Applied Computing and Informatics (2017) 13, 47–56

Saudi Computer Society, King Saud University

Applied Computing and Informatics

(http://computer.org.sa)
www.ksu.edu.sa
www.sciencedirect.com

ORIGINAL ARTICLE

Real-time recommendation algorithms for crowdsourcing systems

Mejdl Safran [a,b,*], Dunren Che [a]

[a] Computer Science Department, Southern Illinois University, Carbondale, IL, USA
[b] Computer Science Department, King Saud University, Riyadh, Saudi Arabia

Received 3 October 2015; revised 31 December 2015; accepted 2 January 2016
Available online 23 January 2016

KEYWORDS

Crowdsourcing;
Recommendation algorithms;
Task recommendation;
Worker recommendation;
Top-*k* tasks;
Top-*k* workers

Abstract Crowdsourcing has become a promising paradigm for solving tasks that are beyond the capabilities of machines alone via *outsourcing* tasks to online crowds of people. Both requesters and workers in crowdsourcing systems confront a flood of data coming along with the vast amount of tasks. Fast, *on-the-fly* recommendation of tasks to workers and workers to requesters is becoming critical for crowdsourcing systems. Traditional recommendation algorithms such as collaborative filtering no longer work satisfactorily because of the unprecedented data flow and the on-the-fly nature of the tasks in crowdsourcing systems. A pressing need for real-time recommendations has emerged in crowdsourcing systems: on the one hand, workers want effective recommendation of the top-*k* most suitable tasks with regard to their skills and preferences, and on the other hand, requesters want reliable recommendation of the top-*k* best workers for their tasks in terms of workers' qualifications and accountability. In this article, we propose two real-time recommendation algorithms for crowdsourcing systems: (1) TOP-K-T that computes the top-*k* most suitable tasks for a given worker and (2) TOP-K-W that computes the top-*k* best workers to a requester with regard to a given task. Experimental study has shown the efficacy of both algorithms.

1. Introduction

It has been a confirmed phenomenon that almost every area of human activities, especially those related to scientific exploration and technological applications is now producing and consuming large scales of data. Google estimated that every two days in 2010 the world generated as much data as the sum it generated up to 2003 [1]. This phenomenon has brought a great challenge to the technological society and to the vast users who often do not find the desired information within an acceptable time frame. In crowdsourcing systems, users

* Corresponding author at: Computer Science Department, Southern Illinois University, Carbondale, IL, USA.
E-mail addresses: mejdl.safran@siu.edu (M. Safran), dch@cs.siu.edu (D. Che).
Peer review under responsibility of King Saud University.

Production and hosting by Elsevier

http://dx.doi.org/10.1016/j.aci.2016.01.001

(both requesters and workers) confront exactly the same problem to which a feasible solution is thorough *real-time*[1] recommendation. For many applications, recommendation is a personalized filter, used to either predict the interestingness of an item (a *prediction problem*), or to identify a set of items that are interesting to a user (a *top-k recommendation problem*) [2,3]. Effective recommendation plays a critical role in such web-based systems and benefits users in multiple ways, discovering relevant information, reducing waiting time, and increasing productivity, to name a few. Many of the leading e-commerce platforms such as Amazon and Netflix have already adopted recommendation systems in their systems and demonstrated the great value of effective recommendation.

Current recommendation systems are based on two main approaches, i.e., content filtering and collaborative filtering. As their names indicate, the content filtering approach relies on the content of the items and the users' profiles to identify the best matches. On the other hand, the collaborative filtering approach relies on the relationships among the items and the correlations among the users to draw new hidden, interesting relationships between the items and the users. These approaches have been showing their successes in many applications, mostly online e-commerce systems.

However, when it comes to recommending tasks on crowdsourcing platforms such as AMT (Amazon Mechanical Turk), these traditional recommendation approaches do not fit well. In other words, they would not be sufficiently efficient anymore when they are used for task recommendations in crowdsourcing systems. Before making detailed explanation, let's review the basics of crowdsourcing. In recent years, crowdsourcing has become a popular paradigm for accomplishing tasks by outsourcing them to online crowds of people. Crowdsourcing systems (usually built as online platforms) originated from the idea of leveraging human abilities on solving problems and accomplishing tasks that machines alone cannot do well. Crowdsourcing has many different definitions [4], each emphasizing different aspects: the nature of the collaboration, the types of the target problems, the motivations of the crowd members, and the types of incentives [5]. The definition in [6] is concise and popular: "crowdsourcing is an online, distributed problem-solving and production model". Crowdsourcing systems may be categorized into four types [7] based on whether the workers' contributions are convergent or not and whether they are homogeneous or heterogeneous. In the rest of this article, our discussion is not limited to any particular type of crowdsourcing systems. Even though we use AMT (a crowd processing system) for illustration, the algorithms we present in this article are suitable for any type of crowdsourcing systems as long as the systems provide task categories or the tasks can be categorized.

In AMT, requesters post tasks with specifications and requirements; workers select tasks to work on according to their interest and skills, and then get paid once the requesters accept their completed results. In terms of performance, the number of tasks posted in AMT within a short period of time can be huge. As a result, the number of available tasks as candidates to be recommended to a worker is also huge. Additionally, tasks in crowdsourcing systems like AMT are short ones (taking a few seconds to a few minutes), so the life cycle of a task in crowdsourcing systems is very short. Consequently, effective recommendation of workers and tasks in crowdsourcing markets such as AMT is not easily fulfilled because of the huge pools of tasks and workers and the limit of short life span of the tasks. In terms of quality of completed tasks, current crowdsourcing systems that include AMT have found numerous drawbacks. Since AMT is based on the first-come-first-serve basis, a less qualified worker may start working on a task while a better-skilled worker may not find the right task as it may be listed at a later page or has been already taken by a less-skilled worker. As a result, less qualified workers may get to work on a task and result in low-quality completion (sometimes still accepted). If unsatisfied, the requester may have to repost the same task many times to get a quality result. Repeated receiving of low quality results may discourage the requesters from using the crowdsourcing system in the future.

