FROG

A Fast and Reliable Crowdsourcing Framework

Outline

Crowdsourcing

Definition of Crowdsourcing

Crowdsourcing is a sourcing model in which individuals or organizations obtain goods and services, including ideas and finances, from a large, relatively open and often rapidly-evolving group of internet users; it divides work between participants to achieve a cumulative result.

Problems in Crowdsourcing Systems

- Accuracy Problem
- Latency Problem
- (May) Lack of active workers

Accuracy and Latency Problems

Worker	Task	Answer	Time Latency	Correctness
w_1	t_1	Normal	8 s	×
w_1	t_2	Accident	9 s	×
w_1	t_3	Accident	12 s	√
w_2	t_1	Accident	15 s	×
w_2	t_2	Normal	**5 min**	√
w_2	t_3	Normal	10 s	×
w_2	t_4	Accident	9 s	√
w_2	t_5	Accident	14 s	×
w_3	t_4	Accident	8 s	√
w_3	t_5	Normal	11 s	√

Figure: Example of Accuracy and Latency Problems in the Crowdsourcing System

High Latency: worker w_2 tags t_2 with **5 minutes**
Low Accuracy: worker w_2 tags 5 pictures with **3 wrong labels**

The Problems

- minimize the maximal latencies for all tasks
- maximize the accuracies (quality) of task results
- invite more offline workers to perform online tasks when the system lacks of active workers

Existing Studies

- **[Rosenblatt 1956] and [Verroios 2015]** increases prices over time to encourage workers to accept tasks to reduce the latency for **specific tasks**
- **iCrowd and CLAMShell** removes low-accuracy or high-latency workers, which may lead to **idleness of workers** and **low throughput of the system**

The FROG Approach

Two important components: *task scheduler* and *notification module*

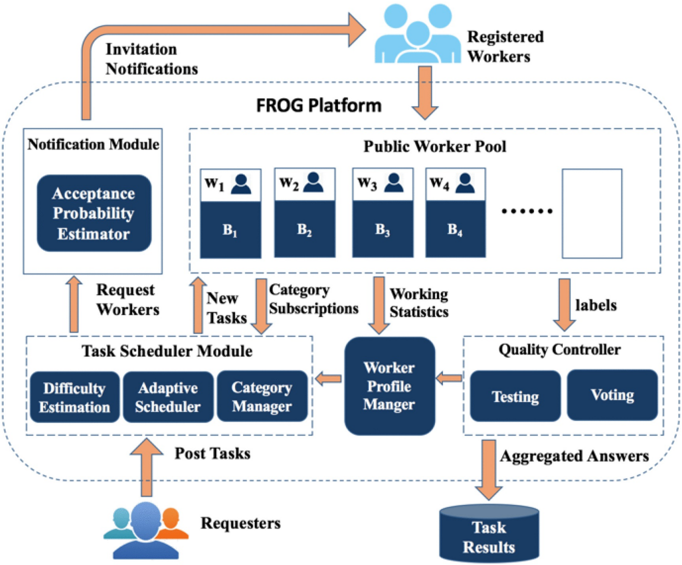
The Task Scheduler

- actively schedules tasks for workers using heuristic algorithms
- considering both accuracy and latency

The Notification Module

- notify offiine workers via invitation messages
- propose a *smooth kernel density model* to estimate the probabilities that workers can accept task invitations

Illustration of the FROG framework



Symbols and Descriptions

Symbol	Description
C	a set of task categories c_i
T	a set of tasks t_i
W	a set of workers w_j
$t_i.c$	the category of task t_i
q_i	a specific quality value of task t_i
s_i	the start time of task t_i
f_i	the finish time of task t_i
a_{jl}	the category accuracy of worker w_j on tasks in category c_l
r_{jl}	the response time of worker w_j on tasks in category c_l

Outline

Tasks

- $T = \{t1, t2, \cdots, t_m\}$ is a set of m tasks in the crowdsourcing platform
- Each task t_i belongs to a task category, $ti.c \in C$
- Each task t_i arrives at the system at s_i
- Each task t_i is associated with a user-specified quality threshold q_i, which is the expected probability that the final result for task t_i is correct

