus \ldots \oplus g_N = I$
  2. $\forall g_k \, (|g_k| \leq \lceil |I|/N \rceil)$
  3. $\forall g_k \, (\exists t_{g_k} \in \pi_{tag}(R') \, \forall i_I \in g_k \, (R'(i_I, t_{g_k})))$

The first condition states that every item belongs to exactly one group. The second condition defines the number of items in each group. The third condition states that there exists a common tag that is connected to all items in the same group. Here, $\pi_{tag}(R')$ is the set of tags that appear in R'.

The reason we need a model M in the definition is that we want to find solutions other than *meaningless solutions*. A meaningless solution is a solution having fabricated tags, such as “a book on the first shelf of this solution,” that do not exist in the original world.

Note that the Bookshelf problem always has *trivial solutions* utilizing a tag t like “item,” which satisfies $\forall i \in I \; R_M(i, t)$.

When $(R', G) = Bookshelf(M, I, T, R, N)$, we call $\pi_{tag}(R') - T$ the *hidden tags* in the solution. We also call the connections among items and tags represented by $R' - R$ the *hidden connections*. We need to add the hidden tags and connections to T and R to obtain the solution (R', G).

When $T = T_M$ and $R = R_M$ are given as the input, we say that R (thus the associated set T of tags) is *complete*. If there is at least a solution s.t. $R' = R$, R (and T) is said to be *su cient*. The Bookshelf problem does not require R to be su cient. However, deciding whether R is su cient or not is NP-complete.

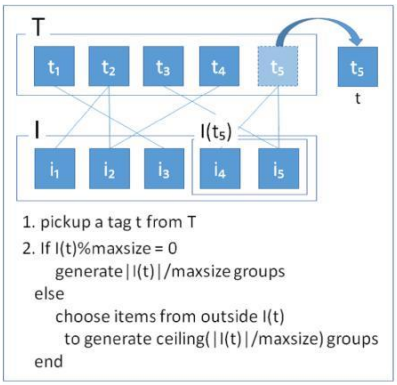

Figure 3: Overview of the proposed method

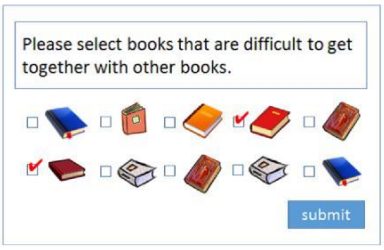

Figure 4: A GHCT

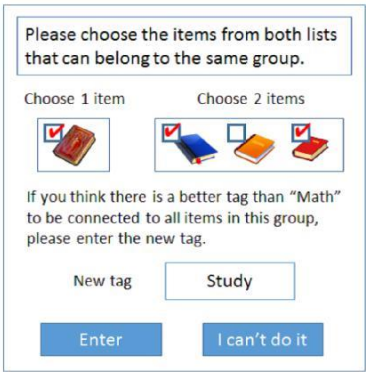

Figure 5: An LHCT

Theorem 1 *Deciding whether R is su cient or not is NP-complete.*

Proof outline. We show the NP-hardness of the Bookshelf problem by reducing the Exact Cover by 3-Sets problem to the problem of determining whether *R* is su cient or not. *2*

4 Proposed Method

This section explains a microtask-based, human-in-the-loop method to solve the Bookshelf problem. The features of the method are as follows. First, because microtasks are automatically generated during the process, humans can focus on the localized problems posed by microtasks. Second, our microtasks try to utilize the human intelligence to reduce the search space, by asking the humans what items are likely to be di cult to get together with other items. As we will show later, the human intelligence is proved to be e ective to avoid the worst case in searching for the solution. Third, the method is guaranteed to give a solution, if any, for the Bookshelf problem.

This section first explains the overview of the method. Then, it explains two types of microtasks, namely, *Global Human Computation Task (GHCT)* and *Local Human Computation Task (LHCT)*, used in the proposed method. Finally, it explains the method in detail.

4.1 Overview

In short, the method works as follows. First, the method initializes the set *G* of groups to an empty set. Then, it constructs item groups one by one and adds them to *G* by repeatedly applying the following two steps (Figure 3).