Consequently, the need for efficient (real-time) recommendation of both tasks and workers in crowdsourcing systems is becoming a rather pressing issue. Workers, on the one side, desire the top-k best-fitting tasks being promptly and effectively recommended to them, and requesters, on the other hand, want the top-k best workers recommended for their tasks. This phenomenon had motivated our investigation and has resulted in some interesting outcomes. The article reports our work and brings in the following main contributions:

(i) We inspect the characteristics of the tasks in crowdsourcing systems as compared to the items/products in e-commerce markets, and the differences of the workers' interests in tasks vs. the users' interests in items/products. The products in e-commerce markets are usually offered as *completed* items to be purchased by online users, whereas the tasks in crowdsourcing systems are posted activities that are *yet to be completed* by online workers. We incorporate a key mechanism, i.e., categories of tasks, into our recommendation approach that significantly accelerates the recommendation procedure.

(ii) We propose the TOP-K-T algorithm to help workers in crowdsourcing systems to instantly identify the top-k most suitable tasks for them in a "pulling" manner. Therefore the workers can spend more time directly on completing the tasks [8] and maximize their productivity and awards.

(iii) We also propose the TOP-K-W algorithm to help requesters in crowdsourcing systems to quickly find the top-k most *qualified* workers in a "pushing" manner. To the best of our knowledge, there is no such algorithm proposed that supports requesters by pushing their tasks to the right workers.

(iv) We conduct extensive experiments on synthesized large-scale datasets generated based on the scenarios of real-world applications of crowdsourcing systems. Our experimental results show that our algorithms (TOP-K-T and TOP-K-W) can make valid recommendations in real-time performance (in milliseconds or less) at all the set scales of the test data.

The remainder of this article is organized as follows. In Section 2, we briefly review the current recommendation

[1] In this article, we use the term "real-time" in a less strict sense just to emphasize the great *promptness* or *efficiency* required for the recommendations in real crowdsourcing systems, given the fact that many short tasks take minutes or even seconds and may have already been completed by others before being recommended to a worker.

approaches. In Section 3, we present our new recommendation algorithms: TOP-K-T and TOP-K-W. In Section 4, we show and discuss our experimental results. In Section 5, we comment on related works and make comparisons with ours. Section 6 concludes the article.

2. Current recommendation approaches

In this section we briefly review current recommendations methods centered around the two main recommendation approaches: (1) the content filtering approach and (2) the collaborative filtering approach.

The *content filtering* approach recommends items to a user based upon the description of the items and the profile of the user's interests [9]. A profile is created for each user and for each product/item. The user profile can be simply a collection of the user's historical ratings on purchased items. The product profile is a set of keywords representing the product. Similarities between user profiles and product profiles are computed. Products with high similarities will be recommended to the corresponding users. Obviously, the approach does not have cold start problem. The problems this approach may face include the following: (1) some items may not be easily described using content keywords; (2) distinct items may share the same set of features described by the same keywords; (3) profile information is not always available; and (4) poor performance scalability.

The *collaborative filtering* approach is probably the most successful and popular approach used in recommendation systems [3]. Comparing to content filtering, the collaborative filtering approach relies only on the user's past behavior. This approach identifies hidden user-products relationships by analyzing the relationships among users and the correlations among products [10,9,3]. One remarkable advantage of this general approach is that it usually generates pretty accurate results because the learned user-product relationships implicitly incorporate many subtle aspects that are hard to be explicitly profiled. The approach notoriously suffers from the cold start problem because it relies on collected historical information that new users/products do not yet have. Another equally remarkable drawback of this approach at the current status is its high runtime complexity, which is typically polynomial w.r.t. both the number of users and the number of items, which both can be huge in future crowdsourcing systems. Therefore, in principle this attractive approach will not result in real-time recommendation which is very much needed by future, very-large-scale crowdsourcing systems. The approach has evolved into two different methods as follows.

The *user-based collaborative filtering* method recommends to a user the products that have already been liked by like-minded peers of the user. It consists of two steps [3]. First, a user's historical information is used to identify a neighborhood of people who in the past have exhibited similar behavior, e.g., purchased similar products. Second, the identified neighborhood is analyzed to figure out new products that may be liked by the user. We furnish additional details of the user-based collaborative filtering method in Supplementary Table 1.

The *item-based collaborative filtering* method recommends items similar to the items that a user already liked [2,10]. This approach consists of two phases. The first phase is called the model phase that computes the similarity between every pair of items, and may be executed offline. The second phase combines and compares the computed similarity scores to determine the most similar items to the items that have been purchased in the past by the user. We provide more details of the item-based collaborative filtering method in Supplementary Table 1. To build the model, the algorithm takes $O(m^2n)$ time [10], where m is the number of items and n is the number of operations needed to compute the similarity between every two items. The second phase takes $O(k|P|)$ time [10] to decide the k most similar items for each item in P (i.e., the set of items purchased by a given user). We will see later in this article why this polynomial algorithm does not result in a satisfactory, real-time solution for crowdsourcing systems.

3. Proposed recommendation algorithms

In this section, we first motivate our overall strategy and present the key supportive data structures. Based on that we then present our two recommendation algorithms, TOP-K-T and TOP-K-W, in turn.

3.1. Motivation, data structures, and matching scores

We observe several aspects of essential difference between traditional e-commerce systems and crowdsourcing systems that ought to be carefully considered in the design of the recommendation algorithms for crowdsourcing systems.