Workers

- $W = \{w_1, w_2, \cdots, w_n\}$ is a set of n workers
- For tasks in category c_l, each worker w_j is associated with an accuracy, a_{jl}, that w_j do tasks in category c_l, and a response time, r_{jl}

Expected Accuracy

The expected accuracy of task t_i is:

$$\Pr\left(W_i, c_l\right) = \sum_{x=\lceil \frac{k}{2} \rceil}^{k} \sum_{W_{i,x}} \left(\prod_{w_j \in W_{i,x}} \alpha_{jl} \prod_{w_j \in W_i - W_{i,x}} \left(1 - \alpha_{jl}\right) \right)$$

Where W_i is the set of k workers that do task t_i, c_l is the category that task t_i belongs to.

The FROG-TS Problem

Given a set T of m crowdsourcing tasks, and n workers in W, the problem of FROG task scheduling(FROG-TS) is to assign workers $w_j \in W$ to tasks $t_i \in T$, such that:

1. the accuracy $Pr(W_i, c_l)$ of task t_i is not lower than the required accuracy threshold q_i
2. the maximum latency $max(l_i)$ of tasks in T is minimized, where $l_i = f_i - s_i$ is the latency of task t_i, that is, the duration from the time s_i task t_i is posted in the system to the time, f_i, task t_i is completed.

The FROG-TS problem is NP-hard

By reducing it from the multiprocessor scheduling problem, we find that the FROG-TS problem is NP-hard.
Due to the NP-hardness of the problem, an adaptive task routing approach is introduced to efficiently retrieve the answers.

Request-Based Scheduling (RBS) Approach

Strategy: return the task with the highest delay probability to the worker

Algorithm 1. GreedyRequest(W, T)

Input: A worker w_j requesting for his/her next task and a set $T = \{t_1, t_2, \ldots, t_v\}$ of v uncompleted tasks

Output: Returned task t_i

foreach *task* t_i ***in*** T **do**
    calculate the delay possibility value of t_i with Eq. (9);
select one task t_i with the highest delay probability;
if *the expected accuracy of* t_i *is higher than* q_i **then**
    Remove t_i from T;
return t_i;

Where the delay probability $L(t_i)$ of task t_i is estimated by:

$$L(t_i) \propto (d_i \cdot q_i)^{\left\lceil \frac{\epsilon_{\max} - \epsilon_i}{r_l} \right\rceil}$$

where d_i is the difficulty of task t_i and ϵ_i is the time lapse of task t_i.[1]

[1] For more information, refer to the original paper.

Batch-Based Scheduling (BBS) Approach

Strategy: assign high-accuracy workers to difficult and urgent tasks and low-accuracy workers with easy and not that urgent tasks

Algorithm 2. GreedyBatch(W, T)

Input: A set, $T = \{t_1, t_2, \ldots, t_m\}$, of m unfinished tasks and a set, $W = \{w_1, w_2, \ldots, w_n\}$, of n workers

Output: Assignment $\mathbb{A} = \{\langle t_i, w_j \rangle\}$

$\mathbb{A} \leftarrow \emptyset$;
foreach *task* t_i ***in*** T **do**
    calculate the delay possibility value of t_i with Eq. (9);
while $T \neq \emptyset$ ***and*** $W \neq \emptyset$ **do**
    select task t_i with the highest delay probability value;
    remove t_i from T;
    $W_o \leftarrow$ MinWorkerSetSelection(t_i, W, W_i);
    if $W_o \neq \emptyset$ **then**
        foreach $w_j \in W_o$ **do**
        Insert $\langle t_i, w_j \rangle$ into $\mathbb{A}$;
        if w_j cannot be assigned with more tasks **then**
            Remove w_j from W;
return $\mathbb{A}$;

1

[1] `MinWorkerSetSelection` selects a minimum set of workers satisfying the constraint of the quality threshold for task t_i

Time Complexity

Request-Based Scheduling Approach

Assume that each task has received h answers. Then with v uncompleted tasks, the time complexity of RBS algorithm is

$$O(v \cdot h)$$

Batch-Based Scheduling Approach

Assume that each task t_i needs to be answered by h workers, then the time complexity of BBS alrogithm is

$$O(m \cdot h)$$

Outline

Worker Dominance

Given two worker candidates w_x and w_y, we say worker w_x dominates worker w_y, if it holds that:

1. $P_{ts}(w_x) > P_{ts}(w_y)$
2. $\alpha_x > \alpha_y$
3. $r_x \leq r_y$

where $P_{ts}(wj)$ is the probability that worker w_j is available, and a_x and r_x are the average accuracy and response time of worker w_x on his/her subscribed categories,respectively.