1. It picks up a tag *t* from *T* .
2. Let $I(t)$ be a set of items that are connected to *t* and have not belonged to any group yet. Let $maxsize = \lceil |I|/N \rceil$. Then, it performs one of the followings according to $I(t)$.
   - If $|I(t)|\%maxsize = 0$, divide the items in $I(t)$ into $|I(t)|/maxsize$ groups.
   - If $|I(t)|\%maxsize \neq 0$, choose $maxsize - |I(t)|\%maxsize$ items from items outside $I(t)$ that have not belonged to any group yet. If they can be merged to $I(t)$ by tagging them with *t*, divide the items in $I(t)$ plus the chosen items into $\lceil I(t)/maxsize \rceil$ groups of items. If not, do nothing.

During the repetition, if the algorithm finds an item that cannot belong to any group, it backtracks. When all items succeed in belonging to groups, it outputs *G* as a solution.

Pickup and Choose operations. Note that there are two key operations that are underlined. First, picking up a tag from *T* a ects the number of necessary backtracks and thus the total computation cost. We call the operation *Pickup* here. A straightforward implementation of the Pickup operation is to randomly pick up a tag. Obviously, it is ine cient. Second, choosing $maxsize - |I(t)|\%maxsize$ items from

the other items than $I(t)$ cannot be performed by computers because they have no common tags with those in $I(t)$. We call the operation *Choose*.

Therefore, we want to use human intelligence for the two operations, expecting that humans are good at performing them. First, humans can help perform the *Choose* operation by directly choosing items that can be merged to items in $I(t)$.

Second, how to help perform *Pickup* operation is more interesting, because the operation determines the global direction in the search space and directly a ects the number of backtracks. For e ciency, we want to make the number of backtracks as small as possible. A widely-used heuristic to cope with this kind of problem is to solve the most di cult problem first. In our context, it is desirable to make groups for those items that are di cult to get together with other items in the early stage. If such items remained in the later stage, it would be extremely di cult to find items to get together with them. Therefore, humans can help perform the operation by picking up items that seem di cult to get together with other items.

We designed two types of microtasks to help the two operations, named *Global Human Computation Tasks (GHCTs)* and *Local Human Computation Tasks (LHCTs)*, which help us perform the Pickup and Choose operations. The results of GHCTs are used to compute values representing the priority of tags in the Pickup operation. LHCTs directly ask humans to perform the Choose operation.

4.2 Microtasks

Global Human Computation Tasks (GHCTs). A GHCT is a task to compute a numerical value between 0 and 1, named *D-value*, that represents each item's di culty to get together with other items. We use the D-values in the Pickup operation as follows. Let $I(t)$ be a set of items that are connected to t and have not belonged to any group yet. Given D-values, we can implement the Pickup operation so that it picks up t s.t. $I(t)$ includes the item whose D-value is larger than any other ungrouped item. This way, we try to make groups for items that are likely to be di cult to find appropriate groups, in an early stage.

Figure 4 is an example of a GHCT. Each GHCT shows a fixed number of items to workers and asks them to select items that are di cult to get together with the other items.

We generate GHCTs as follows. Let L be a fixed parameter that represents the number of shown items in each GHCT. Similarly, let d be a fixed parameter (say, $d = 3$) that represents the minimum number of occurrences of each item in GHCTs. Given the set I of items, we generate $\lfloor |I|/L \rfloor \times (d + 1)$ GHCTs where each item in I appears in GHCTs at least d times. For example, assume that the number of items shown in each GHCT is ten (i.e., $L = 10$), the number of items is 11 ($|I| = 11$), and the minimum number of occurrences of each item is three ($d = 3$). Then, we create 4 GHCTs in total where all items appear at least three times in GHCTs and some of them (in this case, $L \times 4$ *modulo* $(|I| \times d) = 7$ items) appear four times.

We ask workers to perform all GHCTs at the beginning of the proposed method. Given an item i and the results of all GHCTs, let d_i be the number of workers that labeled i as "di cult to get together with other items". Let o_i be the total number of occurrences of i in the GHCTs. Then, the D-value of the item i is computed by d_i/o_i.

Local Human Computation Tasks (LHCTs). LHCTs ask workers to find hidden tags and connections between items and tags. An LHCT shows workers the list of items in $I(t)$ and another list of items, denoted by $\bar{I}(t)$, which contains all items that are *not* connected to t and have not been decided to belong to any item group. Then, it asks them to choose (1) $|I(t)|\%maxsize$ items (denoted by S_1) from $I(t)$ and (2) $maxsize - |S_1|$ items (denoted by S_2) from $\bar{I}(t)$ so that we can generate a group for $S_1 \cup S_2$. Figure 5 is an example of an LHCT. Workers either choose the items or click on "I can't do it" button. Then, we can make an item group (of *maxsize* size) consisting of all items in S_1 and S_2.

