Optimizing Plurality for Human Intelligence Tasks

Luyi Mo†, Reynold Cheng†, Ben Kao†, Xuan S. Yang†, Chenghui Ren†,
Siyu Lei†, David W. Cheung†, Eric Lo‡

† *University of Hong Kong, Pokfulam Road, Hong Kong*
‡ *Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong*
†*{lymo, ckcheng, kao, xyang2, chren, sylei, dcheung}@cs.hku.hk* ‡ *ericlo@comp.polyu.edu.hk*

ABSTRACT

In a crowdsourcing system, Human Intelligence Tasks (*HITs*) (e.g., translating sentences, matching photos, tagging videos with keywords) can be conveniently specified. HITs are made available to a large pool of *workers*, who are paid upon completing the HITs they have selected. Since workers may have different capabilities, some difficult HITs may not be satisfactorily performed by a single worker. If more workers are employed to perform a HIT, the quality of the HIT's answer could be statistically improved. Given a set of HITs and a fixed "budget", we address the important problem of determining the number of workers (or *plurality*) of each HIT so that the overall answer quality is optimized. We propose a dynamic programming (DP) algorithm for solving the *plurality assignment problem* (PAP). We identify two interesting properties, namely, *monotonicity* and *diminishing return*, which are satisfied by a HIT if the quality of the HIT's answer increases monotonically at a decreasing rate with its plurality. We show for HITs that satisfy the two properties (e.g., multiple-choice-question HITs), the PAP is approximable. We propose an efficient greedy algorithm for such case. We conduct extensive experiments on synthetic and real datasets to evaluate our algorithms. Our experiments show that our greedy algorithm provides close-to-optimal solutions in practice.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications

Keywords

Crowdsourcing, Data Quality

1. INTRODUCTION

Recently, there has been a rise of interest in *crowdsourcing systems*, such as the *Amazon Mechanical Turk* (AMT)[1] and CrowdFlower[2]. These Internet-based systems harness human effort to solve problems that are easy for human beings but difficult for computers [3, 6–8]. Typically, a *requester* announces a set of *Human Intelligent Tasks* (or *HITs*). Through a user interface, a *worker* selects her preferred HIT(s) and submits her *answers* to the system. If the requester is satisfied about the answers, the worker is duly rewarded. Example HITs include question answering [3, 8, 16], entity resolution [6, 19, 20], sorting and join [7, 9], data filtering [15], and image tagging [21].

For HIT answers to be useful, the workers involved need to perform the tasks well. In practice, however, workers could be casual Internet users and their answers are hardly perfect [3, 6–8]; they may make careless mistakes or misinterpret the HIT requirements. To improve the quality of a HIT's answer, a requester is suggested to assign a sufficient number of workers to the HIT [1, 3, 6–8]. In AMT, for instance, a requester is asked to specify the *plurality* of a HIT, which is the number of workers required to perform that HIT. With multiple workers, their answers can be *combined* to derive a higher-quality one for a HIT. We call the combined answer the *result* of the HIT. For example, given a HIT of a binary question (such as True or False), the *result* of the HIT could be taken as the most frequent answer given by the workers of the HIT. Statistically speaking, more workers working on a HIT gives a higher-quality result of the HIT because the impact of wrong answers are much reduced by the correct ones, if the latter outnumbers the former. In fact, worker multiplicity is advised by AMT [1] as well as in many other previous works [3, 6–8].

In practice, plurality has to be limited because: (1) a HIT is associated with a cost for paying the worker and the crowdsourcing system; (2) a requester may only have a limited budget for rewarding workers; and (3) a requester has to spend some time to verify the HIT results. An interesting question is thus about how one could wisely assign just the right pluralities to HITs of various formats, difficulties, and costs, to achieve overall high-quality results, subject to a budget constraint. We call this problem the *plurality assignment problem* (PAP). As crowdsourcing systems are becoming more popular, larger in scale, and with a richer variety of HITs, we anticipate that plurality assignment will become an essential component of crowdsourcing systems. So far, however, few works (e.g., [2, 3, 8, 14]) have addressed the problem.

Manually assigning pluralities to HITs is tedious if not infeasible. As an example, we inspected the HITs that are available on AMT on 28th October 2012. (Let's call a requester a "heavy requester" if she submits a large number of HITs.) We observed that the top-10 heaviest requesters together submitted about 90,000 HITs. It is thus not uncommon that a requester has to handle thousands of HITs. Our goal is to develop algorithms for automating the process of plurality assignment, considering the various properties

[1] `https://www.mturk.com`
[2] `http://crowdflower.com`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.
Copyright 2013 ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505755.

of HITs, such as their costs and difficulties, under a given budget constraint.

Among the various types of HITs, *multiple choice questions* (MCQs) are the most popular ones. For example, among all the HITs on AMT on 28th October 2012, more than three quarters are MCQs. Given an MCQ, workers are asked to make a choice among a set of given ones. For instance, a sentiment-analysis HIT contains a sentence, and a worker needs to indicate her opinion (*positive*, *neutral*, or *negative*) about it. Other examples include categorizing objects (e.g., choosing the best category for an image) and assigning rating scores to items. One way to combine workers' answers to an MCQ to obtain a result is *half voting*. Specifically, the choice that is selected by more than half of the workers is taken as the MCQ's result. Due to its popularity, we illustrate our solution framework using MCQ as an example. We remark that our framework can be extended to address other kinds of HITs.

Recall that the objective of plurality assignment is to maximize the overall quality of a set of HITs' results. We therefore need a quantitative measure of a result's quality. For an MCQ, we measure its result's quality (or simply the MCQ's quality) by the likelihood that the result is correct (see Section 3.2). We note that an MCQ's quality generally improves with (1) its plurality and (2) the *accuracies* of its workers. That is, more and better workers improves the MCQ's quality. As a result, we formulate an MCQ's quality as a function of the above two factors. To tackle the plurality assignment problem (PAP), we develop a dynamic programming algorithm `DP`. Given a set of MCQs, `DP` takes the HITs' quality functions and a budget as input and determines the optimal plurality assignment for the MCQs.

Although `DP` gives optimal solutions to PAP, it is not very efficient especially for HIT sets that contain thousands of HITs. For example, for a set of 60,000 HITs extracted from AMT, `DP` takes over 10 hours to execute. In a large crowdsourcing system (e.g., AMT) that manages HITs from many heavy requesters, efficiency becomes an important issue. We have made two interesting observations of a crowdsourcing system which lead to significant speedups in solving PAP. First, we found that an MCQ's quality function possesses two interesting properties: (1) *monotonicity*: the quality function increases with plurality; and (2) *diminishing return*: the rate of quality improvement drops with plurality. We make use of these characteristics of MCQs to design an approximation algorithm `Greedy`, which is much more efficient than `DP`. We show the theoretical approximation ratio of `Greedy`. We also show that `Greedy`'s solution is very close to the optimal solution in practice. Second, we observe that many HITs submitted by the same requester are given the same cost and that these HITs are of very similar nature. For example, a requester performing sentiment analysis may submit numerous HITs, each one being a tweet to be labeled with an opinion (positive, negative, neutral). It is reasonable to assume that these HITs share the same quality function. If we group HITs of the same cost and quality function as a group, then each HIT in the group should be given more or less the same plurality. We exploit this observation and use a "grouping technique" to effectively reduce the problem size of PAP. This technique can be applied to both `DP` and `Greedy` to improve their efficiency.

We have performed extensive experiments to evaluate our approaches on a synthetic dataset. We examine the effectiveness and the efficiency of our algorithms over a large number of HITs and budget values. (By effectiveness, we refer to the HITs' quality as a result of a given plurality assignment computed by an algorithm.) We have also developed a system to collect real data from human users. We found that both `DP` and `Greedy` are on average 20% more effective than other simple approaches (e.g., evenly assigning budget to HITs). Moreover, `Greedy` is about a thousand times faster than `DP`, and the group-based versions of `DP` and `Greedy` are about ten times faster than their non-group-based counterparts.

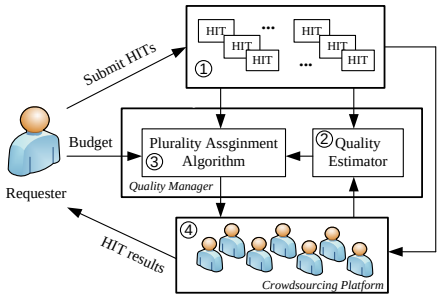


Figure 1: Solution framework.

Figure 1 shows the framework of our approach. (Step 1) A requester sends her HITs and budget information to the *quality manager*, which decides the plurality of the HITs. (Step 2) The manager invokes the *quality estimator*, which keeps track of the accuracy statistics of a pool of workers. (Step 3) Our algorithm, which computes the plurality of each HIT, is then invoked. (Step 4) The HITs are installed in the crowdsourcing system. In this paper we focus on the design of plurality assignment algorithms.

The rest of the paper is as organized as follows. Section 2 discusses the related works. In Section 3 we describe the plurality assignment problem. Section 4 describes various plurality assignment algorithms. We show our experimental results in Section 5. Section 6 discusses how our framework can be extended to address other kinds of HITs. Section 7 concludes the paper.

2. RELATED WORK

Databases for crowdsourcing. To meet the needs of managing large amounts of data collected from crowdsourcing systems, database prototypes such as CrowdDB [6] and Qurk [10, 11] have been developed recently. These systems provide native language support for crowdsourcing, where query operators can be invoked to collect information from workers. In [9], the authors study a database that utilizes workers to compare or to rate database items, and to perform sort and join operations. It is interesting to examine how these systems should operate under the various resource constraints (e.g., limited requester budget, number of workers, and their accuracy). Our work on plurality assignment can very well be applied to a crowdsourcing database to tackle this problem.

Designing HIT questions. A few works have recently addressed the management of resource constraints in crowdsourcing systems. In [16], the authors study human-assisted graph search, and propose algorithms to generate the optimal set of questions. In [19,20], algorithms are developed to derive the minimum set of questions for supporting entity resolution. In [7], the authors study how to generate a set of paired-entity-comparison questions that optimally identify the entity with the maximum value under a limited budget. [15] studies the problem of filtering data using human resources, and designs effective strategies to optimize expected cost and error. In [14], the problem of deciding which items to label, with the aim of improving the model used for active learning, is studied. Notice that these works focus on the design of HITs (i.e., *what* questions to ask). On the other hand, we focus on *how* to obtain high-quality results for HITs under a limited budget by running a plurality assignment algorithm.

Determining plurality. Although the problem of determining the pluralities of HITs has recently attracted some attention, there

are only a few works that have been done for specific types of HITs. These include MCQs [2, 8], binary questions [3], and labeling [14]. In [2, 8], the authors show how to determine the minimum plurality of an MCQ such that a user-given quality threshold can be achieved. Our work differs from [2, 8] in two ways. First, while they deal with a quality threshold as a constraint, we deal with a budget constraint. In some cases, it is more natural and is easier for a requester to specify a budget (in dollar amount) than to specify a quality threshold (in probability). Our work thus complement that of [2,8]. Second, [8] assumes that all MCQs are answered correctly with the same probability, and hence the same plurality is assigned to all the MCQs. We do not make that assumption and consider assigning (different) pluralities to a set of different MCQs.

