High-Throughput Crowdsourcing Mechanisms for Complex Tasks

Guido Sautter[1], Klemens Böhm[1]

[1] KIT, Am Fasanengarten 5, 76128 Karlsruhe, Germany
{guido.sautter, klemens.boehm}@kit.edu

Abstract. Crowdsourcing is popular for large-scale data processing endeavors that require human input. However, working with a large community of users raises new challenges. In particular, both possible misjudgment and dishonesty threaten the quality of the results. Common countermeasures are based on redundancy, giving way to a tradeoff between result quality and throughput. Ideally, measures should (1) maintain high throughput and (2) ensure high result quality at the same time. Existing work on crowdsourcing mostly focuses on result quality, paying little attention to throughput or even to that tradeoff. One reason is that the number of tasks (individual atomic units of work) is usually small. A further problem is that the tasks users work on are small as well. In consequence, existing result-improvement mechanisms do not scale to the number or complexity of tasks that arise, for instance, in proofreading and processing of digitized legacy literature. This paper proposes novel result-improvement mechanisms that (1) are independent of the size and complexity of tasks and (2) allow to trade result quality for throughput to a significant extent. Both mathematical analyses and extensive simulations show the effectiveness of the proposed mechanisms.

Keywords: Crowdsourcing, Data Quality, Throughput

1 Introduction

Recently, crowdsourcing has become popular for tasks that require human input to increase data quality. Crowdsourcing distributes small pieces of a large effort to many users who make small contributions, usually over the Internet. Crowdsourcing has been used successfully for many tasks, e.g., image labeling [4, 10], double-keying individual words for OCR correction [11], grading the relatedness of word pairs for ontology construction [3, 7], or word sense disambiguation [8]. Crowdsourcing poses a number of challenges. In particular, there is no guarantee that user inputs are correct. Here, an input is correct if it is identical to what respective experts would agree on [3]. There are several reasons for incorrect inputs. We distinguish:

- Users can accidentally make **mistakes** due to sloppiness or misjudgment, even if they contribute solely because of interest in the project, like in [2, 4].
- Especially if they receive some reward for their inputs, users may **cheat** to reduce their effort. In particular, they may contribute arbitrary random input instead of

working thoughtfully. Especially if the reward is external, e.g., monetary, like in [3, 8, 11], gathering the reward might well be the only motivation. [3] has observed users following this strategy.

To ease presentation, we introduce several notions: A **Task** T is the unit of work assigned to contributors. Each task consists of **Decisions** $D_1 \ldots D_d$ the user is supposed to take. In [11], for instance, a task consists of two decisions, namely on the correct transcriptions of two words from images. In [3] in turn, tasks consist of 12 decisions, on the relatedness of 12 term pairs. The **original state** of a task is its status before any user has worked on it. Further, the final status, i.e., after a crowdsourcing system regards the task as completed, is its **result**. Finally, **inputs** are the contributions of individual users who work on a task. We formalize these notions in Section 2.

In our context, it is important that incorrect inputs occurring for different reasons (see above) exhibit different properties and require different countermeasures. The crowdsourcing projects mentioned before have developed different respective strategies. One countermeasure against mistakes is redundancy, i.e., to obtain contributions of several users for each task. However, redundancy severely reduces throughput. A mechanism to discourage cheating is to probe users with tasks the system already knows the correct result for, e.g. CAPTCHAs [9].

The tasks crowdsourced in previous projects were relatively simple, e.g., double-keying words [11], grading the relatedness of word pairs [3], or finding meaningful labels for images [10]. If tasks consist of more than one decision, like in [3, 8], the individual decisions are mutually independent and can be freely combined into tasks. Tasks in other applications are much more complex. An example is the generation of semantic markup for legacy documents. In general, the tasks consist of multiple decisions that belong together, or decisions are very complex. In Distributed Proofreaders [5] for instance, decisions are transcriptions of entire document pages, including both the word level and the structure of the page. Tasks of similar complexity arise in the Madagascar Project [6]. Because of the complex decisions and the high level of redundancy, throughput is low in Distributed Proofreaders, around 18,000 documents in 10 years. A more promising approach would be to use a mechanism like reCAPTCHA [11] for the word level transcription and to proofread the page structure separately (we argue). Even if structuring a page can be broken into several decisions, these decisions still form a unit that a user should work on as a whole.