In the proposed method, an LHCT is generated every time the Choose operation is executed for a tag $t \in T$ when $|I(t)|\%maxsize \neq 0$ (Figure 3). The idea behind this is that we can make $\lfloor |I(t)|/maxsize \rfloor$ item groups for $I(t)$ but we have to find items that can get together with the remaining items.

If a worker thinks that there is a better tag t' than t for the group of items containing both S_1 and S_2, she can enter the tag in the task window. Then, the new tag is connected to items in S_1 and S_2. Otherwise, the algorithm connects t to items in S_2.

Given the results of LHCTs, we can create $\lceil |I(t)|/maxsize \rceil$ item groups in total. Here, $\lfloor |I(t)|/maxsize \rfloor$ groups have t as their common tag and one group has t or t' as its common tag.

Algorithm 1 SearchWithGLHCTs

Input: N, I, T, R
Output: $G \neq \phi$
1: $maxsize \leftarrow \lceil |I|/N \rceil$
2: $G \leftarrow \phi$
3: let $T_{\text{available}} = \{t | t \in T, I(t) \neq \phi\}$
4: **while** $T_{\text{available}} \neq \phi$ **do**
5: pickup $t \in T_{\text{available}}$
6: **if** $|I(t)|\%maxsize == 0$ **then**
7: generate $|I(t)|/maxsize$ groups and add them to G
8: **else**
9: choose $S_1 (\in I(t))$ and $S_2 (\in I(t))$ s.t. $|S_1 \cup S_2| = maxsize$ and items in $S_1 \cup S_2$ has a common tag t'
10: **if** $S_1 \cup S_2 = \phi$ **then**
11: $T_{\text{available}} \leftarrow T_{\text{available}} - t$
12: **else**
13: restart with $G = \phi$, $T = T \cup \{t'\}$ and $R = R \cup \{(t', i) | i \in S_1 \cup S_2\}$
14: **end if**
15: **end if**
16: **end while**
17: $G \leftarrow G \cup Groups4Leftovers(I, T, R, G)$
18: **if** $|G| == N$ **then**
19: return G
20: **end if**
21: restart with $G = backtrack(I, T, R, G)$ and restore $T_{\text{available}}$

Note that each LHCT used for performing the Choose operation guarantees that the obtained item group satisfies the constraint that all items in it share at least one common tag.

4.3 Algorithm and Completeness

SearchWithGLHCTs (Algorithm 1) shows the algorithm of our proposed method. The algorithm assumes that we have already submitted GHCTs we explained in Section 4.2 and obtained D-values for the items in I. On the other hand, LHCTs are generated and issued during the execution.

Lines 1 to 3 define three variables used in the algorithm. In Line 1, *maxsize* is the max number of items in each group defined by the Bookshelf problem. In Line 2, G is the list of generated groups. G is initially an empty list but we incrementally add new groups to its tail. We call each intermediate state of G, denoted by $[g_1, \ldots]$, a *state* of G. In Line 3, $T_{\text{available}}$ is a variable that always keeps a set of tags s.t for every tag t in $T_{\text{available}}$ there are items connected to t that do not belong to any item group yet. Note that the number of tags in $T_{\text{available}}$ decreases when we make items groups.

Lines 4 to 21 implement the logic explained in Section 4.1. The loop (lines 4 to 16) applies the steps illustrated in Figure 3. The loop ends when there is no tag in $T_{\text{available}}$.

In each iteration, the algorithm performs the followings. In Line 5, it picks up a tag t from $T_{\text{available}}$ according to D-values s.t. $I(t)$ contains the item whose D-value is larger than any other ungrouped item. In Lines 6 to 7, if $|I(t)|\%maxsize = 0$, it divides items in $I(t)$ into $|I(t)|/maxsize$ groups and adds the groups to G.

When adding groups to G, we check whether the result is not included in the list of *prohibited states*, to avoid repeating the same process. If the list includes the new state generated by adding the new groups to the current state, the algorithm does not add them to G, and removes t from $T_{\text{available}}$. How to maintain the list of prohibited states is explained later when we explain how the algorithm backtracks.