In [3], the authors address binary question (or *BQ*), which contains two possible answers only. An example BQ is similarity comparison, where two images are shown, and a worker is asked to give a yes/no answer to state whether the two images refer to the same person. [3] examines the problem of allocating manpower to work on BQ-type HITs based on the workers' accuracy. Their solution, which assigns specific workers to each HIT, may not be applicable in existing systems. This is because in common crowdsourcing platforms like AMT, workers are allowed to freely choose the HITs they wish to do. Our algorithms only decide plurality values and do not force specific workers to perform specific tasks. Moreover, we study MCQs, which are generalized forms of BQs.

In [14], the authors study the problem of assigning pluralities to HITs that assign keyword labels to Internet resources (e.g., images). To determine pluralities, they propose a simple dynamic-programming-based algorithm. Although their solution share some similar properties as our `DP` algorithm, [14] did not further elaborate on the efficiency and effectiveness of their algorithm. We study some properties of MCQ (Section 3), and the relationship between HIT cost and quality (Section 4.5). We use these observations to develop solutions that are much faster than `DP`.

3. MULTIPLE-CHOICE QUESTIONS

We now describe our data and quality models (Sections 3.1 and 3.2). We focus on MCQ as an illustration of our solution framework. We then discuss two properties of MCQ in Section 3.3. We formally define the plurality assignment problem in Section 3.4. Although we focus on MCQ to simplify our discussion, our solution framework can be extended to cover other HIT types. We will briefly discuss other HIT types in Section 6.

3.1 Data model

The data model we employed here is based on the AMT system. In particular, a requester submits a set of one or more HITs to the crowdsourcing system, where each HIT contains a single MCQ.[3] For each MCQ, the requester specifies the question to be asked, as well as all their possible choices. The requester also associates an amount of reward to each MCQ to indicate the number of monetary units that will be paid to a worker for successfully finishing a HIT.

Notice that the number of MCQs contributed by a requester can be enormous; for instance, on 28 October 2012, the top-10 heaviest requesters contributed over 90,000 questions in total. To enable better HIT management, and to allow workers to choose HITs more easily, requesters may label the HITs and classify them based on themes and other properties.

Now, let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be n MCQs submitted by a requester. We use k_i, a non-negative integer, to denote the plurality of t_i (i.e., k_i workers are needed for t_i). Let c_i be the *cost* of t_i, i.e., the amount of reward given to a worker for completing t_i. Table 1 summaries the symbols used.

[3]In the sequel, when we mention MCQ, we refer to a HIT that contains a single MCQ.

Table 1: Symbols and their meanings.

Notation	Description
$\mathcal{T}$	A set of MCQs
B	Budget for assigning plurality to $\mathcal{T}$
n	Number of MCQs in $\mathcal{T}$
t_i	The i-th MCQ in $\mathcal{T}$
c_i	Cost of t_i (no. of units rewarded for finishing t_i)
k_i	Plurality of t_i
p_i	Accuracy of t_i
$\zeta_i(k)$	Quality function of t_i
$\mathcal{Q}_t(\mathcal{T}, \vec{k})$	Total quality of $\mathcal{T}$, with $\vec{k} = k_1, k_2, \ldots, k_n$
$\mathcal{Q}_a(\mathcal{T}, \vec{k})$	Average quality of $\mathcal{T}$

3.2 Quality Model

As discussed before, we determine the pluralities for a set of MCQs to optimize their overall *quality*. We now explain how to compute the quality of an MCQ t_i. Intuitively, this quantity captures the confidence of the answers given by workers on t_i; it depends on the answers' values, as well as the performances of workers on t_i. To model a worker's performance, the *worker's accuracy* model was proposed in [8]. We first describe this model and then explain how to use it to define an MCQ's quality.

Worker's accuracy [8]. Every HIT t_i is associated with a real value p_i called *worker's accuracy*, or *accuracy* in short. This is the probability that a randomly-chosen worker provides a correct answer for t_i. A way to estimate p_i is to collect information from other HITs whose true answers are known, and whose features are similar to those of p_i. For example, consider a set of HITs, each of which consists of a sentence; a worker is asked to express her sentiment about it. As discussed in [4], the readability of a sentence, and hence the worker's ability to comment it correctly, depends on its length. We can thus cluster these HITs according to the lengths of the sentences associated with them. Given a HIT t_i of a cluster, [8] presents a method to find p_i:

1. Some "sample HITs" are collected from the cluster, whose true answers are known.

2. Each worker is asked to do these HITs, after which her score is obtained (e.g., 90% of the questions are correctly answered).

3. The average score of these workers is taken as p_i.

As we will show in our experiments (Section 5), a small fraction of questions extracted from each cluster suffices to derive an accurate value of p_i.

It is worth noting that the information of "HIT clusters" may be provided by requesters themselves. For instance, requesters in AMT often classify HITs according to features like themes, categories, time, and locations, in order to assist a worker to choose HITs. On 28 October 2012, among the top-70 heaviest requesters, 45% of them provided more than one cluster of HITs, and 17% of them separated their HITs into four or more clusters. These clusters could be used to find p_i, by using the method described above.

MCQ quality. Now, let $\zeta_i(k)$, a real-valued function, be the *quality function* of t_i, which describes the "expected quality score" of t_i after it has been performed by k workers. The exact form of $\zeta_i(k)$ was recently proposed in [8], as

$$\zeta_i(k) = \begin{cases} 0, & k = 0, \\ 1, & k \text{ is odd}, p_i = 1, \\ \sum_{l=\lceil \frac{k}{2} \rceil}^{k} C_k^l p_i^l (1-p_i)^{k-l}, & k \text{ is odd}, p_i \neq 1. \end{cases} \quad (1)$$

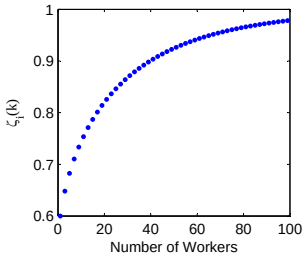


Figure 2: Monotonocity of $\zeta_i(k)$ (p_i = 0.6).

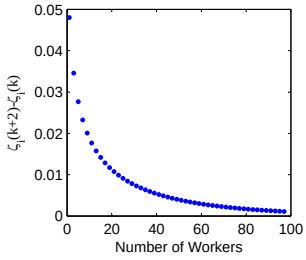


Figure 3: Diminishing Return of $\zeta_i(k)$ (p_i = 0.6).

C_k^l is the notation for the binomial coefficient $\binom{k}{l}$. Note that $\zeta_i(k)$ is not defined for any positive even number k. When k (the plurality) is zero, $\zeta_i(0)$ has the lowest value of zero, since no workers have worked on t_i. When $p_i = 1$, we have full confidence about the workers, and $\zeta_i(k)$ attains the highest value of one. For other cases, $\zeta_i(k)$ is given by:

$$\zeta_i(k) = \sum_{l=\lceil \frac{k}{2} \rceil}^{k} C_k^l p_i^l (1-p_i)^{k-l}. \tag{2}$$

Essentially, it is the probability that, out of the k workers answering t_i, $l \geq \lceil \frac{k}{2} \rceil$ of them give a correct answer. In other words, more than half of the workers provide the correct answer for this question [8]. We require k to be an odd number, so that an unambiguous decision can be made out of the k answers provided. This requirement is also adopted by previous works (e.g., [3, 8]). We remark that BQ is a special case of MCQ (where only two possible answers are provided), and so $\zeta_i(k)$ can also be applied to BQ.

Before we go on, we would like to discuss the effect of p_i on $\zeta_i(k)$. As we will elaborate in Section 3.3, $\zeta_i(k)$ increases *monotonically* with k if and only if $p_i \geq 0.5$. Intuitively, if it is more likely that a worker gives a correct answer than an incorrect one, then the quality of t_i increases monotonically with the number of answers obtained. In our experiments with real human workers, we observed that their accuracies were generally well above 0.5. We thus assume that $p_i \geq 0.5$ in our subsequent discussions.[4]

We next define the *total* and *average* qualities of $\mathcal{T}$.

Definition 1. Let $\vec{k} = (k_1, k_2, \ldots, k_n)$, where k_i is the plurality of t_i. The **total quality of** $\mathcal{T}$, or $\mathcal{Q}_t(\mathcal{T}, \vec{k})$, is:

$$\mathcal{Q}_t(\mathcal{T}, \vec{k}) = \sum_{i=1}^{n} \zeta_i(k_i). \tag{3}$$

Definition 2. The **average quality of** $\mathcal{T}$, or $\mathcal{Q}_a(\mathcal{T}, \vec{k})$, is:

$$\mathcal{Q}_a(\mathcal{T}, \vec{k}) = \frac{\mathcal{Q}_t(\mathcal{T}, \vec{k})}{n}. \tag{4}$$

We will explain later how these two quality measures can be optimized by assigning pluralities.

3.3 Monotonicity and Diminishing Return

We now discuss two interesting properties of MCQ's quality function. As we will show later, these properties are useful in developing plurality assignment algorithms.

[4]If $p_i < 0.5$, the plurality k_i should be set to zero, since $\zeta_i(k)$ does not improve with the number of workers.

(1) Monotonicity. We first observe that the MCQ quality, or $\zeta_i(k)$, increases with plurality k. An example is illustrated in Figure 2, which plots $\zeta_i(k)$ over different (odd) values of k, for $p_i = 0.6$. We see that when one worker is assigned to t_i, $\zeta_i(1) = 0.6$; with three workers, the quality increases to $\zeta_i(3) = 0.648$. Theorem 1 formalizes this phenonmenon.

THEOREM 1. *$\zeta_i(k)$ increases monotonically with k, if and only if $p_i \in [0.5, 1]$.*

The proof can be found in Appendix D. We also note that this theorem is consistent with the observation that the more workers are assigned to an MCQ, the more confident we are about its result. In fact, the AMT Best Practices Guide also suggests a HIT to be assigned higher plurality, in order to yield better results [1].

(2) Diminishing Return (DR). The second property of $\zeta_i(k)$ is that its rate of change drops with k. As an example, Figure 3 shows that the difference between $\zeta_i(k+2)$ and $\zeta_i(k)$ decreases with k, for $p = 0.6$. This concept, also known as *Diminishing Return* (DR) in economics literature [12], can be formalized by *marginal return*:

Definition 3. The **marginal return** of $\zeta_i(k)$, denoted by $\Delta\zeta_i(k)$, is:

$$\Delta\zeta_i(k) = \frac{\zeta_i(k+\Delta k) - \zeta_i(k)}{\Delta k} \tag{5}$$

where $\Delta k = 1$ if $k = 0$; otherwise, $\Delta k = 2$.