Estimation for Worker Availability

We can estimate $P_{ts}(w_x)$ using the following methods:

Kernel Density Estimation

- estimate the probability that a worker is available based on one's historical records
- has **"cold-start" problem**

Smooth Kernel Density Estimation

- for each worker, use historical data of his/her friends to supplement/predict his/her behaviors
- avoids "cold-start" problem

Efficient Worker Notifying Problem

Given a timestamp t_s, a set W of n offiine workers, the historical online records $E_j = \{e_1, e_2, \cdots, e_n\}$ of each worker w_j, and the number, u, of workers that we need to recruit for the public worker pool, the problem of efficient worker notifying is to **select a subset of workers in W with high accuracies and low latencies to send invitation messages**, such that:

1. The expected number, $E(P_{ts}(W))$ of workers who accept the invitations is greater than u
2. The number of workers in W, to whom we send notifications, is minimized, where $P_{ts}()$ is the probability of workers to accept invitations and log in the worker pool at timestamp ts.

Solving of the EWN Problem

Algorithm 4. WorkerNotify(W, T)

Input: A set, $W = \{w_1, w_2, \ldots, w_n\}$, of offline workers, the expected number, u, of acceptance workers, and the current timestamp ts

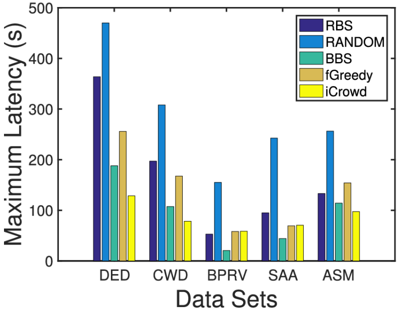
Output: A set, W_n, of workers to be invited

$W_n = \emptyset$;
foreach *worker* w_j ***in*** W **do**
  calculate the ranking score $R(w_j)$ of w_j;
while $u > 0$ ***and*** $|W| > 0$ **do**
  select one worker w_j with the highest ranking score in W;
  $W = W - \{w_j\}$;
  W_n.add(w_j);
  $u = u - P_{ts}(w_j)$
return W_n;

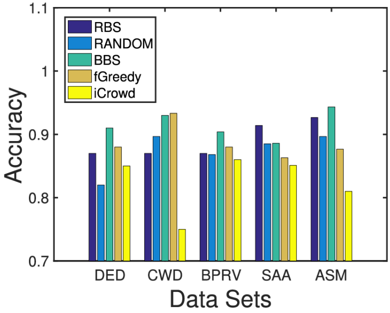
Where the ranking score $R(w_j)$ represents the dominance of the worker w_j.