(1) The items in e-commerce markets are *completed* products to be purchased by online users or customers, whereas the tasks in crowdsourcing systems are posted activities *yet to be completed* (or solved) by online workers. The tasks in crowdsourcing systems may be completed at *varied levels of quality*, and thus could be rejected by the requesters/owners of the tasks if the completion does not fulfill the required quality standard.
(2) The items in e-commerce systems typically have many copies and can be sold by different sellers, whereas the tasks in crowdsourcing systems are *unique* (except for the occasional case that a requester purposefully posts the same task multiple times in order to gather majority opinions or alternative solutions). Furthermore, the tasks in a crowdsourcing system usually have pretty *short life span*, of which many can be completed in minutes or seconds by experienced workers.
(3) A user's interest in e-commerce markets is unrestricted since one can buy basically anything in the market that he/she likes as long as he/she has the money, whereas a worker's choice in a crowdsourcing system is reasonably limited by his/her expertise/skills and personal interests. Generally, a worker only picks up tasks that fit his/her skills and interest, typically are a much restricted subset of a potentially huge set of all tasks available in a crowdsourcing system.

Of the above observation, items 1 and 2 indicate the high complexity, high diversity, and high velocity inherently existent in crowdsourcing systems, which further imply that straightforward utilization of the recommendation methods developed for traditional e-commerce systems would not result in satisfactory performance. A novel recommendation

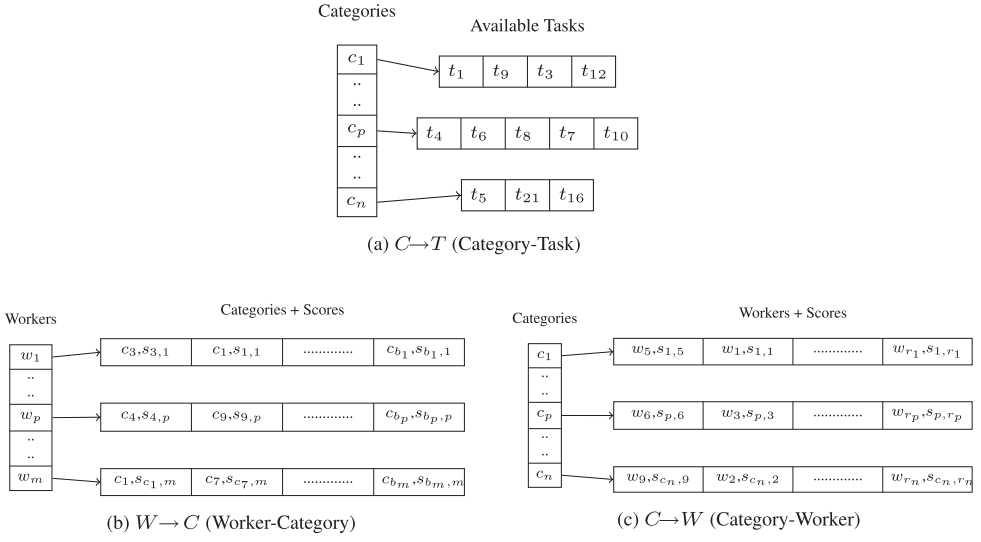
approach is much needed in order to deliver the required high (real-time) performance of recommendations for both tasks and workers in crowdsourcing systems. Fortunately, item 3 brings up a twilight of hope toward overcoming the challenge of recommendations of tasks and workers at real-time speed in crowdsourcing systems. The mapping between the tasks and the workers is much restricted compared to the unlimited mapping between products and buyers in an e-commerce system. This difference is *essential*, analogous to the difference between a highly selective θ-join and an unlimited cross-product operation in a relational database system. This observation enlightened us to introduce an efficient intermediate mechanism sitting between the tasks and the workers to dramatically limit the mapping between the two sets, and thus to avoid computing recommendations from the huge set of all possible pairing of tasks and workers. *Categories* as such a mechanism, a mediator between the set of workers and the set of tasks, thus come into the play. Tasks can be put into various (multi-) categories according to the skills needed, and workers can also be associated with various (multi-) categories per their profiles regarding their expertise/skills, personal interests/preferences, and historical performance. For e-commerce systems, category has little use (except for browsing products) since it is infeasible to limit the buying choice of a customer to a few categories of products. For crowdsourcing systems, categorization of tasks and workers not only makes sense, but is a great deal in facilitating fast recommendation of tasks (for a worker) and workers (for a task requester). In crowdsourcing systems, categories function as effective mediators and provide "short-circuited", but meaningful connections between tasks and workers, which make fast (real-time) recommendations practically achievable. Our two novel algorithms, TOP-K-T and TOP-K-W were thus designed based on leveraging the mediation mechanism of categories, as a result of the afflatus outlined above.

Categorization of tasks is a prerequisite for applications of our proposed algorithms. The literature has many different approaches to achieve satisfactory categorization levels. The latest approach is to use a text classification system to create "filters which allow narrowing down the search results based on predefined filter categories" [11]. Furthermore, the text classification system can be improved with the help of human (online crowd) annotators involved into the training process where different learning techniques can be combined such as ensemble learning and active learning [12]. Categories as an effective mediation mechanism introduced in our algorithms are mainly to boost the recommendation performance and the quality of completed tasks. Other researchers [13] have reported that the worker's perspective is a crucial factor to be considered in crowdsourcing systems.

We may generally consider the top-k task recommendation as computation of a restricted 1-to-K mapping from workers to tasks, and, in reverse, the top-k worker recommendation as computation of a restricted 1-to-K mapping from tasks to workers. In order to efficiently compute these mappings, we design commensurate data structures to facilitate the computation process. These data structures are described below.