Essentially, $\Delta\zeta_i(k)$ describes the rate of change of $\zeta_i(k)$. Notice that $\Delta\zeta_i(k) \geq 0$, since its nominator is always non-negative due to Theorem 1. We now present Theorem 2, which describes the DR property of $\zeta_i(k)$.

THEOREM 2. *$\Delta\zeta_i(k)$ decreases monotonically with plurality k, if and only if $p_i \in [0.5, 1]$.*

To prove this theorem, we express $\Delta\zeta_i(k+2)$ and $\Delta\zeta_i(k)$ by using Equation 2. Then we show that $\Delta\zeta_i(k+2) \leq \Delta\zeta_i(k)$. The detailed proof can be found in Appendix E. Intuitively, the higher the value of k, the smaller is the change of $\zeta_i(k)$. These two properties can be useful to develop efficient algorithms to solve the plurality assignment problem.

3.4 The Plurality Assignment Problem (PAP)

Let B be the *budget* of a requester. That is, a maximum of B cost units is awarded to workers upon successful completion of the MCQs in $\mathcal{T}$. Our goal is to solve the **plurality assignment problem** (denoted by $\mathcal{P}(B, \mathcal{T})$), as defined below.

$$\textbf{given} \quad B, \mathcal{T} \text{ and } p_i, c_i \text{ for } t_i \in \mathcal{T} \tag{6}$$

$$\textbf{maximize} \quad \mathcal{Q}_t(\mathcal{T}, \vec{k}) \tag{7}$$

$$\textbf{subject to} \quad \sum_{i=1}^{n} k_i c_i \leq B \ \textbf{and} \tag{8}$$

$$k_i = 0 \text{ or a positive odd number} \tag{9}$$

Our aim is to find $\vec{k} = \{k_1, k_2, \ldots, k_n\}$ such that $\mathcal{Q}_t(\mathcal{T}, \vec{k})$ is maximized. Since n is constant, optimizing $\mathcal{Q}_t$ is the same as optimizing $\mathcal{Q}_a$ (c.f., Definitions 1 and 2). For simplicity, we use $\mathcal{Q}_t$ as our objective function (Equation 7).[5] Also notice that the above problem has two constraints. First, since each t_i is associated with a cost of $k_i c_i$, the total cost of all the MCQs in $\mathcal{T}$ must not exceed B (Equation 8). Second, k_i has to be a positive odd number or 0 (Equation 9) according to the domain of $\zeta_i(k)$.

[5]We use $\mathcal{Q}_a$ as an effective measure in our experiments.

4. PLURALITY ASSIGNMENT

In this section we present several plurality assignment algorithms. We first discuss two heuristics in Section 4.1. We then present an optimal solution and its fast version in Sections 4.2 and 4.3, respectively. We propose a greedy algorithm in Section 4.4, and discuss how to enhance its performance in Section 4.5.

4.1 The Even and Random Heuristics

Consider a requester with a budget B and n MCQs. Without any information about the workers' accuracies, how would the requester assign her budget? Here we consider two heuristic plurality assignment methods which might be adopted by an "uninformed" requester.

1. `Random`: *We arbitrarily pick an MCQ to increase its plurality and repeat until the budget is exhausted.* Specifically, let b be the currently available budget. Initially, b is equal to B, and all k_i's are equal to zero. We execute two steps: (Step 1) We randomly pick an MCQ t_j such that b is large enough to increase t_j's plurality to the next odd number. (Step 2) We decrease b by c_j (if $k_j = 1$) or $2c_j$ (if $k_j > 1$). These two steps are repeated until no MCQ can be picked in Step 1.

2. `Even`: *We divide the budget evenly across all the MCQs.* Specifically, let b be the amount of budget allocated for each MCQ, we have $b = B/n$. Since the cost consumed for each MCQ should not exceed b, i.e., $k_i c_i \leq b$, we set k_i to be the largest odd number that does not exceed $\lfloor \frac{b}{c_i} \rfloor$ if $b \geq c_i$; otherwise, $k_i = 0$.

These heuristics, which ignore the worker's performance information (p_i's), may not yield an optimal plurality assignment. For example, if p_i is high, then just a few workers are enough to yield a high quality for MCQ t_i. However, these heuristics may still assign an excessively large k_i. We thus treat these two heuristics as our baseline algorithms. We next investigate better solutions.

4.2 A Dynamic Programming Algorithm

Let us now present an optimal solution, called `DP`, to solve PAP. We observe that problem $\mathcal{P}(B, \mathcal{T})$ exhibits *optimal substructure*, which enables dynamic programming. The detailed proof of this property can be found in Appendix A. Now, let $\mathcal{T}_l$ be a subset of $\mathcal{T}$, which contains the MCQs with l smallest indices, i.e., $\mathcal{T}_l = \{t_1, \ldots, t_l\}$. Let $Q(B, n)$ be the optimal total quality of $\mathcal{T}$ for $\mathcal{P}(B, \mathcal{T})$, and $Q(b, l)(0 \leq b \leq B, 0 \leq l \leq n)$ be the optimal total quality of $\mathcal{T}_l$ for the subproblem $\mathcal{P}(b, \mathcal{T}_l)$. Since the objective function of $\mathcal{P}(B, \mathcal{T})$ is $\sum_{i=1}^{n} \zeta_i(k_i)$ (Equation 3), we can compute $Q(B, n)$ recursively:

$$Q(b,l) = \begin{cases} 0, & l = 0, \\ \max\{Q(b - k_l c_l, l-1) + \zeta_l(k_l) | \\ \quad ((k_l \text{ is odd}) \wedge (k_l \leq \left\lfloor \frac{b}{c_l} \right\rfloor)) \vee (k_l = 0)\}, & l > 0. \end{cases} \tag{10}$$

To compute $Q(b, l)$, we enumerate all possible values of k_l and find the optimal one. Let k_l^* be the value of k_l that yields the maximal value of $Q(b, l)$. Then, k_l^* is also the optimal plurality for t_l, which is guaranteed by the optimal substructure of $P(b, \mathcal{T}_l)$. The details of DP can be found in Appendix B.

Discussions. Let $M = \min_{i=1,\ldots,n}\{c_i\}$. Let $F(t_i, k_i)$ be the time for computing $\zeta_i(k_i)$. Then, the complexities of F and `DP` are $O(\frac{B}{M})$ and $O(\frac{nB^3}{M^2})$ respectively. Since $\zeta_i(k)$ can be evaluated frequently ($\frac{nB^2}{M}$ times), if any of n, B, or F is large, `DP` will be very slow. In our experiments, it takes a few hours to compute the plurality of a set of 60,000 HITs! An efficient plurality assignment algorithm is therefore essential for large crowdsourcing systems that manage numerous requesters, or when the plurality assignment has to be recomputed frequently to reflect any changes in worker statistics and HITs. Next, we study how to reduce the time needed to compute $\zeta_i(k)$.

4.3 An Improved `DP` Algorithm

We discuss an alternative way to compute an MCQ's quality function, $\zeta_i(k)$. This result can be used to develop a faster version of `DP`, which we call `DP-inc`. The mathematical derivations of the equations shown in this section can be found in Appendix C.

Our main idea is to express $\zeta_i(k)$ in terms of $\zeta_i(k')$ values, where k' is some number smaller than k. Particularly, let $d_i(k)$ be the difference between $\zeta_i(k'')$ and $\zeta_i(k)$, where k'' is an odd number immediately larger than k. In other words,

$$d_i(k) = \begin{cases} \zeta_i(1) - \zeta_i(0), & k = 0, \\ \zeta_i(k+2) - \zeta_i(k), & k \text{ is odd.} \end{cases} \tag{11}$$

We can then rewrite $\zeta_i(k)$ in terms of $d_i(k')$:

$$\zeta_i(k) = d_i(0) + \sum_{(k'<k) \wedge (k' \text{ is odd})} d_i(k'). \tag{12}$$

Further, we have:

$$d_i(k) = \begin{cases} p_i, & k = 0, \\ d_i(k-2) \cdot 4p_i(1-p_i)\frac{k}{k+1}, & k \text{ is odd.} \end{cases} \tag{13}$$

For $k > 1$, given the value of $d_i(k-2)$, we can obtain $d_i(k)$ by Equation 13, in constant times. We can then "incrementally" obtain $\zeta_i(k+2)$, since $\zeta_i(k+2) = \zeta_i(k) + d_i(k)$ by Equation 11.

In `DP`, the possible plurality values for an MCQ, represented by k_l in Equation 10, are enumerated in ascending order. Hence, we can use the solution above to compute $\zeta_i(k)$ based on the $d_i(k')$ values. The corresponding complexity of computing $\zeta_i(k)$, or F, drops from $O(\frac{B}{M})$ to $O(1)$, and the running time of `DP` is improved to $O(\frac{nB^2}{M})$. We name this modified version of `DP` as `DP-inc`.

4.4 A Greedy Algorithm

Our `Greedy` algorithm adopts the framework of the greedy algorithm in [13], which was developed to solve the 0-1 knapsack problem. In each iteration, we select the "best" MCQ according to some measures, and increase its plurality accordingly. These steps are repeated, until the budget B is exhausted. The criterion that we use to choose the MCQ is the *marginal gain*, as defined below:

Definition 4. The **marginal gain** of t_i, denoted by $h_i(k)$, is:

$$h_i(k) = \frac{\Delta\zeta_i(k)}{c_i}. \tag{14}$$

In essence, $h_i(k)$ is the marginal return (Definition 3) per unit cost after k workers have been assigned to t_i.

Details. The implementation of `Greedy` is shown in Algorithm 4. We let b be the currently available budget, which is initially equal to B (Line 3). We use a priority queue structure, Q, to store the IDs of MCQs. An MCQ with a larger marginal gain has a higher priority in Q. We also initialize the plurality of every MCQ to zero (Lines 4 to 6).

Next, in every iteration, we select an MCQ with the largest marginal gain, and see whether we can raise its plurality (Lines 7 to 14). Specifically, the MCQ t_i with the largest $h_i(k_i)$ in Q is popped (Line 8). If the budget b is sufficient for $h_i(k_i)$ to be increased by Δk, we update the values of k_i and b (Line 12). We then recompute the new $h_i(k_i)$ value, and push t_i to Q (Lines 13 to 14). This process is repeated until (1) the budget is exhausted or (2) Q is empty (Line 7). In Line 15, `Greedy` returns $\vec{k}$ as the final answer.