If $|I(t)|\%maxsize \neq 0$ (Line 8), we issue LHCTs to execute the Choose operation (Line 9). If workers cannot find an appropriate set of items, the algorithm concludes that t is not useful for making groups and removes it from $T_{\text{available}}$. Removed tags are restored when it backtracks because they may be useful after the backtrack. If workers find items to get together, it concludes that t is useful (Lines 12 to 13). Different from what we explained with Figure 3, the algorithm does not generate item groups immediately in this case. Instead, it restarts the search process with the obtained tag t' included in T (Line 13). Note that workers have the option of not entering a new tag (Figure 5) to argue that there is no better tag than t in an LHCT and then $t' = t$. The reason for restarting the process is that we want to do the best to obey the principle of making item groups for items with the highest D-value first.

When the algorithm generates all item groups for $T_{\text{available}}$, it quits the loop. Then, it generates item groups for those items that have not belonged to any group yet. This happens when tags having un-

grouped items are removed from $T_{\text{available}}$ (Line 11). *Groups4Lef tovers*(*I, T, R, G*) generates such groups in a best e ort fashion. The function first computes $I_{leftover}$, which is a set of ungrouped items at the time the algorithm quits the while loop. Then, it makes groups as follows: For every t s.t. $|I(t)|/maxsize = J$ and $|I(t)|\%maxsize = K$ with items in $I_{leftover}$, generate J groups containing *maxsize* items and one group containing K items.

Finally, if the total number of groups is N, it returns the set of group. Otherwise, the algorithm backtracks and restarts the search process. When backtracking, we add G to the list of prohibited states to avoid reaching the state again[1].

The destination state of the backtrack is computed as follows. Let i' be an item whose D-value is the largest of the items in $I_{leftover}$ that share a common tag with an item i in G. Then, let $G = [g_1, \ldots, g_j, \ldots, g_k]$ be the state when it quit the loop where g_j is the last group in the sequence that contains such an i'. The algorithm backtracks to the state $[g_1, \ldots, g_{j-1}]$. If we cannot find such an i', it backtracks to the state $[g_1, \ldots, g_{k-1}]$ (i.e., $j = k$). We temporarily add any state containing g_j, such as $[g_1, \ldots, g_j]$, to the list of prohibited states to avoid grouping i with other items than i'. If we need to backtrack later again, we remove the temporarily prohibited states from the list because prohibiting g_j did not lead to a solution.

Completeness of the Algorithm. In order to guarantee that the algorithm generates a solution, we add nameless tags $\{\mathbf{NT}_i | i \in I\}$ to T and connect each $\mathbf{NT}_i$ to i before executing the algorithm. Then, the following theorem holds.

Theorem 2 *SearchWithGLHCTs gives a solution, if any, for the Bookshelf problem when* $|I|$ *modulo* $N = 0$.

Proof outline. We show the following two things. First, SearchWithGLHCTs does not generate any state after it knew that the state is not a subsequence of any solution. Second, let a solution be $[g_1, g_2, \ldots, g_N]$, each of which contains $|I|/N$ items. Then, given a subsequence of the solution $[g_1, \ldots, g_i]$, the algorithm guarantees that g_{i+1} (of the solution) can be added to the subsequence by LHCTs. *2*

In addition, we can extend SearchWithGLHCTs by (1) adding *wildcard items* to I when $|I|$ *modulo* $N \neq 0$ so that $|I|$ *modulo* N becomes zero, and (2) prohibiting any state containing a group of wildcard items only. Wildcard items are those that can get together with any items. Then, the extended version, named SearchWithGLHCTs2, generates a solution for the Bookshelf problem.

Theorem 3 *SearchWithGLHCTs2 gives a solution, if any, for the Bookshelf problem.* *2*

Note that the two theorems do not guarantee that the algorithms always give the same solution when they run multiple times.

5 Experiment

We conducted an experiment to examine the e ect of human computation on reducing the search space in the Bookshelf problem. For that purpose, we investigated how D-values computed from the results of GHCTs a ected the number of LHCTs, assuming that workers perform LHCTs without mistakes. We compared the four settings to compute D-values in the number of LHCTs and backtracks required to give a solution.