We introduce $C \rightarrow T$ (category–task) as an array of n categories, $\{c_1, c_2, \ldots, c_n\}$, where each element c_i points to an array of available tasks in category c_i as illustrated in Fig. 1(a); and $W \rightarrow C$ (worker–category) as an array of m workers, $\{w_1, w_2, \ldots, w_m\}$, where each element w_j points to an array of n categories, $\{c_1, c_2, \ldots, c_n\}$, which each is paired with the worker's matching score $s_{i,j}$ (to be defined shortly) with the corresponding category (as illustrated in Fig. 1(b)). A matching score $s_{i,j}$ measures the historical performance, preference, and expertise of worker w_j with category c_i. For each worker w_j, we keep n matching scores in correspondence to n distinct categories. The list of n categories of each worker w_j $(j \leqslant m)$ is sorted in non-increasing order of the workers' matching scores, $s_{i,j}$ $(1 \leqslant i \leqslant n)$. Expression $W_j \rightarrow C_i$ quickly retrieves the matching score of the ith most preferable task category of worker w_j. For example, $W_1 \rightarrow C_1$ gives the matching score of the first most preferable category of worker w_1. The $W \rightarrow C$ data structure is illustrated in Fig. 1(b) using some randomly set matching scores.

To facilitate top-k worker recommendation, the $W \rightarrow C$ data structure (Fig. 1(b)) alone is not enough since the $W \rightarrow C$ only represents the relationship between workers and categories in one direction (i.e., from workers to categories).



(a) $C \rightarrow T$ (Category-Task)

(b) $W \rightarrow C$ (Worker-Category)

(c) $C \rightarrow W$ (Category-Worker)

Figure 1 Data structures.

We need an additional data structure to conveniently associate qualified workers to corresponding task categories, i.e., $C \rightarrow W$, as illustrated in Fig. 1(c). To avoid redundancy, the worker lists in $C \rightarrow W$ hold pointers to the same entries in $W \rightarrow C$, but ordered differently. In $C \rightarrow W$, each category is associated with a list of workers sorted according to their matching scores in that category, whereas in $W \rightarrow C$, each worker is associated with a list of categories sorted based on the workers' matching scores with those categories.

In the above discussion, we forward referenced the metric term, matching score, which is yet to be defined below. For a given work W_j (i.e., the jth worker) and a task category C_i (i.e., the ith category), the worker's matching score with the tasks of this category is defined by the following equation:

$$S_{i,j} = AR_{i,j} \times CPS_{i,j} \times Similarity(p_j, c_i) \quad (1)$$

The equation defines matching score as a product of three factors, which in turn stands for the acceptance rate, the category preference score, and the profile-category similarity score, which are respectively defined by Eqs. (2)–(4) in the sequel.

$$AR_{i,j} = \frac{AT_{i,j}}{CT_{i,j}} \quad (2)$$

$$CPS_{i,j} = \frac{CT_{i,j}}{TCT_j} \quad (3)$$

$$Similarity(p_j, c_i) = \frac{\sum_{k=1}^{h} p_{j_k} \times c_{i_k}}{\sqrt{\sum_{k=1}^{h} p_{j_k}^2} \times \sqrt{\sum_{k=1}^{h} c_{i_k}^2}} \quad (4)$$

Acceptance Rate ($AR_{i,j}$) is the ratio of the number of accepted tasks to the total number of completed tasks in category c_i by worker w_j. *Category Preference Score* ($CP_{i,j}$) is the percentage of completed tasks in category c_i by worker w_j to the total number of completed tasks by the worker in all categories. *Profile–Category Similarity* is a cosine similarity between a worker's profile p_j (e.g., expertise, certificates, honors, etc.) and the category description c_i. It is worth to note that this score is computed offline after worker registration and recomputed every time the worker's profile is updated. The vector-space model is used where the profile and category are represented by their keywords (h keywords as in Eq. 4). The $tf \times idf$ weighting method proposed in [14] is used to give each keyword a weight, where tf measures the frequency of a keyword t and idf varies inversely with the number of documents that contains t.

3.2. TOP-K-T recommendation algorithm

As the popularity and scale (including both the numbers of tasks and workers) of crowdsourcing systems keep increasing, workers tend to spend more time on finding proper tasks matching their skills and interests than on the tasks themselves [15,8]. The motive of our work is to help workers to instantly find best matching tasks and to help requesters to quickly identify the best workers for their tasks at hand. Our first algorithm, TOP-K-T, was thus designed to make recommendation of the top-k most *suitable* tasks for a worker at real-time speed.

Algorithm 1. TOP-K-T Task Recommendation Algorithm.

Input: $C \rightarrow T$: the category–task data structure.
$W \rightarrow C$: the worker–category data structure.
index: the index of the worker seeking tasks.
k: the number of tasks to be recommended.
Output: L: an array of the top-k tasks for the worker.

```
1  Initialize output array L
2  passtonext ← 0
3  for i ← 1 to W_index → C.size do
4     C_target ← getCategory(W_index → C_i);
5     S_{C_target,W_index} ← getScore(W_index → C_i)
6     numselected ← cascaderound(S_{C_target,W_index} × k);
7     if numselected + passtonext > C_target → T.size then
8        passtonext ← passtonext + numselected − C_target → T.size
9        L ← all tasks in C_target → T
10    else
11       L ← randomly select (numseleted + passtonext) tasks
from C_target → T
12    end
13 end
```

Our TOP-K-T was designed with the assumption that, in a crowdsourcing system, at any given time, there is a huge list of available tasks ($\{t_1, t_2, \ldots, t_s\}$) of which each belongs to certain categories, and a huge list of online workers which each is associated to certain (usually a few) categories. With the support of the data structures introduced earlier, our algorithm TOP-K-T is nearly straightforward, as shown in Algorithm 1.

For a given worker, W_{index}, the algorithm iterates through the categories of the worker maintained in $W_{index} \rightarrow C$ that is sorted in non-increasing order based on the matching scores of the worker's profile with these categories (Line 3 in Algorithm 1). The matching scores of the worker with the associated categories are pre-normalized to sum up to 1. During each iteration, one category of the worker is retrieved from $W_{index} \rightarrow C$ into C_{target} (line 4), and the matching score of the worker with this category is fetched into $S_{C_{target},W_{index}}$ (line 5). The algorithm then evaluates category C_{target} to select some available tasks from that category. The number of tasks to be selected from C_{target} is the result of *cascade rounding* (to be explained shortly via an example) of the product of the worker's normalized matching score with the category and the parameter k (line 6). Herein, two cases need to be differentiated. (i) The number of available tasks in the considered category $C_{target} \rightarrow T$ is less than needed. In this case, all the available tasks in $C_{target} \rightarrow T$ are selected and added into the output task list L (line 9) and the remaining number of tasks yet to be identified is passed to the next iteration (line 8). (ii) The number of available tasks in $C_{target} \rightarrow T$ is more than needed. In this case, the algorithm randomly selects $numselected + passtonext$ tasks from $C_{target} \rightarrow T$ and adds them to output list L (line 11). An illustrative example is shown in Supplementary Table 2, furnished with explanations.