This calls for crowdsourcing mechanisms that (1) effectively counter errors and thus enforce data quality, (2) yield a high throughput, and (3) work with large tasks. Previous crowdsourcing projects have mostly addressed (1), but not in combination with (2) or (3). In particular, they have not addressed the tradeoff between data throughput and result quality. In this paper, we therefore study generic quality-enforcement techniques that are independent of the nature of the tasks and counter both mistakes and cheating:

- **v-Voting** counters mistakes. For each task, it obtains inputs from several users and aggregates them to the overall result. Unlike static redundancy, it uses a voting mechanism (controlled by parameter 'v'), which reduces the number of inputs required.
- **Vote Boosting** builds upon v-Voting, to further increase throughput. It increases the weight of inputs from users who are known from prior observations to make few mistakes, thus reducing the number of answers required. If a reward system is

in place, the reward can be specified to increase with the weight of the vote. We expect this to foster high-quality inputs.

These mechanisms assume that most users contribute useful inputs, an assumption common to crowdsourcing projects. If most inputs were arbitrary, there would be no chance of obtaining any meaningful data at all. Note that the mechanisms ensure data quality in the presence of cheating, but do not prevent or discourage dishonest user behavior in itself. This would require some sort of user probing mechanism, e.g. one akin to CAPTCHA.

To assess the effectiveness of our mechanisms, we have conducted a thorough evaluation, considering both mistakes and cheating. It comprises theoretical analyses of the expected throughput and result quality, as well as simulations. The results are that v-Voting and Vote Boosting serve their respective purpose well; in particular, they yield the same result quality as static redundancy with fewer inputs.

This paper is part of a larger effort that will also cover user experiments. Since such experiments are expensive even when covering only few points in the parameter space, it is mandatory to study the alternatives with other methods beforehand. This paper reports on the respective results.

Paper Outline. Section 2 introduces formal notions required for our analysis. Section 3 reviews related work. Section 4 provides an in-depth explanation and mathematical assessment of the data-quality-enforcement mechanisms. Section 5 features simulation results, Section 6 concludes.

2 Formal Notions

To facilitate formal analysis of crowdsourcing, this section formalizes some notions.

2.1 Decisions, Tasks & Functions

Definition: A **Decision D** is an atomic parameter set by a user. □

For instance, a decision is to classify a named entity, or to specify if a given paragraph belongs to a document's main text or is a page header or a caption.

Notation: Opts(D) := {O_1, …, O_o} denotes the **options available for D**. **N ∉ Opts(D)** denotes the **null option**, which models the case that D is undecided. □

For instance, the options for a decision can be the classes available for named entities or the paragraphs types. Note that Opts(D) can be large. In particular, this is the case when users have to type words into a text field, like in [10, 11].

At every point of its time of residence in the crowdsourcing system, a decision D has an option S(D) ∈ Opts(D) ∪ {N} assigned to it. We refer to S(D) as the **state** of D. There are several dedicated states to be distinguished:

Notation: S_O(D) ∈ Opts(D) ∪ {N} is the **original state of D**, i.e., the state assigned to D when it enters the crowdsourcing system. **$S_{I,U}$(D)** ∈ Opts(D) denotes the **state a user U has assigned to D in his input**, i.e., the option this user has selected. An input state cannot be N. **S_R(D)** ∈ Opts(D) ∪ {N} is the **result of D**, i.e., the state of D when leaving the system. A null result, i.e., $S_R(D) = N$, indicates that the system could not

determine a meaningful result for D. $\mathbf{S_C(D)} \in$ Opts(D) is the **correct state of D**, i.e., the outcome respective experts would agree on. **Input(D)** = ($S_{I,U1}(D)$, …, $S_{I,Uu}(D)$) is the **input list of D**, containingthe inputs that users U_1, …, U_u have contributed to D. □

For instance, the original state can be the class an NLP tool has assigned to a named entity.

Definition: A **Task T** = (D_1, …, D_d) is the unit of work assigned to users, consisting of one or more decisions D_1, …, D_d. □

The individual decisions that make up a task can be **connected** or **independent**. In the first case, a crowdsourcing system cannot modify tasks by adding or removing decisions. In the latter case, the system can freely put together decisions to tasks.