Algorithm 1 `Greedy`

1: **Input:** B, $\mathcal{T}$ and p_i, c_i for $t_i \in \mathcal{T}$
2: **Output:** $\vec{k}$
3: $b \leftarrow B$
4: **for** $i \leftarrow 1$ to n **do**
5: $k_i \leftarrow 0$, calculate $h_i(k_i)$ by Equation 14
6: Q.push(t_i) // Q is a priority queue.
7: **while** $b > 0$ and Q is nonempty **do**
8: $t_i \leftarrow Q$.pop() // The MCQ with the largest marginal gain, i.e., $h_i(k_i)$, is popped.
9: **if** $k_i = 0$ **then** $\Delta k \leftarrow 1$
10: **else** $\Delta k \leftarrow 2$
11: **if** $c_i \Delta k \leq b$ **then** // There is enough budget.
12: $k_i \leftarrow k_i + \Delta k$, $b \leftarrow b - c_i \Delta k$
13: calculate $h_i(k_i)$ by Equation 14
14: Q.push(t_i)
15: **return** $\vec{k}$

Complexity. Notice that $h_i(k_i)$ is a function of $d_i(k_i)$, and $d_i(k)$ can be incrementally computed in $O(1)$ times (using Equation 13). In `Greedy`, we compute $h_i(k_i)$ in ascending order of k_i; therefore, the time complexity of computing each $h_i(k_i)$ is $O(1)$. Updating Q costs $O(\log n)$ time, and the maximum value of k_i cannot exceed B/M. Hence, the time and space complexities for `Greedy` are respectively $O((\frac{B}{M} + n) \log n)$ and $O(n)$.

Accuracy. We next sketch the proof that `Greedy` is a 0.5-approximation algorithm. First, we show that our plurality optimization problem $\mathcal{P}$ is just a variant of the 0-1 knapsack problem $\mathcal{P}'$ [13]. (Specifically, $\mathcal{P}$ is equivalent to $\mathcal{P}'$ with some additional constraint.) Second, by using the monotonicity and DR properties of $\zeta_i(k)$, we prove that the optimal solutions of $\mathcal{P}$ and $\mathcal{P}'$ have the same objective value. Third, we consider `KPGreedy`, which solves $\mathcal{P}'$ with a worst-case effectiveness ratio of 0.5 [13]. We show that `Greedy` is the same as `KPGreedy`, and is thus a 0.5-approximation algorithm. For more details, please consult Appendix F.

In practice, `KPGreedy` has a close-to-optimal effectiveness [17]. This can also be said for `Greedy`. In fact, our experiments also show that the accuracy of `Greedy` is close to that of `DP`.

4.5 Enhancing Performance by Groups

For the algorithms presented so far (i.e., `DP`, `DP-inc`, and `Greedy`), there is one thing in common: the plurality k_i of each MCQ t_i is determined *independently*. Computing k_i involves a cost; for `DP` and `DP-inc`, their running times can be large when the budget B is big. If n, the number of MCQs under consideration, is large too, then the execution times of our algorithms could be substantial. Unfortunately, B and n can be huge in reality: the MCQs submitted by a requester in AMT is often in thousands. For example, we extracted over 60,000 MCQs from AMT for our experiments. Even with a small budget for these MCQs, say B = 60,000 (i.e., each HIT has an average plurality of one), `DP-inc` does not terminate within ten hours! Hence, it is important to improve the performance of our solutions in order to handle large HIT sets.

Our idea for enhancing the algorithms is not complicated. Also, the algorithms' effectiveness is not affected at all. To illustrate, consider two MCQs, t_i and t_j, which have the same cost and worker accuracy. If budget allows, they *must* be allocated the *same* plurality by `DP`, `DP-inc`, and `Greedy`. This follows from the fact that these algorithms only consider cost and accuracy of MCQs, and so they cannot treat t_i and t_j differently. If we have determined the plurality k_i, then k_j can be immediately deduced to be equal to k_i. This can be faster than computing k_i and k_j independently.

Based on this intuition, we first partition the set of MCQs into *groups*; the MCQs that belong to each group possess the same cost and accuracy. For each group, we select a "representative MCQ" and evaluate its plurality. Given a sufficient budget size, this plurality is assigned to other MCQs in the same group. This is faster than computing the plurality of every MCQ in the group individually.

Due to the limitation of space, we do not show all the details of how group information is incorporated in our algorithms. An implementation issue that we want to discuss is that the given budget may not be enough to enable all the MCQs in the same group to be assigned the same plurality. Let us consider Line 12 of `Greedy` (Algorithm 4), which increases k_i by Δk if the remaining budget, b, is sufficient (Line 11). In the "group-based version" of `Greedy`, we treat t_i as the representative of a group g. In Line 11, we calculate the maximum number m of MCQs that can have their plurality increased by Δk with budget b. In Line 12, we arbitrarily select m MCQs from g, and increase their plurality values accordingly. Then g is disregarded, since we can no longer increase the plurality of the MCQs in g. More details about how we consider group information in our algorithms can be found in Appendix G.

We also remark that the MCQ set used in our experiments consists of only a few "big" groups. That is to say, each group contains a large number of HITs. Consequently, the algorithms that make use of group information significantly outperform those that handle MCQs individually. Next we explain these results in more detail.

5. RESULTS

We now present the experiment results of `DP`, `DP-inc`, `Greedy`, and the baseline heuristics, `Even` and `Random`. Results on synthetic data are reported in Section 5.1. We also examine our solutions on the data collected from our system prototype (Section 5.2).

In this section, `DP`, `DP-inc`, and `Greedy` refer to their variants that consider the grouping of MCQs discussed in Section 4.5. We use `DP-NG`, `DP-inc-NG`, and `Greedy-NG` to denote the versions of algorithms that manage MCQs independently (i.e., no grouping). In the sequel, we focus on `DP`, `DP-inc`, and `Greedy`.[6] Each data point is the average result of 100 runs. All our experiments are implemented with C++ on a 64-bit Ubuntu system with 8G memory and an Intel i5 processor.

5.1 Results on Synthetic Data

(a) Experiment setup. We use the MCQs provided by an AMT requester called "CrowdSource" on 28 October 2012 as the basis of the synthetic dataset. This requester submits a total of $67,075$ MCQs. These MCQs have been classified into N groups by the requester, where $N = 12$. Recall that every MCQ in the same group has the same cost and worker accuracy. For convenience, we label each group as g_j, where $j = 1, \ldots, N$. Each g_j has m_j MCQs. For each MCQ in g_j, its cost and accuracy are α_j and ρ_j respectively. In these experiments, ρ_j is randomly selected from the range $[0.5, 1]$. We use the average quality, $\mathcal{Q}_a(\mathcal{T}, \vec{k})$ (Equation 2), to measure the effectiveness of our algorithms. The costs of MCQs vary from \$0.08 to \$0.24, with an average of \$0.12. The requester's budget is $\$20K$ by default. If plurality is not considered, this budget allows each MCQ to be worked by an average of 3 workers. Table 2 shows the detailed setup of our experiments.

(b) Effect of budget B. Figure 4(a) shows the effectiveness of `DP-inc`, `Greedy`, `Even` and `Random` under different values of budget B. Since `DP-inc`, a faster version of `DP`, has the same effectiveness as `DP`, we do not show the results of `DP` here. Also,

[6]The use of groups does not change the effectiveness of the algorithms being considered. Moreover, they can be much faster than if groups are not used. Hence, we use the "group-based algorithms" as our default.

Table 2: Summary of groups g_j (m_j and α_j are true values extracted from AMT).

j	1	2	3	4	5	6	7	8	9	10	11	12
m_j	14206	7153	8977	8970	7577	7568	3024	3024	2610	2605	688	673
α_j in \$	0.08	0.08	0.15	0.15	0.12	0.12	0.12	0.12	0.24	0.24	0.15	0.15
ρ_j	Uniformly distributed in $[0.5, 1]$											

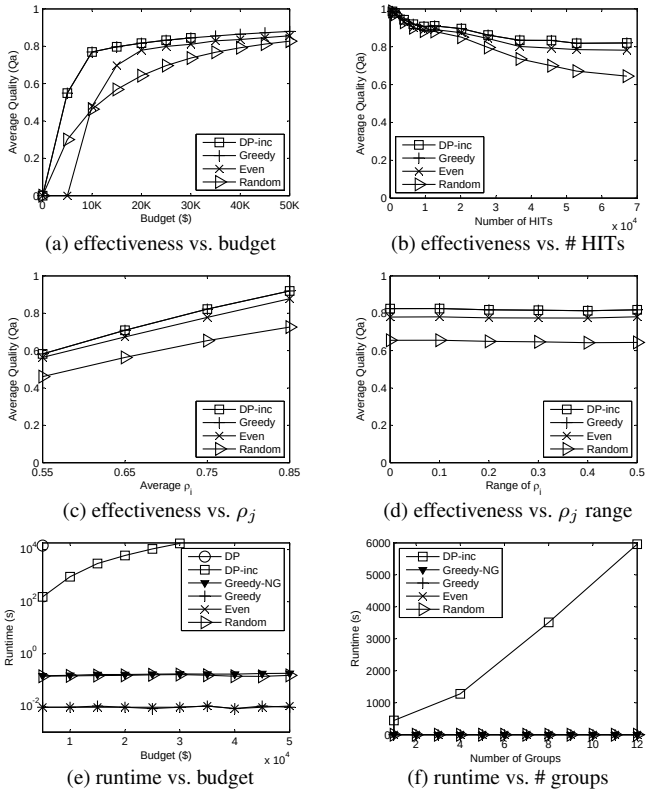
as shown later in our performance experiments, `DP-inc` runs very slowly under a large budget, and so we only present its effectiveness for $B \leq \$30K$. For all the algorithms shown, $\mathcal{Q}_a$ increases with B. This means that in general a larger budget gives better quality result. However, the increase rate of $\mathcal{Q}_a$ drops with B; increasing its value constantly does not give a significant improvement of $\mathcal{Q}_a$. Hence, it is not necessary to use a very large value of B to obtain high-quality results.

We also see that `DP-inc` attains the highest effectiveness, and `Greedy` comes very close to it. On average, the difference between these two algorithms is 1% or less. At $B = \$20K$, `DP-inc` and `Greedy` both have $\mathcal{Q}_a = 81\%$, which is 5% higher than that of `Even` (77%) and 20% higher than that of `Random` (64%). Under a smaller budget (say, $\$10K$), the gap is even wider: the effectiveness of `DP-inc` and `Greedy` are about 76%, while that of `Even` and `Random` is both less than 50%. This difference is due to the fact that neither `Even` nor `Random` considers accuracy information in setting plurality values. As a result, MCQs with lower accuracy may not receive enough number of answers, while the highly accurate MCQs may be associated with unnecessarily high plurality values. Since `DP-inc` considers group accuracy information and produces an optimal solution, its effectiveness is the best. The effectiveness of `Greedy` is almost as good as `DP-inc`. Also, although `Even` performs better than `Random` in most of cases, it performs the worst under a small budget (of $10K$ or less). In `Even`, expensive MCQs may have a zero plurality, rendering poor effectiveness. Another point is that both `DP-inc` and `Greedy` need a budget of $\$20K$ to achieve $\mathcal{Q}_a = 80\%$, while `Even` and `Random` will respectively use more than $\$30K$ and $\$45K$ to achieve the same effectiveness. Hence, if we want to attain a specified value of $\mathcal{Q}_a$, `DP-inc` and `Greedy` only needs about 67% of the cost of `Even` and 44% of the cost of `Random`.