Updating of workers' matching scores. Each time when a worker completes a task, the acceptance rate (defined by Eq. (2)) and the category preference score (defined by Eq. (3)) of the worker ought to be updated, which further propagates to the worker's matching score and calls for resorting of the entries in arrays $C \rightarrow T$ and $C \rightarrow W$. However, it would be too time-consuming to update these scores and the arrays

every time a worker completes a task. Therefore, our approach adopts periodic batch update on these scores and the arrays as we assume that each time when a worker completes one task, the affect on the worker's matching score is marginal. This approach may not satisfy the need of the workers who want the recommendation system to recommend tasks that are similar to their recently chosen and completed ones [13]. This is an issue of trade-off between performance and accuracy of recommendations. A possible solution would be to offer "instant score update" as a user controllable function in the interface. Alternatively, the system may be set to automatically adjust the update pace based on explicit feedback from workers regarding their satisfaction with the recommendations made by the system to them. This information ought to be integrated into the profiles of the workers and used in the future toward more personalized recommendation. The above ideas have not been reflected in our current algorithm.

Dealing with the cold-start problem. New workers face a cold-start problem, which means the matching scores of these workers are not available. This is a serious problem that, if left unattended, new workers will never be recommended a task to start with. To solve this problem, for new workers we count only on the similarity scores (Eq. (4)). Recommendations based on similarity scores reflect the worker's expertise, skills, and personal preferences only. Once a new worker has a certain number of tasks completed, the standard matching score computation (Eq. (1)) will be switched on. Here we assume that a crowdsourcing system requires workers to choose some preferences when they try to register to the system. This requirement is not directly reflected in our algorithm shown in Algorithm 1. Alternatively, new workers may choose to browse the listed tasks, select and work on their interested tasks to mitigate the cold-start problem. Using similarity scores as substitute for the matching scores may not well encourage diversification of task categories since higher similarity scores tend to lock on fewer categories. We propose to differentiate between eager workers and lazy workers, of which, the former are inclined to try new types of tasks even though they may not do their best with, while the latter want to keep working on the types of tasks they used to work on and can do their best with. A partial solution to task diversification for eager workers is to switch our cascade rounding technique (exemplified in Supplementary Table 2) from rounding-up to rounding-down in order to increase the chance of selecting tasks from additional categories.

For a given worker, our TOP-K-T algorithm takes $O(\alpha + k)$ runtime to produce recommendation, where α is the number of task categories covered by the worker's profile and k is the number of tasks to be recommended to the worker. Both parameters α and k are usually small numbers in real-world scenarios. Evidently, algorithm TOP-K-T is extremely time-efficient, regardless of the size of the data flow in a crowdsourcing system; the real-time recommendation of top-k tasks to any given worker can be guaranteed. To recap, our TOP-K-T algorithm achieves its real-time performance owing to the creative incorporation of categories that significantly reduces the search space for best matching tasks; the algorithm also "welcomes" new workers (i.e., dealing with the cold-start problem) by automatic switching of its matching score computation.

3.3. TOP-K-W recommendation algorithm

For a posted task, our TOP-K-W algorithm identifies the top-k most *suitable* (technically qualified and interested) workers to recommend to the requester for soliciting them to work on the task. In order to obtain real-time performance, similar strategy and data structures are used in the algorithm. The recommendation problem faced by our TOP-K-W algorithm is stated as follows: given any task t associated with a weighted list of categories that t belongs to, say, $\{(c_1, \beta_1), (c_2, \beta_2), \ldots, (c_r, \beta_r)\}$, what are the top-$k$ most suitable workers to be recommended to work on the task?

The TOP-K-W algorithm is described in Algorithm 2. The algorithm iterates through the categories of a given task, i.e., list *TaskCat* (starting at line 3 in Algorithm 2). During each iteration, the next category index is fetched from list *TaskCat* into variable C_{target}, and the associated category weight is fetched into variable C_{weight} (lines 4 and 5, respectively). The number of workers selected from C_{target} is decided by the product of the category's weight C_{weight} and the parameter k after applying cascade rounding (line 6). Similar to algorithm TOP-K-T, two specific cases need to be separately addressed, of which the deliberation is omitted due to analogous disposition.

Algorithm 2. TOP-K-W Recommendation Algorithm.

Input: $C \rightarrow W$: the category–worker data structure.
TaskCat: list of the weighted categories of given task.
k: the number of workers to be recommended
Output: L: list of top-k workers for recommendation.

```
1  Initialize output array L
2  passtonext ← 0
3  for i ← 1 to TaskCat.size do
4      C_target ← getCategory(TaskCat[i])
5      C_weight ← getWeight(TaskCat[i])
6      numselected ← cascaderound(C_weight × k)
7      if numselected + passtonext > C_target → W.size then
8          passtonext ← passtonext + numselected − C_target
           → W.size
9          L ← all workers in C_target → W (not already in L)
10     else
11         L ← first (numseleted + passtonext) workers from
           C_target → W (not already in L)
12     end
13 end
```

Special case discussion. The TOP-K-W algorithm may face a situation where all the recommended k workers are busy doing other tasks or unavailable for any other special reasons. Our TOP-K-W algorithm solves this problem by taking the advantage of the already sorted list of workers stored in $C_{index} \rightarrow W$. The algorithm periodically increases the value of k until one of the recommended workers starts working on the task. Instead of simply recommending alternative workers in the subsequent rounds of recommendation, our algorithm retains previously recommended workers in the recommendation list in order to

retain the opportunity of getting those high-ranked workers as their statuses may change soon.