At any point of its time of residence in the crowdsourcing system, a task T has a **state** S(T). The state of a task is the composition of the states of the individual decisions it consists of, namely S(T) = ($S(D_1)$, …, $S(D_d)$). Analogously to individual decisions, we make the following distinctions:

Notation: $\mathbf{S_O(T)}$ = ($S_O(D_1)$, …, $S_O(D_d)$) is the **original state of T**. $\mathbf{S_{I,U}(T)}$ = ($S_{I,U}(D_1)$, …, $S_{I,U}(D_d)$) is the **input of U to T**. $\mathbf{S_R(T)}$ = ($S_R(D_1)$, …, $S_R(D_d)$) is the **result of T**, i.e., its state after all user interactions. $\mathbf{S_C(T)}$ = ($S_C(D_1)$, …, $S_C(D_d)$) is the **correct state** of T. **Input(T)** = ($S_{I,U1}(T)$, …, $S_{I,Uu}(T)$) is the **input list of T**, comprising the inputs that users U_1, …, U_u have contributed to T. □

Definition: An **abstract input-aggregation function Result(Input(T))** is a function of type Input(T) → {∅, $S_R(T)$} that computes the result of T from Input(T). □

A crowdsourcing system successively obtains inputs from users and adds them to Input(T). It evaluates Result(Input(T)) after the addition of each input; once Result(Input(T)) does not return ∅, T is complete, and no further input is required.

Notation: Work(T) denotes the expected value of |Input(T)| at the moment the input-aggregation function returns a non-empty result. □

In other words, Work(T) is the expected number of inputs to collect.

2.2 Types of Errors

This section investigates which errors can occur in the inputs that users contribute to crowdsourced tasks. Note that it is not our goal to enable crowdsourcing systems to distinguish between these errors. In general, this is not possible. This is because an error typically does not reveal the motivation of the user who incurred it. However, errors occurring for different reasons differ in their statistical nature, i.e., follow different patterns of occurrence. They thus require specific countermeasures.

In general, there is an error in a decision D if $S(D) \neq S_C(D)$. We are interested in the prevention of errors in the result of D, namely that $S_R(D) \neq S_C(D)$. Orthogonal to the distinction discussed below, there are two types of errors: (1) **Miss Errors** are errors that remain undetected; formally, a miss error exists if $S_O(D) \neq S_C(D)$, and $S_R(D) \neq S_C(D)$. (2) **Added Errors** are errors introduced by users; i.e., $S_O(D) = S_C(D)$, and $S_R(D) \neq S_C(D)$.

Accidental Errors are errors in the inputs of benevolent users incurred by mistake, be it out of sloppiness, lack of focus, or erroneous judgment. We assume that accidental errors occur randomly. Further, errors resulting from sloppiness tend to be miss errors, while the ones resulting from misjudgments can be of both types.

Notation: P('accidental miss') is the average probability across all users that some user accidentally misses an error in a decision D of a task T. **P('accidental add')** is the average probability that some user accidentally adds an error in a decision D. □

Cheating Errors occur because users do not bother to contribute thoughtful input. If the original state of a task $S_O(T)$ is a valid input, we assume that cheating users simply submit $S_O(T)$ as their input because this is the least effort possible. If the original state of a task consists of null values, like the initially empty text fields in [10, 11], we assume cheating users to randomly select an option from Opts(D) as their input. In the former case, adding an error requires making a change to the original state of a task. So submitting the original state as an input without changing anything cannot add any error. Thus, cheating errors generally are miss errors in this case.

Notation: P('cheat') is the average probability that some user cheats on a task T and thereby contributes an input with miss errors for all errors in $S_O(T)$. □

Combined Error Probability. To simplify subsequent computations, we aggregate the individual error probabilities.

Notation/Observation: P('miss') is the average probability of a miss error in a single input. This happens if a user cheats on T, or if he does not cheat and misses the error in some decision $D \in T$ by mistake, namely:

$$P(\text{'miss'}) = P(\text{'cheat'}) + (1-P(\text{'cheat'})) \cdot P(\text{'accidental miss'})$$

P('add') is the average probability of an add error in a single input. This happens if a user does not cheat and adds an error in some decision D by mistake, namely:

$$P(\text{'add'}) = (1-P(\text{'cheat'})) \cdot P(\text{'accidental add'})$$ □

2.3 Parameters & Figures

This section lists the exogenous and endogenous parameters of crowdsourcing systems and describes the optimization goals.

The **exogenous parameters** are: (1) The nature of the tasks, i.e., the number of decisions they consist of, the number of options in the decisions, and whether the decisions are connected or not. (2) The accuracy of the initial states of the tasks, or, in other words, the number of errors to correct in each task. (3) The probabilities of users to make accidental errors and to cheat on tasks.