5.1 Settings

5.1.1 Data

We applied the proposed method to make a session program for DEIM2014, a large domestic academic conference in Japan. We obtained data from the DEIM2014 web site and constructed the input for the algorithm as follows.

- I: All papers taken from each DEIM2014 session with six presentations. $|I| = 192$.
- T : All keywords associated to the papers in I and the nameless tags $\{\mathbf{NT}_i | i \in I\}$. $|T| = 566 + 192$.
- R: Connections from keywords (including the nameless tags) in T to the papers in I. $|R| = 637 + 192$.

[1] Note that this is not $G \cup G'$, since G' is computed based on G.

• N: 32 (equals to the number of the DEIM2014 sessions with six presentations).

Under di erent settings on the way to compute D-values we explain next, we counted the number of LHCTs and backtracks required to find a solution G that equals to the original sessions of DEIM2014.

5.1.2 Four Settings to Compute D-values

The four settings to compute D-values are as follows.

Setting 1 [Const]: We do not use the results of GHCTs. Instead, we use a constant D-value C for all items.

Setting 2 [D_R]: We do not use the results of GHCTs. Instead, we use R to compute the D-values of each i, denoted by $D_R(i)$. $D_R(i)$ is defined as the multiplicative inverse of (one plus) the number of other items that share at least one tag with i. The idea behind this is that an item having many other items that share at least one tag with the item is likely to be easy to get together with other items. $D_R(i)$ is recalculated every time new connections are add to R in the process of the proposed algorithm.

Setting 3 [D_{ghct}]: We used the results of GHCTs to compute D-values (Section 4.2). We call such a D-value a $D_{ghct}(i)$.

Setting 4 [D_R&D_{ghct}]: We use a pair ($D_R(i)$, $D_{ghct}(i)$) as the D-value of each i . We define the order relation among them by the dictionary order of the components.

5.1.3 Pickup Operation

As explained, we use D-values to pick up tags in the Pickup operation. In the experiment, we also implemented some heuristics in the operation to e ciently find appropriate solutions. Note that the order of picking up tags does not a ect Theorems 2 and 3.

The implemented Pickup operation uses the following rules to pick up tags. The rule with the smaller number has a higher priority. If a rule picks up more than one tag, we use the next priority rule to break the tie. Similarly, if a rule cannot pick up any tag, we use the next rule. If no rule can pick up a tag, a tag is randomly selected.

1. If there is t s.t. $|I(t)|\%maxsize = 0$, use t, because it would not require any LHCTs to make groups.
2. Let $D_{Max}(t)$ be the highest D-value of items in $I(t)$. Then, use t having the largest $D_{Max}(t)$, because it allows us to make a group for the most di cult item.
3. Let $D_{Avg}(t)$ be the average of D-values of items in $I(t)$. Then, use t having the largest $D_{Avg}(t)$, because it allows us to make groups for di cult items.
4. Pick up t s.t. $|I(t)|\%maxsize$ is the largest. This increases the number of items in S_1 (guaranteed to share a common tag) in each LHCT and reduces the number of items the worker has to choose for S_2 to make a group.
5. Pick up t s.t. $|I(t)|$ is the smallest. This allows us to make groups with a variety of tags because we avoid using tags that are common to many items.

5.1.4 Performing microtasks

We generated GHCTs with $L = 10$ and $d = 3$ (See Section 4.2). Since $|I| = 192$, we obtained $\lfloor |I|/L \rfloor \times (d + 1) = 76$ GHCTs in total. In each GHCT, the title, keywords, and the first author's name and a liation of each paper was presented to workers.

We submitted the GHCTs to the Crowd4U crowdsourcing platform (Morishima, Shinagawa, Mitsu-ishi, Aoki, & Fukusumi, 2012). Workers for the tasks were chosen from those who belong to laboratories of University of Tsukuba whose members attended DEIM2014. This makes sense because a typical setting for this application is that we ask paper authors of the conference to be workers. The average and median of times required to perform a GHCT are 70.0 seconds and 41 seconds, respectively.