In summary, the TOP-K-W algorithm takes $O(\lambda + k)$ runtime, where λ is the number of categories that a given task belongs to and k is the number of workers to be recommended for the task. It is evident that both λ and k are usually small numbers in real-world scenarios. Our TOP-K-W algorithm is able to obtain real-time performance (in less than a millisecond per our experiments, to be detailed shortly), regardless of the potentially huge volume of data flow typically found in crowdsourcing systems. The TOP-K-W algorithm adopts an incremental recommendation strategy to bring down the possible delay of task completion to the minimum. Similar to algorithm TOP-K-T, this algorithm also assumes offline batch update on workers' matching scores and resorting of the worker list under each category in order to deliver real-time performance.

4. Experimental results

In this section, we evaluate our proposed algorithms, TOP-K-T and TOP-K-W, through experimental study. As currently (to the best of our knowledge), there are no *applicable* datasets gathered from real-world crowdsourcing systems that are publicly available and fit the need of the presented work, we conducted our study based on synthesized datasets. To make synthesized datasets realistic and representative, we generated our datasets at various scales and at every step we tried to mimic the scenarios in a real-world crowdsourcing system. Table 1 shows the characteristics of generated datasets. Uniform distribution had been assumed at several places during the process of generating these synthesized datasets. Supplementary Figs. 3–6 show the distribution features of a sample of the synthesized datasets. Our proposed algorithms and the data generation are implemented using C#. All experiments are conducted on a PC with Intel Xeon 2.40 GHz processors and 16 GB DDR3 RAM in a light load condition. Every experiment presented below is conducted twenty times and the average running time is computed. In the following, we present the experiment results of algorithms TOP-K-T and TOP-K-W, respectively.

4.1. TOP-K-T

As mentioned earlier, the theoretical time complexity of the TOP-K-T algorithm is $O(\alpha + k)$, where α is the number of worker's categories and k is the number of tasks to be recommended. To fairly evaluate the performance of this algorithm, we select three different types of workers, i.e., workers with maximum number of categories (MAXworker), workers with average number of categories (AVGworker), and workers with minimum number of categories (MINworker). Besides, we consider three different values for k, namely, 1, 20, and 50, to see how different values of k would affect the performance of TOP-K-T.

Table 1 Synthesized datasets and characteristics.

Dataset	#Workers	#Categories	#Completed tasks	#Available tasks
DS1	100,000	300	178,898,799	80,710
DS2	300,000	500	974,854,896	490,815
DS3	500,000	600	1,908,867,719	960,814
DS4	1,000,000	1000	3,127,310,923	1,396,721

Fig. 2(a) shows the performance of TOP-K-T with the parameter k set to 1 (i.e., to recommend only one task to each worker). As shown in the figure, the runtimes are basically constant in terms of dataset sizes, but with noticeable fluctuations in the range from 132 to 143 ns. The reason that caused the fluctuations is the randomness in the numbers of categories of workers, the numbers of tasks of categories, etc. For example, when a preferred task category does not have any or does not have enough available tasks, the algorithm needs to run additional iterations to selects tasks from subsequent categories. Fig. 2(b) shows the performance of TOP-K-T with k set to 20 (i.e., to recommend top 20 tasks for each worker). The 20 tasks are selected from different categories, and the running time increases as the number of categories probed increases. For example, the case with MINworker runs the fastest as it involves the minimum number of categories to process. Fig. 2(c) shows the performance of TOP-K-T with k set to 50 (i.e., to recommend top 50 tasks to each worker). As expected, the times taken accordingly increase when k increases from 20 to 50. The performance plots show basically constant performance with regard to varied dataset sizes and our explanation for the performance data is basically the same as with the cases of smaller k values. The only thing we would like to point out herein is that when k is 50, which is pretty large in real-world scenarios, our TOP-K-T algorithm remains extremely efficient, taking up to only a couple of milliseconds in our experiments.

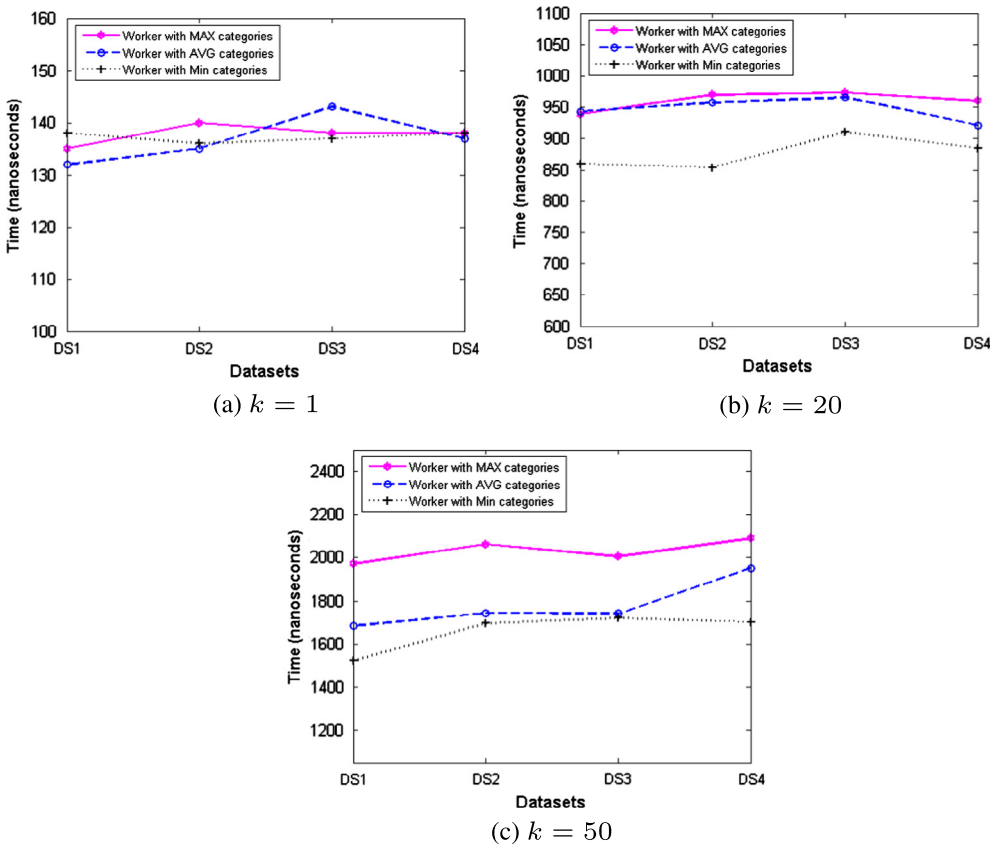
Analytically, the running time of our TOP-K-T algorithm is affected only by α and k which are typically very small numbers, regardless of the data sizes (the numbers of workers, tasks, etc.). Our experimental study confirms the validness and the constant time performance of our algorithm.