The sole **endogenous parameter** is the answer-aggregation function in use and its parameterization.

The **numbers to optimize** are: (1) the expected accuracy of task results, namely $P(\text{'}S_R(T) = S_C(T)\text{'})$, and (2) the expected number of inputs required to achieve this accuracy, i.e., the expected value of |Input(T)|. The latter is particularly important when using third-party crowdsourcing platforms that require a fixed monetary reward per input, like the Amazon Mechanical Turk [1].

3 Related Work

This section discusses recent crowdsourcing projects, the mechanisms used to enforce data quality, and some experiences.

3.1 r-Redundancy

Many projects [3, 4, 8] use a simple input-aggregation function, namely **r-Redundancy**, where r is the parameter specifying the number of inputs required. r-Redundancy means that, once r inputs are given for a task T, the most frequently given input in Input(D) becomes the result of D, for each Decision D in T. r usually is an odd number. r-Redundancy is suboptimal with regard to throughput. This is because a task always takes r inputs to complete, even if the first (r+1)/2 inputs agree completely.

Eckert et al. [3] use a 5-redundant approach to arrange terms into a concept hierarchy. Each task consists of 12 independent decisions. Each decision was to compare a pair of terms with regard to relatedness and relative generality. To detect inputs of low quality, each task included two very easy decisions P and Q. If users got them wrong, this served as an indicator for them not paying attention. With this mechanism, [3] achieved a degree of data quality comparable to that of a concept hierarchy constructed from the same terms by domain experts. However, embedding decisions with known results like P and Q in every task only works with independent decisions that a crowdsourcing system can freely bundle into tasks. It is impossible to use with tasks that consist of connected decisions.

Snow [8] successfully used 10-Redundancy based crowdsourcing for detail level NLP tasks like word sense disambiguation, achieving a result quality similar to [3]. All tasks consist of 30 independent decisions bundled randomly. The system did not include any mechanisms to detect or filter inputs of low quality.

3.2 Agreement Games

Agreement Games *synchronously* obtain inputs from two random users, referred to as U and V. Each task T usually consists of a single decision D, and usually $S_O(D) = N$. If the two inputs agree, they count as correct, and both users get a reward.

Von Ahn has successfully used this approach for image labeling [10]. **OntoGame** [7] has shown that it also works well for ontology construction and alignment, and for named entity disambiguation. However, the agreement approach is unlikely to work well for tasks with multiple decisions. This is because such tasks make it much harder for users to make inputs that agree in all decisions – a single mistake in one input renders both inputs useless.

3.3 Other Approaches

ReCAPTCHA [11] is a crowdsourcing project that double-keys images of document pages in a word-by-word fashion. The CAPTCHAs users have to solve consist of two random word images. One of them is the crowdsourcing task T, a single decision D on the correct transcription of the given word image. The other one is the actual CAPTCHA, referred to as C in the following, a word image whose correct transcription $S_C(C)$ is already known to the system. The presence of the CAPTCHA C that is indistinguishable from the actual task T (= {D}) counters cheating well.

ReCAPTCHA considers an input for D only if the CAPTCHA is solved, i.e., $S_I(C) = S_C(C)$. A task is complete as soon as there are 3 agreeing inputs.

However, reCAPTCHA tasks are tiny. Tasks that take more time are impractical as CAPTCHAs. Furthermore, insisting on agreeing inputs is impractical with regard to throughput if tasks consist of multiple decisions, as we will show.

Another crowdsourcing project related to the digitization of legacy documents is **Distributed Proofreaders** [5]. Its purpose is to correct OCR errors by means of redundancy. Tasks consist of one very large decision, namely the transcript of an entire page. Data throughput has been low so far, around 18,000 works in roughly eight years. A more sophisticated process separating the pages into smaller chunks might be more promising, e.g., using reCAPTCHA on the word level.

The **GalaxyZoo** [4] project had over a million galaxy images classified into six basic categories by over 10.000 volunteers in less than 200 days. Their system presented each user randomly selected images. However, this approach requires the whole set of tasks to be available from the start, which is not a given in digitization efforts. In addition, GalaxyZoo computed results only in the very end, using a centrality measure to weight the inputs of individual users.