(c) Effect of number of HITs n. Figure 4(b) examines the effect of the number of HITs (n), obtained by varying the number of groups. We observe that the effectiveness of all the algorithms drop with the increase of n. Under a fixed budget, with more MCQs, the plurality assigned to each MCQ tends to be smaller. Hence, $\mathcal{Q}_a$ drops. Another observation is that the quality drop rates of `DP-inc` and `Greedy` are both slower than those of `Even` and `Random`. When the number of MCQs is small, all the algorithms concerned have few choices, and so their effectiveness values are similar. When the number of MCQs increases, `DP-inc` and `Greedy` allocate plurality to MCQs with higher accuracy and lower cost, and so they perform better than both `Even` and `Random`.

(d) Effect of worker's accuracy ρ_j. Figures 4(c) and (d) study the impact of accuracy of an MCQ of each group (i.e., ρ_j) on the algorithms' effectiveness. For Figure 4(c), ρ_j is uniformly distributed between $[u - 0.05, u + 0.05]$, and we vary the value of u. Observe that with a higher MCQ accuracy, the effectiveness of all the algorithms increases; in other words, better MCQ results can be obtained. On average, `DP-inc` and `Greedy` are 5% and 18% more effective than `Even` and `Random`, respectively.

Figure 4(d) examines the effect of the range of ρ_j on $\mathcal{Q}_a$. Here, the values of ρ_j are uniformly distributed between $[0.75 - \frac{r}{2}, 0.75 + \frac{r}{2}]$. When r increases, the difference in ρ_j values among different groups increases. Notice that the effectiveness of all the algorithms



(a) effectiveness vs. budget (b) effectiveness vs. # HITs

(c) effectiveness vs. ρ_j (d) effectiveness vs. ρ_j range

(e) runtime vs. budget (f) runtime vs. # groups

Figure 4: Result on synthetic data.

is stable over a wide range of r. Also, `DP-inc` and `Greedy` are 5% and 18% better than `Even` and `Random` respectively.

(e) Performance. Figure 4(e) illustrates the performance of the plurality-setting algorithms under different budget size B. Since `DP` and `DP-inc` are very slow, we only show their results for small values of B. We see that `DP-inc` is about 100 times faster than `DP`. This is because `DP-inc` computes the MCQ quality function more efficiently than `DP`. Unfortunately, `DP-inc` is still very slow compared to heuristics (i.e., `Random` and `Even`) and our approximate solution (i.e., `Greedy`). At $B = 20K$, for example, `DP-inc`, which completes in 6000s, is 1,000 times slower than `Random`, which finishes in less than 1s, and is four orders of magnitude poorer than `Even` and `Greedy`, whose running times are less than 10ms. Moreover, while the execution time of `DP-inc` increases sharply with B, the performance of other solutions is relatively stable. We conclude that the performance of `DP` and `DP-inc` is significantly inferior to other solutions.

Grouping. On the same graph, we display `Greedy-NG`, which does not use any grouping information. We see that it is 20 times slower than `Greedy`. Recall that the MCQs tested here are partitioned into 12 groups (Table 2). Also, `Greedy-NG` has to compute the plurality independently for every MCQ. Since the number of MCQs is tremendous ($67K$), its performance is much worse than `Greedy`, which evaluates the plurality for only 12 group represen-

tatives (and assigns these values to other MCQs). We have also tested DP-NG and DP-inc-NG. Since they are extremely inefficient, we do not show all their results here. Under a mild budget of $5K$ (i.e., each MCQ has an average plurality of one), they cannot finish in 10 hours. By using group information, DP and DP-inc complete in 4 hours and 10 minutes respectively. Hence, the use of grouping information significantly improves the performance of DP, DP-inc, and Greedy.

Figure 4(f) shows the performance of the plurality-setting algorithms under different number of groups N. Similar to (e), DP-inc performs much worse than other solutions. For example, at $N = 8$, DP-inc needs 3,500 seconds to complete, while the finishing time of Greedy is only 10ms. Moreover, while the execution time of DP-inc increases sharply with N, the completion time of other solutions increases much more slowly with N. We skip the results for DP-NG and DP-inc-NG here, since they are extremely slow.

(f) Summary. By assigning plurality to MCQs based on their accuracy and cost information, DP-inc and Greedy achieve higher effectiveness than simple heuristics (e.g., Even and Random). The effectiveness of DP-inc and Greedy is stable over a wide range of worker's accuracy values. The use of grouping techniques significantly improves the performance of these algorithms without sacrificing effectiveness. The best algorithm is the Greedy; while its effectiveness is close to optimal, it is efficient even under large values of B and n.

5.2 Results on Real Data

(a) Experiment setup. We have created 100 MCQs for collecting the sentiment information of a comment. These 100 comments are extracted from ten popular Youtube videos. For each MCQ, a worker is asked to select an answer from {*positive*, *neutral*, *negative*}. As discussed in [4], a longer comment is usually harder to interpret. We thus separate these MCQs into three groups of different lengths. We also associate more incentives to groups with longer comments, as shown in Table 3. We have implemented a web application for workers to perform these MCQs (Figure 5). The worker can view the comments and submit her choice. Help messages, displayed at the bottom of the screen, assist the worker to use the system. We have employed 25 graduate students, whose second language is English, to work on these MCQs.

We compute the worker's accuracy values based on their input, based on the method described in Section 3.2. First, we manually check each comment to generate our ground truth. Then, we randomly sample $s = 10\%$ MCQs from each group, and ask all our workers to do these selected questions. (For convenience, we call s the *sampling rate*.) By comparing their answers with the ground truth, we can estimate their performance of doing the MCQs in each group. Specifically, let S be the set of sampled MCQs, and W be the set of workers. For each worker $w \in W$, let $A(w)$ be the set of answers provided by w for questions in S, and a be an answer of $A(w)$. We then compute the *effectiveness* of each worker $p(w)$ for questions in S, where

$$p(w) = \frac{\|\{a \text{ is correct} | a \in A(w)\}\|}{\|A(w)\|}. \quad (15)$$

The accuracy of an MCQ for each group is estimated as the average of all $p(w)$ values, that is, $\frac{\sum_{w \in S} p(w)}{\|W\|}$. Table 3 shows the accuracy values. We can see that the longer the comments, the lower is the accuracy.

In the sequel, we present our results on the effectiveness of the plurality assignment algorithms, as well as the actual amount of time needed by workers to answer the MCQs. We have also examined the performance of our algorithms. Their relative performance

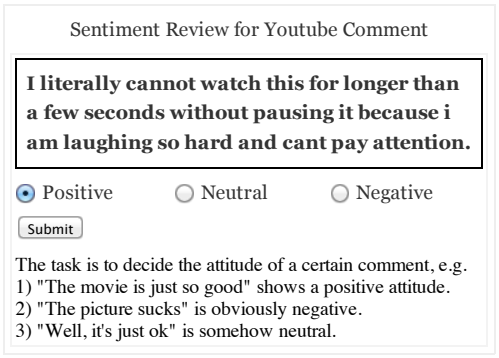


Figure 5: Sentiment collection: screenshot.

Table 3: A real dataset.

group	g_1	g_2	g_3
# words in comments	≤ 10	$(10, 20]$	> 20
# questions	36	36	28
cost ($)	0.1	0.2	0.3
MCQ accuracy (%)	83.4	70.1	61.3
avg. completion time (s)	6.15	8.87	10.33

is similar to that of synthetic data. Due to space constraints, we do not report their results here.

(b) Effect of budget B. Figure 6 (a) shows the average quality, $\mathcal{Q}_a$, over different values of B. We can see that the trend of the curves is similar to those obtained for synthetic data in Figure 6(a), where DP-inc and Greedy are always better than Even and Random. At $B = \$20$, DP-inc and Greedy are 26.3% and 42.1% more effective than Even and Random, respectively. Moreover, to achieve $\mathcal{Q}_a = 80\%$, DP-inc and Greedy require a budget of \$40, which is 30% and 55% smaller compared to the budget needed by Even and Random, respectively.

(c) Effect of sampling rate s. Figures 6(b) shows the effect of sampling rate on $\mathcal{Q}_a$. Here, each line represents the effectiveness of Greedy based on a fraction s of the MCQs. Observe that s does not have significant influence on $\mathcal{Q}_a$. For example, at $B = \$30$, the quality difference between the two cases $\{s = 10\%, s = 100\%\}$ is only 0.49%. This shows that a low sampling rate (e.g., 10%) is sufficient to yield high effectiveness.

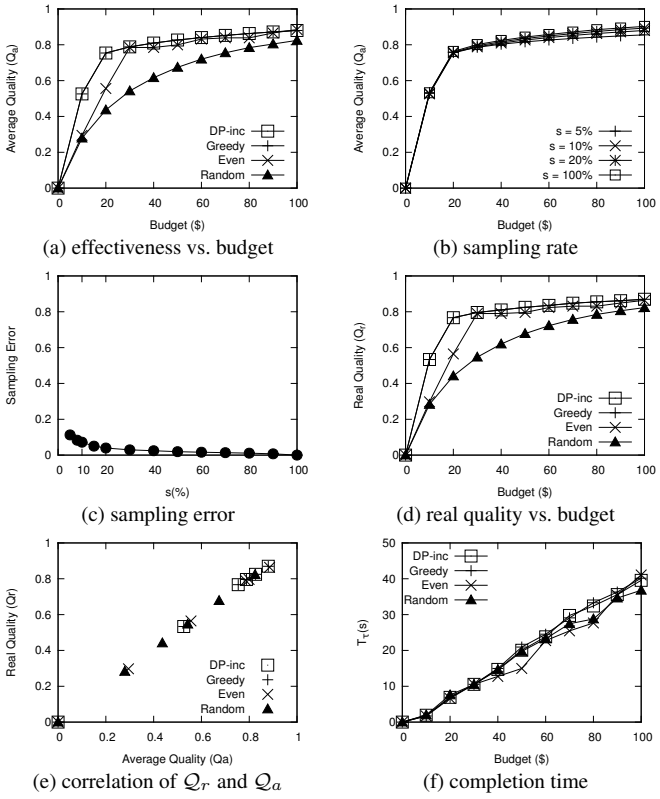
Let us further investigate why the values of s tested do not have a big impact at $\mathcal{Q}_a$. We first compute the estimated accuracy values $\vec{\rho} = \{\rho_1^s, \rho_2^s, \ldots, \rho_N^s\}$ for $\mathcal{G} = \{g_1, g_2, \ldots, g_N\}$, where ρ_i^s denotes the sampling rate used for estimating ρ_i.

We then define the *sampling error* as:

$$\texttt{err}(\vec{\rho}) = \frac{\sum_{j=1}^{N} |\rho_j^s - \rho_j^{100}|}{N} \quad (16)$$

Figure 6(c) shows $\texttt{err}(\vec{\rho})$ over different values of s. We can see that $\texttt{err}(\vec{\rho})$ is quite stable (less than 4%) when $s \geq 20\%$. Hence, the effectiveness of Greedy is not very sensitive to s.