4.2. TOP-K-W

As pointed out before, our TOP-K-W algorithm takes $O(\lambda + k)$ time, where λ is the number of categories that a given task belongs to and k is the number of workers to be recommended to the task requester. To evaluate the algorithm we assume the following scenario: (1) a requester posts 5 tasks $(t_1, t_2, t_3, t_4, t_5)$ and sets parameter k to 3, 10, 20, 30, and 50, respectively; (2) the tasks, t_1, t_2, t_3, t_4, and t_5 respectively belong to 1 category, 4 categories, 7 categories, 3 categories, and 10 categories. Fig. 3 shows the performance plots of TOP-K-W with regard to the above assumptions. It can be easily observed from Fig. 3 that the running times are basically constant in terms of dataset sizes, affected only by parameters λ and k (both in practice are very small numbers). It can also be observed that the sizes of the datasets do not have noticeable influence on the running time of the algorithm as the time plots all appear to be constant plots with regard to varied dataset sizes.

5. Related works

The study of task recommendation in crowdsourcing systems has been growing to such a point of forming a distinct disci-



(a) $k = 1$ (b) $k = 20$

(c) $k = 50$

Figure 2 Performance of TOP-K-T with $k = 1$, 2 and 50.

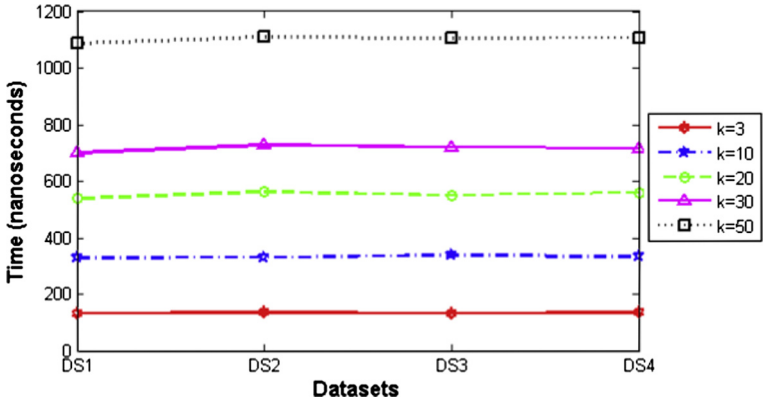


Figure 3 Performance of TOP-K-W with varied k values.

pline with its own identity and merits. Several representative approaches [15–18] have been proposed to tackle the recommendation problem in crowdsourcing systems. In this section, while we review these related works, we make comparisons with ours.

Our argument for effective, real-time recommendation of top-k tasks and top-k workers in crowdsourcing systems is not alone. Ipeirotis [15] examined the task posting and completion activities on AMT, and concluded that AMT is a heavy-tailed market, i.e., it has a heavy-tailed distribution of both the completion time and posting time, as illustrated in Supplementary Fig. 1, where the number of tasks arrived on November 10th, 2013 reached 50,000, followed by 400,000 on the next day. The tasks in crowdsourcing systems include micro-tasks, particularly on AMT, that have very short life spans, e.g., from minutes (if not seconds) to hours. These unique features of crowdsourcing systems, i.e., huge flow of tasks with very short life spans, make traditional recommendation algorithms inapplicable simply because they were not designed and are unable to deliver the desired real-time recommendation performance by most crowdsourcing systems.

Ambati et al. [16] proposed a task recommendation approach based on a classification technique. Their proposed approach first generates a worker model by acquiring the

worker's performance information, based on which, a classifier is then trained to classify the available tasks as interesting or uninteresting to a given worker. Per our understanding, this approach may suffer with the following three issues: (1) similarity computed based only on task description may not adequately capture the true nature of the tasks since two tasks with similar keywords may not be similar in nature and the skills needed; (2) this approach relies on a carefully-selected, balanced training set which is very hard to obtain for emerging research topics such as crowdsourcing; and (3) this approach may also suffer from poor scalability when applied to really large crowdsourcing systems.

The work reported in [17] improves in [16] by additionally incorporating a worker's task selection history besides the worker's performance history. The proposed algorithm in [17] assumes that there is a set of categories predefined; and any task posted by a requester can be categorized into one of those categories. The algorithm recommends to a worker a list of tasks sorted according to the worker's preferences and the acceptance tendency of the worker's completed tasks by the requesters. Comparing to our work, this approach has the following limitations: (1) limited scalability since it iterates through *all available tasks in all categories* (which can be enormous) every time a worker logs into the system and needs to updates the worker's scores every time s/he completes a task and (2) this approach has the cold-start problem.