Outline

Performance of Task Scheduler Module

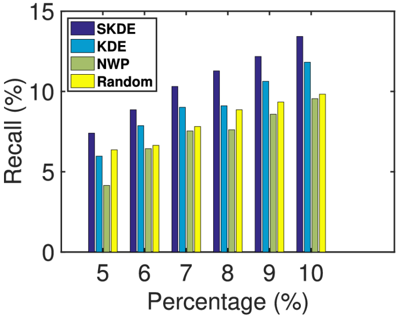


(a) Maximum Latency

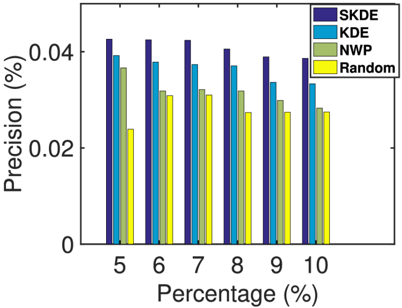


(b) Average Accuracy

Performance of the Notification Module

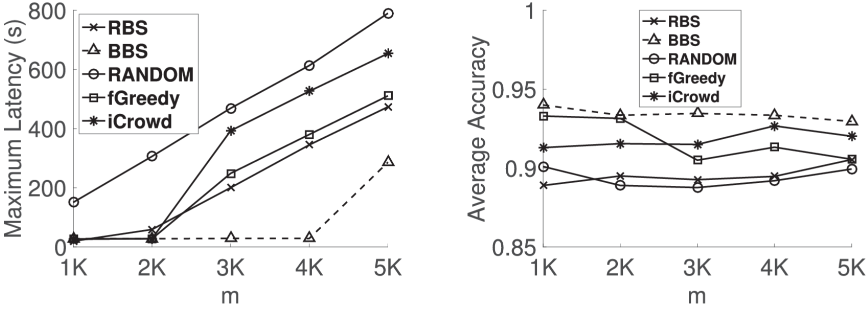


(a) Recall



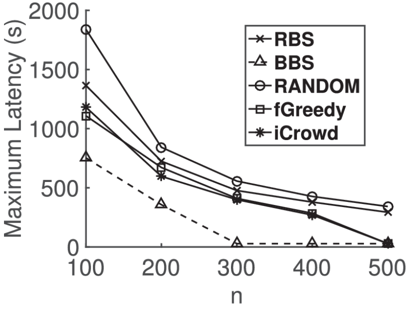
(b) Precision

Effect of the number of tasks m

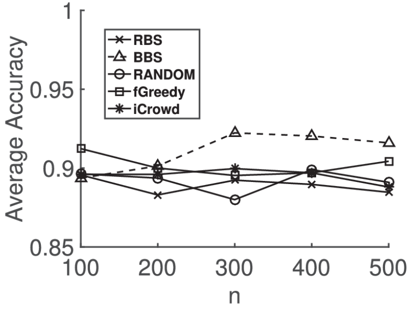


(a) Maximum Latency (b) Average Accuracy

Effect of the number of workers n

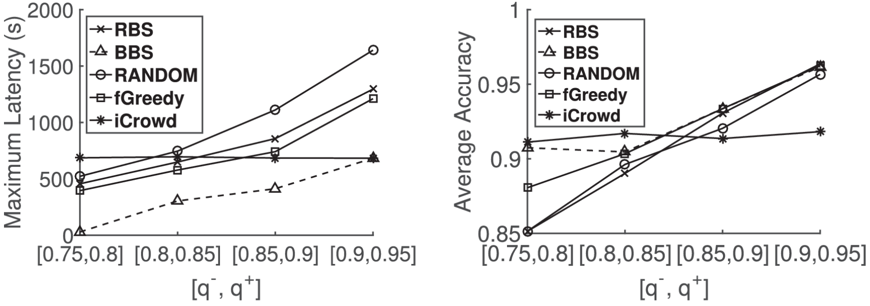


(a) Maximum Latency



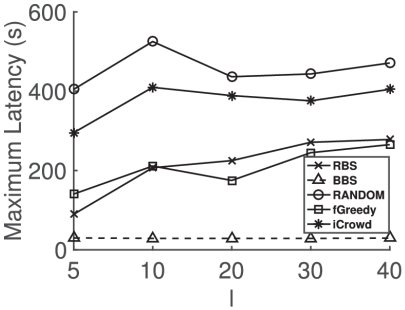
(b) Average Accuracy

Effect of the specific quality value range $[q^-, q^+]$

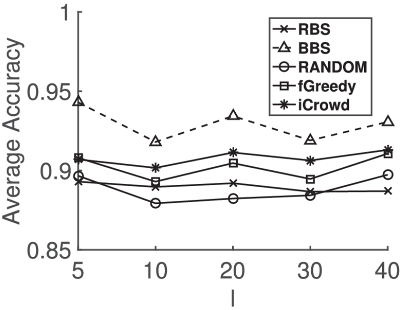


(a) Maximum Latency (b) Average Accuracy

Effect of the number of categories l



(a) Maximum Latency



(b) Average Accuracy

Outline

Conclusion

In this paper, the authors

- propose a novel fast and reliable crowdsourcing(FROG) framework
- formalize the FROG task scheduling(FROG-TS) and efficient worker notifying (EWN) problems
- proposed effective and efficient approaches to solve the problems
- demonstrate the effectiveness and efficiency of our proposed FROG framework on both real and synthetic data sets

Thank You