4 High-Throughput Crowdsourcing

To facilitate crowdsourcing of large numbers of complex tasks like proofreading digitized documents, this section now introduces respective data-quality-enforcement mechanisms. To ease presentation, we first investigate a base case that assumes a single input to complete a task. We then present our mechanisms and evaluate them.

We use the following running example: Think of a task $T = \{D_1, D_2, D_3, D_4\}$. D_i is determining the type of the i-th paragraph in a page. Further suppose that

$Opts(D_i)$ = {'page header', 'main text', 'caption', 'footnote'},
$S_O(T)$ = ('main text', 'main text', 'caption', 'footnote'), and
$S_C(T)$ = ('page header', 'main text', 'main text', 'main text').

This corresponds to only 25% accuracy in automated classification, a very low value. We chose this below-standard value for presentation purposes.

For our analysis, we use conservative, yet realistic figures. Namely, we assume that on average, for an individual decision D in a generic task T

$P('S_O(D) = S_C(D)') = 80\%$, $P('miss') = 10\%$, and $P('add') = 5\%$.

4.1 Base Case

As the baseline for assessing the effectiveness of individual countermeasures, we first formalize the base case ('BC'), i.e., that exactly one user contributes to each task.

Then, the probabilities $P_{BC}('miss')$ of a miss error and $P_{BC}('add')$ of an add error occurring in a decision D are

$P_{BC}('miss') = P('miss')$, $P_{BC}('add') = P('add')$

This results in the following probability of a correct result:

$$P_{BC}('S_R(D)=S_C(D)') = 1 - P('S_O(D)=S_C(D)') \cdot P_{BC}('add') - P('S_O(D)\neq S_C(D)') \cdot P_{BC}('miss')$$

Note that always $Work_{BC}(T) = 1$, representing the optimal throughput. With the values from the running example, we obtain

$P_{BC}(\text{`}S_R(D)=S_C(D)\text{'}) = 0.94$ and $P_{BC}(\text{`}S_R(T)=S_C(T)\text{'}) \approx 0.7807$.

4.2 v-Voting

v-Voting ('V') is a means to counter accidental errors. As r-Redundancy, it does so by obtaining and aggregating several inputs for each task. As opposed to r-Redundancy, it uses an agreement-based input-aggregation function. That is, there is a fixed level of agreement to reach, but no fixed number of inputs to obtain. [11] uses this technique for individual words, with a fixed v = 3. We generalize it here to a parametric level of agreement, referred to as v, and for any multi-decision task.

Notation: $Result_V(Input(T))$ is the input-aggregation function for v-Voting. $R_V(Input(D))$ is an auxiliary function that computes if there is an agreed-upon result for a decision D. Formally, this is:

$$R_V(Input(D)) := \begin{cases} \{O_0\} & \text{if } \exists\, O_0 \in Opts(D) \text{ such that } \left|\{S_I(D) \in Input(D) \mid S_I(D) = O_0\}\right| \geq v \\ N & \text{otherwise} \end{cases}$$

$$Result_V(Input(T)) := \begin{cases} \varnothing & \text{if } \exists\, D \in T : R_V(Input(D)) = N \\ (R_V(Input(D)) \text{ for all } D \in T) & \text{otherwise} \end{cases}$$

□

Note that $Result_V(Input(T))$ avoids the ambiguous cases that can occur with r-Redundancy. Another advantage of $Result_V(Input(T))$ is that it requires fewer inputs than r-Redundancy for the same expected result quality. Further note that $Result_V(Input(T))$ computes the result decision-wise and does not require whole inputs to agree, in contrast to [11].

Example 1. Suppose that v = 2, that three users U_1, U_2, and U_3 contribute inputs to the task T from the running example, and that the inputs are as follows:

$S_{I,U1}(T)$ = ('page header', 'main text', 'main text', 'footnote')
$S_{I,U2}(T)$ = ('main text', 'main text', 'main text', 'main text')
$S_{I,U3}(T)$ = ('page header', 'main text', 'caption', 'main text')

Even though no two inputs are equal, and all deviate from $S_C(T)$ in one decision, at least two inputs agree for each decision. Namely, the agreed-upon overall result $S_R(T)$ is ('page header', 'main text', 'main text', 'main text'), which is equal to $S_C(T)$, even though none of the users actually provided this input. Had users U_1 and U_2 given the same overall input, U_3 would not have been asked to contribute an input to T at all. ■

Decision-wise voting can considerably decrease the number of inputs required for an agreed-upon result, as illustrated in the example. The larger the number of decisions a given task comprises, the higher the advantage.