The same group extended their algorithm in [18]. In the extended version, a worker-task matrix is used where each entry in the matrix has a value from 1 to 5. The main goal of this approach is to predict the missing values in the worker-task matrix. This approach employs matrix factorization technique to understand the worker's preference on the tasks. Comparing to our work, this approach has the following concerns: (1) this approach may still suffer from scalability issues since it records all interactions between all the workers and the system, and yet, its matrix is expanding rapidly which makes the relearning of the matrix much harder to handle and (2) it is difficult to obtain the worker's task searching history since it is only accessible to the crowdsourcing systems' administrators as the authors pointed [18].

6. Conclusion and future work

With the increased popularity and scales, crowdsourcing systems involve a flood of data which could leave the workers and requesters at dismay when they (as workers) are trying to find suitable tasks to work on or (as requesters) to find the best workers for their tasks. Therefore, making the good recommendation on the fly has become critical to these systems. In this article, we revealed our insight into the essential difference between the tasks in crowdsourcing systems and the products/items in e-commerce markets, and the difference between a buyer's interest in products/items and a worker's interest in tasks. Our insight inspired us to bring up *categories* as a key mediation mechanism between workers and tasks, which has been proven an highly effective means in our effort toward designing extremely scalable and efficient recommendation algorithms for crowdsourcing systems. Our effort has resulted in two novel algorithms, TOP-K-T (computing the top-k most suitable tasks to recommend to a worker) and TOP-K-W (computing the top-k best workers to recommend to a task requester). Both algorithms demonstrate superb (real-time) performance — make valid recommendations in just a few milliseconds regardless of dataset sizes, which explains the great scalability and efficiency of our algorithms. Besides *categories* as a general mediation mechanism, our various data structures (illustrated in Fig. 1(a)–(c)) provide instrumental support to the implementation of our approaches. These data structures absorb a major part of the intrinsic complexities of the recommendation problems, and render us succinct algorithms with great efficiency and scalability. We have done extensive experimental study of our algorithms with synthesized datasets because no suitable real dataset is available for our study. We did not do horizontal comparison with related algorithms in the experimental study as our algorithms are quite different in nature from all other related algorithms, and are evidently superb to them, which makes equal-footing empirical comparison with them less interesting and unnecessary.

As part of our future work, we are looking forward to obtaining real datasets from the existing crowdsourcing systems such as AMT to further evaluate our algorithms. Meanwhile, we plan to take workers' rewards and preferred work times as additional factors into our recommendation framework in order to make our recommendations more appealing to both workers and requesters.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.aci.2016.01.001.

References

[1] D. Che, M. Safran, Z. Peng, From big data to big data mining: challenges, issues, and opportunities, Database Systems for Advanced Applications, vol. 7827, Springer, Berlin, Heidelberg, 2013, pp. 1–15 (Chapter 1).
[2] B. Sarwar, G. Karypis, J. Konstan, J. Reidl, Item-based collaborative filtering recommendation algorithms, in: 10th International Conference on World Wide Web, Hong Kong, Hong Kong, 2001, pp. 285–295.
[3] G. Karypis, Evaluation of item-based top-n recommendation algorithms, in: 10th International Conference on Information and Knowledge Management, 2001, pp. 247–254.
[4] E. Estellés-Arolas, F. González-Ladrón-De-Guevara, Towards an integrated crowdsourcing definition, Inf. Sci. 38 (2) (2012) 189–200.
[5] A. Doan, R. Ramakrishnan, A. Halevy, Crowdsourcing systems on the world-wide web, Commun. ACM 54 (4) (2011) 86–96.
[6] D. Brabham, Crowdsourcing as a model for problem solving: an introduction and cases, Convergence: Int. J. Res. New Media Technol. 14 (1) (2008) 75–90.
[7] D. Geiger, M. Schader, Personalized task recommendation in crowdsourcing information systems current state of the art, Decis. Support Syst. 65 (2014) 3–16.
[8] L. Chilton, J. Horton, R. Miller, S. Azenkot, Task search in a human computation market, in: The ACM SIGKDD Workshop on Human Computation, New York, NY, USA, 2010, pp. 1–9.
[9] Y. Koren, R. Bell, C. Volinsky, Matrix factorization techniques for recommender systems, Computer 42 (8) (2009) 30–37.
[10] M. Deshpande, G. Karypis, Item-based top-n recommendation algorithms, ACM Trans. Inf. Syst. 22 (1) (2004) 143–177.

[11] S. Schmidt, S. Schnitzer, C. Rensing, Text classification based filters for a domain-specific search engine, Comput. Ind. (in press), 2015 (http://dx.doi.org/10.1016/j.compind.2015.10.004).
[12] S. Schnitzer, S. Schmidt, C. Rensing, B. Harriehausen-Muhlabauer, Combining active and ensemble learning for efficient classification of web documents, Polibits 49 (2014) 39–45.
[13] S. Schnitzer, C. Rensing, S. Schmidt, K. Borchert, M. Hirth, P. Tran-Gia, Demands on task recommendation in crowdsourcing platforms – the workers perspective, in: CrowdRec Workershop, ACM RecSys, 2015.
[14] G. Salton, C. Buckely, Term-weighting approaches in automatic text retrieval, Inf. Process. Manage. 24 (5) (1988) 513–523.
[15] P. Ipeirotis, Analyzing the Amazon Mechanical Turk marketplace, ACM XRDS 17 (2) (2010) 16–21.
[16] V. Ambati, S. Vogel, J. Carbonell, Towards task recommendation in micro-task markets, in: The 25th AAAI Workshop in Human Computation, AAAI, 2011.
[17] M.C. Yuen, I. King, K.S. Leung, Task matching in crowdsourcing, in: The 4th IEEE International Conference on Cyber, Physical and Social Computing, IEEE Computer Society, 2011, pp. 409–412.
[18] M.C. Yuen, I. King, K.S. Leung, Task recommendation in crowdsourcing systems, in: ACM KDD Workshop on Data Mining and Knowledge Discovery with Crowdsourcing, 2012.