(d) Real quality. In this experiment, we analyze the answers obtained from the workers, after the number of answers provided for each MCQ has reached the assigned plurality. We would like to compute the goodness of these results, which we call *real quality*, and compare it with $\mathcal{Q}_a$. Specifically, for every MCQ t_i with plurality k_i, we obtain a set $A(t_i)$ of k_i worker's answers for t_i. Then, we use half-voting to decide the result of this MCQ. Let the half-voting result on $A(t_i)$ be $V(t_i)$. We define the real quality, or



(a) effectiveness vs. budget (b) sampling rate

(c) sampling error (d) real quality vs. budget

(e) correlation of $\mathcal{Q}_r$ and $\mathcal{Q}_a$ (f) completion time

Figure 6: Result on real data.

$\mathcal{Q}_r$, as:

$$\mathcal{Q}_r(\mathcal{T}, \vec{k}) = \frac{\|\{V(t_i) \texttt{ is correct} | t_i \in \mathcal{T}\}\|}{n}.$$

Essentially, $\mathcal{Q}_r$ is the fraction of the MCQs, whose half-voting result is same as the ground truth.

Figure 6(d) shows the effect of budget on the real quality. Notice that the graph is highly similar to Figure 6(a) (which plots $\mathcal{Q}_a$ against B). This indicates that $\mathcal{Q}_a$ reflects the number of MCQs correctly answered. To study the relationship between $\mathcal{Q}_a$ and $\mathcal{Q}_r$, we plot Figure 6(e), where each point (x_i, y_i) corresponds to a plurality assignment $\vec{k}$ based on some budget and plurality assignment algorithm. Specifically, $x_i = \mathcal{Q}_a(\mathcal{T}, \vec{k})$ and $y_i = \mathcal{Q}_r(\mathcal{T}, \vec{k})$. We observe a strong correlation between $\mathcal{Q}_a$ and $\mathcal{Q}_r$. We further compute the correlation [5] of the two sets of $\{x_i\}$ and $\{y_i\}$ values, and find that the correlation is over 99.8%. Hence, in this experiment, our average quality measure is a good indicator of the correctness of workers' results.

(e) Completion time. We next analyze the *average completion time* of workers to work on an MCQ. Specifically, we record the time a particular worker uses to give an answer to MCQ t_i. This is defined as the *completion time* of this worker on t_i. Table 3 summarizes the average completion time for MCQs in each group. We observe that for a longer comment, workers may consume a longer time to read it and decide its sentiment. Similar to the experiment on real quality, for every MCQ t_i with plurality k_i, we obtain k_i workers, and also their corresponding completion time. We then sum them up to get the total completion time for all these k_i workers on t_i, denoted by T_i. The average completion time of $\mathcal{T}$, denoted by $T_\mathcal{T}$, is

$$T_\mathcal{T} = \frac{1}{n} \sum_{i=1}^{n} T_i. \tag{17}$$

Figure 6(f) shows the effect of budget on $T_\mathcal{T}$. We observe that $T_\mathcal{T}$ of all algorithms increases with budget since more plurality is allocated to each MCQ as budget increases. Besides, $T_\mathcal{T}$ for `DP-inc` (21.52s) and `Greedy` (21.89s) are both about 5% higher than that of `Random` (20.5s), and 9% higher than that of `Even` (19.88s). This is because in this real dataset, MCQs with lower accuracy require more completion time, and both `DP-inc` and `Greedy` allocate higher plurality to MCQs with lower accuracy. Consequently, the average completion time of `DP-inc` and `Greedy` are slightly higher than others. This increase in completion time, however, is justified, because the answers produced by `DP-inc` and `Greedy` have a higher quality than those of `Even` and `Random`.

6. OTHER HIT TYPES

We have focused our discussion so far on MCQ, which is the most popular HIT type on AMT, as an illustration of our solution framework. We remark that other kinds of HITs can also be similarly handled. In this section we further illustrate our framework by briefly discussing how plurality assignment can be done on two other kinds of HITs, namely, *Enumeration Query* (EQ) and *Tagging Query* (TQ).

First, we note that the key component of our framework (see Figure 1) is the quality manager, which consists of a *quality estimator* and a *plurality assignment algorithm*. Basically, the job of the quality estimator is to derive the quality function $\zeta_i(k)$ of a HIT t_i after the HIT has received k answers. For MCQ, we model $\zeta_i(k)$ by Equation 1. Note that this model has one parameter p_i. The job of the quality estimator is thus to estimate the model parameter p_i, which is done through a sampling process (Section 3.2). Given the quality functions $\zeta_i(k)$ of all the HITs as input, the plurality assignment algorithm outputs an assignment. The choice of the assignment algorithm depends on the properties of the quality function — `DP` is applicable for all quality functions, while `Greedy` is applicable as long as the quality function observes monotonicity and diminishing return. So, for any other types of HITs, to apply our plurality assignment framework, we only need (1) a model of the quality function $\zeta_i(k)$ and a way to estimate the model's parameters (such as p_i for MCQ) and (2) analyze the quality function and see if it satisfies the two conditions required by `Greedy` (and if so, we apply `Greedy`; otherwise, we apply `DP`). Let us illustrate this process using two other HIT types, EQ and TQ.

The objective of an EQ is to obtain the complete set of distinct elements for a set query. (For example, "*Name a state in the US.*" with the purpose of getting all the states' names.) From [18], we can model the quality function $\zeta_i(k)$ by $\hat{f}_0 \left[1 - \left(1 - \frac{1-\hat{C}}{\hat{f}_0}\right)^{k-k_0}\right]$. The variables $\hat{f}_0$ and $\hat{C}$ are two model parameters that captures the information of the total number of distinct answers, and the percentage of distinct answers collected so far, respectively, and k_0 is the number of answers given by workers so far. This quality function estimates the number of distinct answers obtained by k workers. In [18], it is shown how the model parameters can be estimated by statistical methods. It is easy to verify that the above quality function satisfies monotonicity and diminishing return. Hence, PAP on EQ can be efficiently solved by our `Greedy` algorithm.

As another example, we consider *Tagging Query* (TQ). The objective of a TQ is to obtain keywords (or tags) that best describe an object (such as an image or a web page). TQ has been recently studied in [21], which proposes a quality metric for measuring the quality of tags collected through workers' answers. Although not yet proven, the empirical results presented in [21] indicates that the quality metric proposed exhibits a power-law relationship with the

number of workers' answers. We can thus model our quality function $\zeta_i(k)$ for TQ using some power-law models. Since power-law models satisfy monotonicity and diminishing return, `Greedy` is again applicable to TQ. As a future work, we are studying how to effectively estimate the power-law model parameters for TQ.

7. CONCLUSIONS

We study the problem of setting plurality for HITs in crowdsourcing environments. We develop a solution that enables an optimal assignment of plurality for MCQs under a limited budget. We show that the quality of MCQs demonstrate monotonicity and diminishing return. We use these properties to develop effective and efficient plurality algorithms. We have performed extensive experiments on real and synthetic data. We conclude that the greedy algorithm is highly effective and efficient. We briefly discuss how our framework can be extended to support other kinds of HITs (i.e., EQ and TQ). We plan to study these extensions in more detail.

Acknowledgements

This research is partly supported by Hong Kong Research Grants Council grant HKU711309E and HKU712712E. We would like to thank the anonymous reviewers for their insightful comments.

8. REFERENCES

[1] Amazon Web Services LLC. Amazon Mechanical Turk: Best Practices Guide. `http://mturkpublic.s3.amazonaws.com/docs/MTURK_BP.pdf`.
[2] D. W. Barowy et al. Automan: A platform for integrating human-based and digital computation. *OOPSLA*, 2012.
[3] C. Cao and J. She et al. Whom to ask? Jury selection for decision making tasks on micro-blog services. *VLDB*, 2012.
[4] E. Dale and J. S. Chall. A formula for predicting readability. *Educational research bulletin*, 1948.
[5] S. Dowdy. *Statistics for Research*. Wiley, 1983.
[6] M. J. Franklin et al. CrowdDB: answering queries with crowdsourcing. In *SIGMOD*, 2011.
[7] S. Guo and A. Parameswaran et al. So who won? dynamic max discovery with the crowd. *SIGMOD*, 2012.
[8] X. Liu, M. Lu, B. Ooi, Y. Shen, S. Wu, and M. Zhang. CDAS: A crowdsourcing data analytics system. *VLDB*, 2012.
[9] A. Marcus, E. Wu, D. Karger, S. Madden, and R. Miller. Human-powered sorts and joins. *VLDB*, 2011.
[10] A. Marcus, E. Wu, S. Madden, and R. Miller. Crowdsourced databases: Query processing with people. *CIDR*, 2011.
[11] A. Marcus and E. Wu et al. Demonstration of qurk: A query processor for human operators. *SIGMOD*, 2011.
[12] A. Marshall. 1920. *Principles of economics*, 8, 1890.
[13] S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. John Wiley & Sons, Inc., 1990.
[14] B. Mozafari et al. Active learning for crowd-sourced databases. *arXiv preprint arXiv:1209.3686*, 2012.
[15] A. Parameswaran and H. Garcia-Molina et al. Crowdscreen: Algorithms for filtering data with humans. *SIGMOD*, 2012.
[16] A. Parameswaran and A. Sarma et al. Human-assisted graph search: it's okay to ask questions. *VLDB*, 2011.
[17] T. Saaty. *Mathematical methods of operations research*. Dover Publications, 2004.
[18] B. Trushkowsky, T. Kraska, M. J. Franklin, and P. Sarkar. Crowdsourced enumeration queries. *ICDE*, 2013.
[19] J. Wang, T. Kraska, M. Franklin, and J. Feng. Crowder: crowdsourcing entity resolution. *VLDB*, 2012.
[20] S. Whang et al. Question selection for crowd entity resolution. Technical report, Stanford InfoLab, 2012.
[21] X. Yang et al. On incentive-based tagging. *ICDE*, 2013.

APPENDIX

A. OPTIMAL SUBSTRUCTURE OF PAP

In this section, we show that $\mathcal{P}(B, \mathcal{T})$ exhibits *optimal substructure*, and so Bellman's optimality principle is hold. We let (Q, K) to be the optimal solution to $\mathcal{P}(B, \mathcal{T})$, where Q is the optimal value and $K = \{k_i | i = 1, 2, \ldots, n\}$ is the plurality of k_i to t_i in the optimal solution. Consider the subproblem $\mathcal{P}(B - k_n c_n, \mathcal{T}_{n-1})$. We claim that $(Q - \zeta_n(k_n), K - \{k_n\})$ is the optimal solution to this subproblem. Suppose that this is not true. Let (Q', K'), where $K' = \{k'_1, \ldots, k'_{n-1}\}$, be the optimal solution to $\mathcal{P}(B - k_n c_n, \mathcal{T}_{n-1})$, and $Q' > Q - \zeta_n(k_n)$. Let $(Q'', K'') = (Q' + \zeta_n(k_n),\ K' \cup \{k_n\})$. Since $\sum_{i=1}^{n-1} k'_i c_i \leq B - k_n c_n$, (Q'', K'') is a feasible solution to $\mathcal{P}(B, \mathcal{T})$. However, $Q'' = Q' + \zeta_n(k_n) > Q$, which violates the assumption that Q is the optimal value to $\mathcal{P}(B, \mathcal{T})$. Therefore, $(Q - \zeta_n(k_n), K - \{k_n\})$ must be the optimal solution to the subproblem.