Formal Analysis. What is the overall probability of a correct result for a task T, i.e., $P_V(\text{`}S_R(T) = S_C(T)\text{'})$? We compute this in the following. For ease of presentation, we assume v = 2. To keep the computation simple, we further assume the worst case that, if several inputs contain add errors on a decision D of a task T, these errors are identical and become part of the result of T. This actually holds only for binary decisions, i.e., $|Opts(D)| = 2$. In non-binary decisions like the task from the running example, the assumption heavily increases the probability of an error. It helps us because it restricts $|Input(T)|$ and thus reduces the number of cases to consider. Our

simulations will show that |Input(T)| barely increases for |Opts(D)| > 2, in the range of a few percent, over a wide range of values for the other exogenous parameters.

Notation: P_V('miss') and P_V('add') denote the probabilities of a miss error and an add error occurring in the result of a decision $D \in T$, respectively. □

Informally, an error in the result of a decision D occurs if the first two inputs are erroneous, and if one of the two first and the third input are erroneous. Formally, P_V('miss') and P_V('add') are as follows:

$$P_V(\text{'miss'}) = 3 \cdot P(\text{'miss'})^2 - 2 \cdot P(\text{'miss'})^3$$

$$P_V(\text{'add'}) = 3 \cdot P(\text{'add'})^2 - 2 \cdot P(\text{'add'})^3$$

The overall probability for a decision $D \in T$ to be correct in the result then is:

$$P_V(\text{'}S_R(D) = S_C(D)\text{'}) = 1 - P(\text{'}S_O(D)=S_C(D)\text{'}) \cdot P_V(\text{'add'}) - P(\text{'}S_O(D)\neq S_C(D)\text{'}) \cdot P_V(\text{'miss'})$$

The overall probability to obtain a correct result for a task T consisting of d decisions $D_1 \ldots D_d$, is:

$$P_V(\text{'}S_R(T) = S_C(T)\text{'}) = P_V(\text{'}S_R(D) = S_C(D)\text{'})^d$$

Example 2. To illustrate the above, we compute the probability of a correct result for the task from the running example, with the exogenous parameters given there:

$$P_V(\text{'}S_R(D) = S_C(D)\text{'}) = 0.9886 \text{ and } P_V(\text{'}S_R(T) = S_C(T)\text{'}) \approx 0.9552$$

In the base case, the respective values are 0.94 and 0.7807. With no user input at all, the probability of a correct result would be, just for comparison: $0.8^4 = 0.4096$ ■

Discussion. In Example 2, 2-Voting increases the probability of a correct result for the example task T to about 96% from about 78% in the base case. This corresponds to a reduction of error by a factor of about 6, for the at most threefold effort.

Note that accuracy, for instance that of classifiers, is usually measured for individual objects. This corresponds to the individual decisions of a task. In this example, 2-Voting increases the probability of a correct final result for a decision D of a task T from 94% to about 99%. This corresponds to a reduction of error by a factor of almost 6 compared to the base case, again, for at most three times the effort.

The following notions are auxiliary; we use them to formally derive our main results, the computation of the expected throughput $Work_{V2}(T)$.

Notation: P('$S_{I,U1}(D)=S_{I,U2}(D)$') is the probability that the first two inputs $S_{I,U1}(D)$ and $S_{I,U2}(D)$ agree for a decision D. □

Informally, this is the probability that either none or both $S_{I,U2}(D)$ and $S_{I,U2}(D)$ are erroneous in some way. Formally, it is as follows:

$$\begin{aligned} P(\text{'}S_{I,U1}(D)=S_{I,U2}(D)\text{'}) = {} & P(\text{'}S_O(D)=S_C(D)\text{'}) \cdot (P(\text{'add'})^2 + (1-P(\text{'add'}))^2) \\ & + P(\text{'}S_O(D)\neq S_C(D)\text{'}) \cdot (P(\text{'miss'})^2 + (1-P(\text{'miss'}))^2) \end{aligned}$$

Notation: P('$S_{I,U1}(T) = S_{I,U2}(T)$') denotes the probability that the first two inputs $S_{I,U1}(T)$ and $S_{I,U2}(T)$ agree for an entire task T. □