D_{ghct}	#papers
0	105
0.25	45
0.33	2
0.5	27
0.75	11
1	2

Table 1: Distribution of D_{ghcts}

	#LHCTs			#backtracks		
Setting	Max	Min	Average	Max	Min	Average
Setting 1 [Const]	1997	107	490.8	667	0	33.9
Setting 2 [D_R]	2305	95	453.8	116	2	28.3
Setting 3 [D_{ghct}]	1234	80	515.8	186	1	38.9
Setting 4 [D_R&D_{ghct}]	324	152	211.7	10	4	6.5

Table 2: The number of LHCTs and backtracks with the four settings

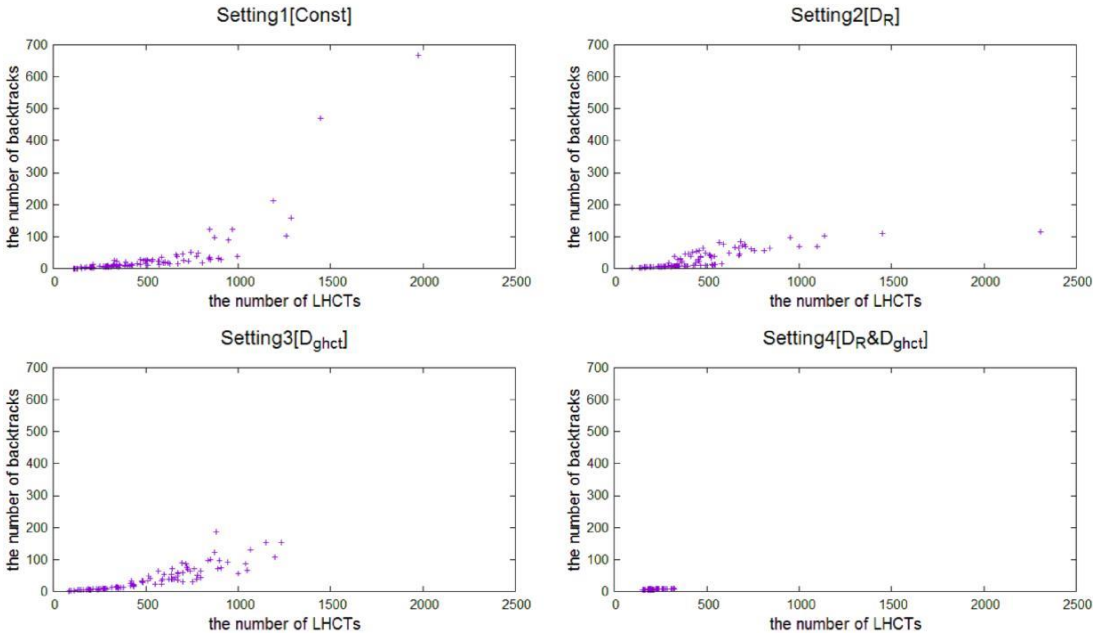

Figure 6: Distribution of the numbers of LHCTs and backtracks.

As the purpose of this experiment is to examine the e ects of human computation on the search space, we assumed that all LHCTs are performed without mistakes[2]. Therefore, we implemented an agent that works as a worker who does not make any mistakes. Given an LHCT with a set of papers, each worker chooses papers only if they are in a correct DEIM2014 session, and she clicks on “I can’t do it” otherwise. She always provides a correct tag to LHCTs.

Since the algorithm uses random numbers in the Pickup operation, we executed it 100 times in each setting.

5.2 Results

Table 1 shows the statistics on the obtained D_{ghct}s. From the definition, the largest number of D_{ghct} is 1 and the table shows that workers thought that two papers were the most di cult to get together with other papers. On the other hand, the D_{ghct}s of more than 50% papers are 0, which means that they thought that many papers were easy to be grouped with other papers.

Table 2 summarizes the numbers of LHCTs and backtracks required to obtain the correct session program in the executions in each of the four settings. As the table shows, we found that the Setting 4 with (D_R, D_{ghct})s generated the best result. In particular, the maximum number and average number is the smallest among those of the four settings. Even if we add the number of GHCTs (76), the maximum number and average number is the smallest.

Figures 6 shows the distribution of the numbers of LHCTs and backtracks in detail. Compared to

[2] Note that even if workers make mistakes, Theorem 2 guarantees that the algorithm eventually generates a solution as long as we can determine whether the item groups in the output is not a correct solution.