B. DYNAMIC PROGRAMMING SOLUTION

Algorithm 2 `DP`

1: **Input:** $B, \mathcal{T}$ and p_i, c_i for $t_i \in \mathcal{T}$
2: **Output:** Q^*, K^*
3: **for** $b \leftarrow 0$ **to** B **do** $Q[b, 0] \leftarrow 0$
4: **for** $l \leftarrow 1$ **to** n **do**
5: **for** $b \leftarrow 0$ **to** B **do**
6: $Q[b, l] \leftarrow Q[b, l-1] + \zeta_l(0), y[b, l] \leftarrow 0$
7: **for** $k_l \leftarrow 1$ **to** $\left\lfloor \frac{b}{c_l} \right\rfloor$ **do**
8: **if** $Q[b, l] < Q[b - k_l c_l, l - 1] + \zeta_l(k_l)$ **then**
9: $y[b, l] \leftarrow k_l, Q[b, l] \leftarrow Q[b - k_l c_l, l - 1] + \zeta_l(k_l)$
10: $Q^* \leftarrow Q[B, n]$
11: $b \leftarrow B$
12: **for** $l \leftarrow n$ **downto** 1 **do**
13: $K^*[l] \leftarrow y[b, l], b \leftarrow b - y[b, l] \cdot c_l$
return (Q^*, K^*)

Our bottom-up, dynamic programming algorithm is shown in Algorithm 3. $Q[b, l]$ and $y[b, l]$ are both $B \times n$ arrays that store the optimal value of $\mathcal{P}(b, \mathcal{T}_l)$ and the optimal plurality k_l for t_l in this subproblem. So the optimal value of $\mathcal{P}(B, \mathcal{T})$ is $Q[B, n]$.

We first deal with the boundary case where $l = 0$. Then for each subproblem $\mathcal{P}(b, \mathcal{T}_l)$, we enumerate all the possible settings to k_l and find out the one at which the optimal value is achieved. $Q[b, l]$ and $y[b, l]$ are updated accordingly(Step 4 to 10). Thus, the optimal value $Q^* = Q[B, n]$.

Step 11 to 13 are used to recover the optimal plurality assignment K^* based on optimal plurality of k_l for each subproblem.

As we can see, all subproblems are scanned once during the process and each time all possible values of k_l are checked once from 0 to $\left\lfloor \frac{b}{c_l} \right\rfloor$ (which cannot exceed $\frac{B}{M}$) to find the optimal one. Besides, calculating the specific value of the quality function requires $O(F)$ time. Therefore, the time and space complexities of this algorithm are $O(\frac{nB^2F}{M})$ and $O(nB)$, respectively.

C. INCREMENTAL COMPUTATION OF ζ_i

Here we explain how to derive Equation 13. When k is odd, we have $\lceil \frac{k}{2} \rceil = \frac{k+1}{2}$, $\lceil \frac{k+2}{2} \rceil = \frac{k+1}{2} + 1$.

Consider C_{k+2}^l, where $l \geq \lceil \frac{k+2}{2} \rceil = \frac{k+1}{2} + 1$. By Pascal's Formula, we have $C_{k+2}^l = C_{k+1}^l + C_{k+1}^{l-1} = C_k^l + 2C_k^{l-1} + C_k^{l-2}$

Note that the above formula holds for $l \leq k$. For $l > k$, we have $C_{k+2}^{k+1} = 2 + k = 2C_k^k + C_k^{k-1}$, and $C_{k+2}^{k+2} = C_k^k$.

Let $r = \frac{k+1}{2}$. By breaking down each combination in $\zeta_i(k+2)$, i.e., C_{k+2}^l, by the above formula, and merging terms with the same combination together, we have:

$$\begin{aligned}\zeta_i(k+2) &= p_i(1-p_i)C_k^{r-1}p_i^r(1-p_i)^{k-r} \\ &\quad +(2p_i - p_i^2)C_k^r p_i^r(1-p_i)^{k-r} \\ &\quad + \sum_{l=r+1}^{k} C_k^l p_i^l (1-p_i)^{k-l}\end{aligned}$$

Since $\zeta_i(k) = \sum_{l=r}^{k} C_k^l p_i^l (1-p_i)^{k-l}$, we get

$$\begin{aligned}\zeta_i(k+2) - \zeta_i(k) &= p_i(1-p_i)C_k^{r-1}p_i^r(1-p_i)^{k-r} \\ &\quad +(2p_i - p_i^2 - 1)C_k^r p_i^r(1-p_i)^{k-r} \\ &= (p_i C_k^{r-1} - (1-p_i)C_k^r)p_i^r(1-p_i)^r\end{aligned}$$

Since $r = \frac{k+1}{2}$, we have $r+(r-1) = k$. Therefore $C_k^r = C_k^{r-1}$. Thus, we get

$$\begin{aligned}\frac{\zeta_i(k+2) - \zeta_i(k)}{p_i^r(1-p_i)^r} &= (p_i C_k^{r-1} - (1-p_i)C_k^r) \\ &= (p_i - (1-p_i))C_k^r \\ &= (2p_i - 1)C_k^r\end{aligned}$$

Since $d_i(k) = \zeta_i(k+2) - \zeta_i(k)$, we have

$$d_i(k) = C_k^r(2p_i - 1)p_i^r(1-p_i)^r. \tag{18}$$

For $k > 1$, we can derive $d_i(k+2)$ from $d_i(k)$ as follows.

$$\begin{aligned}d_i(k+2) &= d_i(k) \cdot p_i(1-p_i)\frac{C_{k+2}^{r+1}}{C_k^r} \\ &= d_i(k) \cdot 4p_i(1-p_i)\frac{k+2}{k+3}\end{aligned}$$

Hence, Equation 13 is proved.

D. PROOF OF THEOREM 1

This proof is based on the incremental computation of $\zeta_i(k)$ (c.f. Section 4.3 and Appendix C).

Recall that in the definition of $\zeta_i(k)$ in Equation 1, if $p_i = 1$, $\zeta_i(k) = 1$ for all $k > 0$. Hence, $\zeta_i(k)$ monotonically increases with k.

We next consider the case that $p_i \neq 1$. Since $\zeta_i(0) = 0$ and $\zeta_i(1) = p_i$, $\zeta_i(1) > \zeta_i(0)$ for all $0.5 \leq p_i < 1$. For $k \geq 1$, we have

$$\zeta_i(k+2) = \zeta_i(k) + d_i(k).$$

Recall that $d_i(k) = C_k^r(2p_i - 1)p_i^r(1-p_i)^r$, $r = \lceil \frac{k}{2} \rceil$ (Equation 18). We obtain for $0.5 \leq p_i < 1$, $d_i(k) > 0$. Thus, we have $\zeta_i(k+2) \geq \zeta_i(k)$ if and only if $0.5 \leq p_i < 1$.

Thus, $\zeta_i(k)$ monotonically increases with k if and only if $0.5 \leq p_i \leq 1$.

E. PROOF OF THEOREM 2

This proof is based on the incremental computation of $\zeta_i(k)$ (c.f. Section 4.3 and Appendix C).

If $p_i = 1$, $d_i(0) = 1$ and $d_i(k) = 0$ for any positive odd integer k. According to the relationship of $\Delta\zeta_i(k)$ and $d_i(k)$, we can see that $\Delta\zeta_i(k') \leq \Delta\zeta_i(k)$ for $k' > k$.

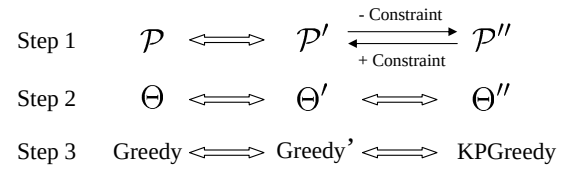


Figure 7: Analyzing effectiveness of `Greedy`.

We next consider the case where $p_i \neq 1$. According to Equation 18, we have $d_i(0) = p_i$ and $d_i(1) = 2(p_i - \frac{1}{2})p_i(1-p_i)$. Since $p_i(1-p_i) \leq \frac{1}{4}$, and $0.5 \leq p_i < 1$, we have

$$\Delta\zeta_i(1) = \frac{1}{2}d_i(1) \leq \frac{1}{4}(p_i - \frac{1}{2}) < p_i < d_i(0) = \Delta\zeta_i(0).$$

We next prove that for any positive odd integer k, $\Delta\zeta_i(k+2) \leq \Delta\zeta_i(k)$. This can be derived by the fact that

$$\frac{\Delta\zeta_i(k+2)}{\Delta\zeta_i(k)} = \frac{d_i(k+2)}{d_i(k)} = 4p_i(1-p_i)\frac{k+2}{k+3} \leq \frac{k+2}{k+3} < 1.$$

Thus, if $\Delta\zeta_i(k) \geq 0$, we have $\Delta\zeta_i(k+2) \leq \Delta\zeta_i(k)$. According to Theorem 1, $\zeta_i(k)$ is monotonically increasing with k if and only if $0.5 \leq p_i \leq 1$. Hence, $\Delta\zeta_i(k)$ monotonically decreases with k, i.e., $\zeta_i(k)$ satisfies the property of DR, if and only if $0.5 \leq p_i \leq 1$.

F. EFFECTIVENESS ANALYSIS OF `GREEDY`

In this section, we study the accuracy of `Greedy`. We follow the analyzing flow in Figure 7, and summarize Step 1 to 3 in Theorem 3 to 5, respectively. First, recall that $\mathcal{P}$ is the plurality setting problem defined in Section 3.4. We further define two problems, namely $\mathcal{P}'$ and $\mathcal{P}''$, as follows.

• **Problem $\mathcal{P}'$**:

$$\textbf{given} \quad B, \mathcal{T} \text{ and } p_i, c_i \text{ for } t_i \in \mathcal{T} \tag{19}$$

$$\textbf{maximize} \quad \sum_{i=1}^{n}\sum_{j=0}^{J_i} y_{i,j} \cdot b_{i,j} \tag{20}$$

$$\textbf{subject to} \quad \sum_{i=1}^{n}\sum_{j=0}^{J_i} y_{i,j} \cdot \gamma_{i,j} \leq B \tag{21}$$

$$y_{i,j} = 0 \text{ or } 1 \tag{22}$$

$$y_{i,j} \geq y_{i,j+1} \tag{23}$$

where

$$J_i = \frac{B}{2c_i} \tag{24}$$

$$b_{i,j} = \begin{cases} d_i(0), & j = 0, \\ d_i(2j-1), & \text{otherwise.} \end{cases} \tag{25}$$

$$\gamma_{i,j} = \begin{cases} c_i, & j = 0, \\ 2c_i, & \text{otherwise.} \end{cases} \tag{26}$$

Essentially, we would like to find all the values of $y_{i,j}$ such that the objective function of $\mathcal{P}'$, or Equation 20, is optimized.