Formally, this is: $P(\text{'}S_{I,U1}(T) = S_{I,U2}(T)\text{'}) = P(\text{'}S_{I,U1}(D) = S_{I,U2}(D)\text{'})^{|T|}$

Throughput. The actual increase in effort in comparison to the base case depends on the probability $P(\text{'}S_{I,U1}(T)=S_{I,U2}(T)\text{'})$ of the first two inputs to agree on all decisions in T, namely:

$$Work_{V2}(T) = 2 \cdot P(\text{'}S_{I,U1}(T) = S_{I,U2}(T)\text{'}) + 3 \cdot P(\text{'}S_{I,U1}(T) \neq S_{I,U2}(T)\text{'})$$

Further, $Work_{V2}(T) / Work_{BC}(T)$ is the overhead 2-Voting incurs in comparison to the base case. Likewise, $1 - Work_{V2}(T) / Work_R(T)$ is the reduction in effort 2-Voting yields in comparison to 3-Redundancy.

Example 3. With the values from the running example, the probability of the first two inputs to agree and the expected number of inputs required are:

$P(\text{`}S_{I,U1}(T) = S_{I,U2}(T)\text{'}) = 0.6162$ and thus $Work_{V2}(T) = 2.3838$

Compared to the reduction in error, the overhead over the base case is relatively low. The reduction in effort, as compared to 3-Redundancy, is 21%, corresponding to a 26% increase in throughput, at no increase of the probability of errors at all. ■

Discussion. Note that in reality both P('miss') and P('add') will be far lower than the pessimistic values from our example computations. Further, the probability of a correct original state $P(\text{`}S_O(D)=S_C(D)\text{'})$ is often higher, resulting in a higher probability of the first two inputs to agree $P(\text{`}S_{I,U1}(D)=S_{I,U2}(D)\text{'})$. On the other hand, tasks can comprise far more decisions, so the exponent in the computation of $P(\text{`}S_{I,U1}(T) = S_{I,U2}(T)\text{'})$ increases, resulting in lower values. Depending on the actual numbers, the effect can go either way:

Example 4. If $P(\text{`}S_{I,U1}(D)=S_{I,U2}(D)\text{'})$ is 99% in a task with 20 decisions, $P(\text{`}S_{I,U1}(T)=S_{I,U2}(T)\text{'})$ is 82%; in a task with 50 decisions in turn, it drops to 61%. ■

4.3 Vote Boosting

Vote Boosting ('VB') increases the weight of inputs of users who make few mistakes. It exploits that presumably not all users make mistakes with the same probability, and that v-Voting allows to observe the frequency of mistakes for each user U. If U has made few mistakes recently, Vote Boosting gives higher weight to an input from U in the aggregation function. Thus, it reduces the number of inputs required to compute a result.

Definition: CoinFlip(c) is a random function that returns 1 with a probability of c and 0 with a probability of (1-c). □

Notation: BoostProb(U,T) is the function that computes the probability that the input $S_{I,U}(T)$ of a user U for a task T receives a vote boost (referred to as the **boost probability** in the following). □

We derive a formula for this probability below.

Definition: $Result_{VB}(Input(T))$ is the input-aggregation function for Vote Boosting, as follows:

$$Result_{VB}(Input(T)) := \begin{cases} S_{I,U}(T) & \text{if } |Input(T)| = 1 \text{ and } CoinFlip(BoostProb(U,T)) = 1 \\ Result_{V}(Input(T)) & \text{otherwise} \end{cases}$$ □

With this definition of $Result_{VB}(Input(T))$, with a probability of BoostProb(U,T), $S_{I,U}(T)$ immediately becomes the result of T, bypassing the v-Voting mechanism. This reduces $Work_{VB}(T)$ to 1, the baseline level, completely eliminating the overhead. However, it also abandons the error-prevention functionality of v-Voting. Thus, BoostProb(U,T) may return a boost probability considerably above 0 only for users who are very unlikely to make mistakes. We formalize BoostProb(U,T) as follows:

Notation: C denotes the minimum probability required for the result of a task T to be correct, i.e., the required result quality. $P(\text{`}S_{I,U}(D)=S_C(D)\text{'})$ is the probability that a user U contributes a correct input to a decision D, the respective probability for a task T is $P(\text{`}S_{I,U}(T)=S_C(T)\text{'}) = P(\text{`}S_{I,U}(D)=S_C(D)\text{'})^{|T|}$. Further, **Correct(U)** denotes the observed number of correct inputs from user U since his last erroneous input. **m**

denotes the maximum probability the system accepts for user U to receive a vote boost for a task T even though actually P('$S_{I,U}(T)=S_C(T)$') < C. □