Setting 1, Setting 2 and Setting 3 reduced the average and the maximum numbers of LHCTs, respectively. Therefore, the results suggested that the attached keywords and the results of human computation were e ective to reduce the search space in di erent ways. Setting 4 gave the best results in both of the numbers of backtracks and LHCTs. The average number of LHCTs for Setting 4 was 2.3x smaller than that for Setting 1, and more importantly, the variance was much smaller: The maximum number of LHCTs for Setting 4 was 6.2x smaller than that for Setting 1.

Given those results, we conclude that human computation is e ective to avoid the worst case in searching for solutions of the Bookshelf problem, especially when the results of human computation are combined with those of machine computation.

6 Conclusion

This paper defined the Bookshelf problem to model the problem of grouping data items with incomplete information at the beginning, and proposed a complete, human-in-the-loop method that uses two types of microtasks for solving the problem. In addition, we conducted an experiment with a set of real data taken from a large academic conference, to investigate the e ects of human computation on the e ciency in searching for solutions. We found that human computation is e ective in avoiding the worst case and reducing the expected number of microtasks, especially when the results of human computation are combined with the results of machine computation. Future work includes the development of a parallel version of our algorithm to allow many workers to process LHCTs in parallel.

References

Chapelle, O., Schölkopf, B., Zien, A., et al. (2006). Semi-supervised learning.

Chilton, L. B., Kim, J., André, P., Cordeiro, F., Landay, J. A., Weld, D. S., . . . Zhang, H. (2014). Frenzy: collaborative data organization for creating conference sessions. In *CHI conference on human factors in computing systems, chi'14, toronto, on, canada - april 26 - may 01, 2014* (pp. 1255–1264).

Chilton, L. B., Little, G., Edge, D., Weld, D. S., & Landay, J. A. (2013). Cascade: crowdsourcing taxonomy creation. In *2013 ACM SIGCHI conference on human factors in computing systems, CHI '13, paris, france, april 27 - may 2, 2013* (pp. 1999–2008).

Fortunato, S. (2009). Community detection in graphs. *CoRR*, *abs/0906.0612* . Retrieved from http://arxiv.org/abs/0906.0612

Imran, M., Castillo, C., Lucas, J., Meier, P., & Vieweg, S. (2014). AIDR: artificial intelligence for disaster response. In *23rd international world wide web conference, WWW '14, seoul, republic of korea, april 7-11, 2014, companion volume* (pp. 159–162).

Imran, M., Castillo, C., Lucas, J., Patrick, M., & Rogstadius, J. (2014). Coordinating human and machine intelligence to classify microblog communications in crises. *Proc. of ISCRAM* .

Lotosh, I., Milo, T., & Novgorodov, S. (2013). Crowdplanr: Planning made easy with crowd. In *29th IEEE international conference on data engineering, ICDE 2013, brisbane, australia, april 8-12, 2013* (pp. 1344–1347).

Morishima, A., Shinagawa, N., Mitsuishi, T., Aoki, H., & Fukusumi, S. (2012). Cylog/crowd4u: A declarative platform for complex data-centric crowdsourcing. *PVLDB*, *5* (12), 1918–1921.

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: An open architecture for collaborative filtering of netnews. In *CSCW '94, proceedings of the conference on computer supported cooperative work, chapel hill, nc, usa, october 22-26, 1994* (pp. 175–186).

Resnick, P., & Varian, H. R. (1997). Recommender systems - introduction to the special section. *Commun. ACM* , *40* (3), 56–58.

Sun, C., Rampalli, N., Yang, F., & Doan, A. (2014). Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *PVLDB*, *7* (13), 1529–1540.

Vijayanarasimhan, S., & Grauman, K. (2011). Large-scale live active learning: Training object detectors with crawled data and crowds. In *The 24th IEEE conference on computer vision and pattern recognition, CVPR 2011, colorado springs, co, usa, 20-25 june 2011* (pp. 1449–1456).

Zhang, H., André, P., Chilton, L. B., Kim, J., Dow, S. P., Miller, R. C., . . . Beaudouin-Lafon, M. (2013). *Cobi*: communitysourcing large-scale conference scheduling. In *2013 ACM SIGCHI conference on human factors in computing systems, CHI '13, paris, france, april 27 - may 2, 2013, extended abstracts* (pp. 3011–3014).

Zhu, X., & Ghahramani, Z. (2002). *Learning from labeled and unlabeled data with label propagation* (Tech. Rep.). CMU-CALD-02-107, Carnegie Mellon University.