• **Problem $\mathcal{P}''$**: identical to $\mathcal{P}'$, except that an optimization constraint, Equation 23, is removed. Notice that $\mathcal{P}''$ is a variant of a 0-1 knapsack problem [13] with $\sum_{i=1}^{n}(J_i + 1)$ items; each item has a value of $b_{i,j}$ and a cost of $\gamma_{i,j}$.

Let us denote the optimal solutions of $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{P}''$ be Θ, Θ', and Θ'' respectively. We next present three lemmas, with the aid of Figure 7.

THEOREM 3. *$\mathcal{P}$ is equivalent to $\mathcal{P}'$.*

PROOF. Constraints 22 and 23 in $\mathcal{P}'$ indicate that, in any feasible solution for $\mathcal{P}'$, there exists a_i such that for every $j < a_i$, $y_{i,j} = 1$, and for every $j \geq a_i$, $y_{i,j} = 0$. If we let $k_i = \max\{2a_i - 1, 0\}$, we have $\sum_{i=1}^{n}\sum_{j=0}^{J_i} y_{i,j}b_{i,j} = \sum_{i=1}^{n}\zeta_i(k_i)$, which is exactly the objective function of $\mathcal{P}$ (Equation 7). We can also deduce that Equation 21 is equivalent to the budget constraint of $\mathcal{P}$ (i.e., Equation 8), since $\sum_{i=1}^{n} y_{i,j}\gamma_{i,j} = \sum_{i=1}^{n} k_i c_i$. Thus, $\mathcal{P}$ and $\mathcal{P}'$ represent the same problem, as illustrated by Figure 7. □

THEOREM 4. *The values of the objective functions of $\mathcal{P}$, $\mathcal{P}'$, and $\mathcal{P}''$, computed respectively by the solutions of Θ, Θ', and Θ'', are identical.*

PROOF. The proof is illustrated in Figure 7. First, notice that $\mathcal{P}$ can be rewritten as $\mathcal{P}'$ (Theorem 3). Therefore, the values of the objective functions obtained by Θ and Θ' are identical.

We next prove that the values of the objective functions obtained by Θ' and Θ'' are also the same. This is equivalent to showing that the solution of Θ'' satisfies Equation 23. Suppose that this is not true. This means that in algorithm Θ'', the optimal solution for $y_{i,j}$ is $y^*_{i,j}$, and $\exists\ i_0,\ j_0$, and j_1 such that $j_1 = j_0 + 1$, $y^*_{i_0,j_0} = 0$, and $y^*_{i_0,j_1} = 1$. Now, consider another solution where $y^+_{i,j} = y^*_{i,j}$, except that $y^+_{i_0,j_0} = 1$ and $y^+_{i_0,j_1} = 0$. Since $\gamma_{i,j_1} \geq \gamma_{i,j_0}$, we have $\sum_i\sum_j y^+_{i,j}\cdot\gamma_{i,j} \leq \sum_i\sum_j y^*_{i,j}\cdot\gamma_{i,j} \leq B$. Thus, $y^+_{i,j}$ is a feasible solution to $\mathcal{P}''$. According to Theorem 2, $d_i(2j_1 - 1) < d_i(2j_0 - 1)$, and so $b_{i,j_1} < b_{i,j_0}$. Thus, $\sum_i\sum_j y^+_{i,j}\cdot b_{i,j} > \sum_i\sum_j y^*_{i,j}\cdot b_{i,j}$, which violates the assumption that $y^*_{i,j}$ is the optimal solution to $\mathcal{P}''$. Hence, the theorem holds. □

THEOREM 5. *The values of the objective functions of $\mathcal{P}$ and $\mathcal{P}''$, computed respectively by the solutions of* `Greedy` *and* `KPGreedy`*, are identical.*

PROOF. Since $\mathcal{P}$ and $\mathcal{P}'$ are equivalent (Theorem 3), we can adapt `Greedy` to solve $\mathcal{P}'$. The resulting algorithm, called `Greedy'`, attains the same objective function values as `Greedy`. Specifically, let $\hat{k}_i$, $\hat{y}_{i,j}$ be the solution of `Greedy`, and `Greedy'` respectively. We can obtain:

$$\hat{y}_{i,j} = \begin{cases} 1, & \text{if } j < \left\lceil \frac{\hat{k}_i}{2} \right\rceil, \\ 0, & \text{otherwise.} \end{cases} \quad (27)$$

We can then see that `Greedy` and `Greedy'` yields the same objective function values for problems $\mathcal{P}$ and $\mathcal{P}'$ respectively.

As explained before, $\mathcal{P}''$ is a variant of the 0-1 knapsack problem, which can be solved effectively by a well-known greedy algorithm in [13], denoted as `KPGreedy`. We next prove that $\hat{y}_{i,j}$ is the same solution obtained by `KPGreedy`. We first state without proof that $h_i(k) = \frac{b_{i,j}}{\gamma_{i,j}}$, where $j = \lceil \frac{k}{2} \rceil$. Using Theorem 2, we deduce that $h_i(k') \leq h_i(k)$ for all $k' \geq k$. Hence,

$$\frac{b_{i,j'}}{\gamma_{i,j'}} \leq \frac{b_{i,j}}{\gamma_{i,j}} \qquad \forall j' \geq j. \quad (28)$$

We remark that Equation 28 is a criterion used by `KPGreedy` to select the next item. Thus, `Greedy'` is the same as `KPGreedy`. Combined with the previous results, we can see that the solutions of `Greedy` and `KPGreedy` produce the same objective function values for respectively $\mathcal{P}$ and $\mathcal{P}''$. The theorem is thus proved. □

Effectiveness of `Greedy`. In [13], it was shown that `KPGreedy` attains a worst-case effectiveness ratio of 0.5 with respect to the optimal solution, Θ'', for Problem $\mathcal{P}''$. Using Theorems 4 and 5, we can see that `Greedy` is also a 0.5-approximation algorithm. In practice, `KPGreedy` has a close-to-optimal effectiveness [17]. This can also be said for `Greedy`. Let us examine `Greedy` experimentally next.

G. PSEUDO-CODE FOR GROUP-BASED ALGORITHMS

Algorithm 3 `Group-based DP`

Input: B, $\mathcal{T}$ and p_i, c_i for $t_i \in \mathcal{T}$
Output: Q^*, K^*
$\mathcal{G} \leftarrow$ GETGROUPS($\mathcal{T}$)
$m_j \leftarrow$ no. of HITs in $g_j \in \mathcal{G}$
$\hat{t}_j \leftarrow$ representative of $g_j \in \mathcal{G}$
$\hat{\zeta}_j \leftarrow$ quality function of $\hat{t}_j$
$\alpha_j \leftarrow$ cost of $\hat{t}_j$
for $b \leftarrow 0$ **to** B **do** $Q[b, 0] \leftarrow 0$
for $j \leftarrow 1$ **to** $|\mathcal{G}|$ **do**
  for $b \leftarrow 0$ **to** B **do**
    $Q[b, j] \leftarrow Q[b, j-1]$, $y[b, j] \leftarrow (0, 0)$
    for $m \leftarrow 1$ **to** m_j **do**
      if $\alpha_j \cdot m \leq b$ **and** $Q[b, j] < Q[b - \alpha_j \cdot m, j-1] + m \cdot \hat{\zeta}_j(1)$ **then**
        $Q[b, j] \leftarrow Q[b - \alpha_j \cdot m, j-1] + m \cdot \hat{\zeta}_j(1)$
        $y[b, j] \leftarrow (0, m)$
    for $k_j \leftarrow 1$ **to** $\left\lfloor \frac{b}{m_j \alpha_j} \right\rfloor$ **do**
      for $m \leftarrow 0$ **to** m_j **do**
        $C \leftarrow \alpha_j \cdot (k_j \cdot m_j + 2 \cdot m)$
        if $C \leq b$ **and** $Q[b, j] < Q[b - C, j-1] + (m_j - m) \cdot \hat{\zeta}_j(k_j) + m \cdot \hat{\zeta}_j(k_j + 2)$ **then**
          $Q[b, j] \leftarrow Q[b - C, j-1] + (m_j - m) \cdot \hat{\zeta}_j(k_j) + m \cdot \hat{\zeta}_j(k_j + 2)$
          $y[b, j] \leftarrow (k_j, m)$
$Q^* \leftarrow Q[B, n]$
$b \leftarrow B$
for $j \leftarrow |\mathcal{G}|$ **downto** 1 **do**
  $(k, m) \leftarrow y[b, j]$
  set plurality of all HITs in g_j to k
  if $k = 0$ **then**
    select m HITs from g_j, and increase their plurality by 1
    $b \leftarrow b - \alpha_j \cdot m$
  else
    select m HITs from g_j, and increase their plurality by 2
    $b \leftarrow b - \alpha_j \cdot (k \cdot m_j + 2 \cdot m)$
return (Q^*, K^*)

Algorithm 4 `Group-based Greedy`

Input: $B, \mathcal{T}$ and p_i, c_i for $t_i \in \mathcal{T}$
Output: $\vec{k}$
$\mathcal{G} \leftarrow$ GETGROUPS($\mathcal{T}$)
$\hat{t}_j \leftarrow$ representative of $g_j \in \mathcal{G}$
$\alpha_j \leftarrow$ cost of $\hat{t}_j$
$b \leftarrow B$
for $j \leftarrow 1$ to $|\mathcal{G}|$ **do**
    set plurality of all HITs in g_j to zero
    calculate marginal gain of $\hat{t}_j$ by Equation 14
    Q.push($\hat{t}_j$) // Q is a priority queue.
while $b > 0$ and Q is nonempty **do**
    $\hat{t}_j \leftarrow Q$.pop() // The MCQ with the largest marginal gain is popped.
    if plurality of $\hat{t}_j = 0$ **then** $\Delta k \leftarrow 1$
    else $\Delta k \leftarrow 2$
    $m \leftarrow \lfloor \frac{b}{\alpha_j \Delta k} \rfloor$
    if $m <$ no. of HITs in g_j **then** // There is not enough budget.
        select m HITs from g_j, and increase their plurality by Δk
        $b \leftarrow b - \alpha_j \Delta k \times m$
    else
        increase plurality of all HITs in g_j by Δk
        $b \leftarrow b - \alpha_j \Delta k \times$ no. of HITs in g_j
        calculate marginal gain of $\hat{t}_j$ by Equation 14
        Q.push($\hat{t}_j$)
return $\vec{k}$