The actual value of P('$S_{I,U}(T)=S_C(T)$') is unknown, but we can estimate it with high certainty from Correct(U). In particular, we can compute BoostProb(U,T) by means of a significance test for accepting the hypothesis **"P('$S_{I,U}(T)=S_C(T)$') ≥ C"** based on Correct(U) correct observed inputs. This hypothesis states that user U has a sufficiently high probability of providing correct input for T to be eligible for a vote boost. We derive an upper bound b for BoostProb(U,T) from a significance test, namely the highest value b for which the hypothesis is true at a significance level of m / b:

$$P(\text{"}P(\text{'}S_{I,U}(T) = S_C(T)\text{'} < C\text{"}) \le m/b \Leftrightarrow C^{\frac{Correct(U)}{|T|}} \le m/b \Leftrightarrow \frac{m}{C^{\frac{Correct(U)}{|T|}}} \ge b$$

Note that b increases exponentially with Correct(U) / |T|. To prevent voting to be completely deactivated for any user (i.e., his boost probability rises to 1), we use (1 - m) as an additional upper bound. Further, we want BoostProb(U,T) to be 0 for Correct(U) = 0 and therefore subtract m.

Definition:

$$BoostProb(U,T) := \min\left((1-m), m\cdot(C^{\frac{-Correct(U)}{|T|}} - 1)\right)$$ □

Example 5. This example illustrates how the boost probability increases as a user gives more and more correct inputs: Suppose that a given task T consists of 3 decisions. Further, suppose user U has contributed a correct input to the previous Correct(U) = 100 decisions. Finally, let m = 1%, and C = 99%. Then the probability of boosting the vote of U is:

$$BoostProb(U,T) = 0.01\cdot(0.99^{\frac{-100}{3}} - 1) = 0.4\%$$

For the boost probability to exceed 50% for the given T, m, and C, Correct(U) has to exceed 1173. This means that U has to contribute inputs to 391 tasks without any mistake for this to happen. After increasing Correct(U) to 1374, i.e., after 458 tasks the size of T, the boost probability finally reaches its upper limit of (1 - m) = 99%. ■

5 Evaluation

To evaluate our mechanisms, we have run extensive simulations. We have tested many variations of input-aggregation functions.

5.1 Experimental Setup

The **sets of tasks** used here have two parameters: the number of options per decision, and the accuracy of the initial states. We generated 9 sets of 1,000,000 tasks, with 2, 3, or 4 options per decision and 80%, 90%, and 95% as the accuracy for the original states. Each task consists of 5 to 10 decisions, normally distributed over that interval.

The **user populations** tested have two parameters: their mean probabilities of cheating and of mistaking. We used values of 1%, 4%, and 15% for both, generating populations of 1000 users for each of the resulting 9 combinations. For the individual users, the probabilities of cheating and of making errors by mistake were exponentially distributed over [0,1] around the respective mean values.

We have implemented users as follows: In case of an add error on a decision with more than two options, a user selects one of the erroneous options at random. Users take a fixed time t per decision when contributing thoughtfully. Changing the state of a decision increases this time to 2·t. Cheating decreases it to t/2. At runtime, each user is a separate thread, so users are independent of each other and work concurrently.

In all, we ran simulations for 46 input-aggregation functions: One is the base case, i.e., each task receives one input. The other 45 are as follows: r-Redundancy with r = 3,5,7, v-Voting with v = 2,3,4, combined with 14 different parameter combinations for Vote Boosting, one being to deactivate it.

5.2 Results

From a total of 14,661 simulated scenarios, we report only on the four analyses we deem the most interesting, to save space.

v-Voting vs. r-Redundancy. Table 1 shows the average result quality and the average number of answers per task for v-Voting and r-Redundancy. For fairness, the numbers for v-Voting exclusively come from input-aggregation functions that do not use Vote Boosting. All numbers are aggregated over all user populations and task sets. Clearly, v-Voting yields better throughput than r-Redundancy. This substantiates the results of the analysis. Interestingly, result quality also improves slightly with 2-Voting and 3-Voting in comparison to 3-Redundancy and 5-Redundancy, respectively. We figure that this is because v-Voting avoids ambiguous decisions.

Table 1. Inputs per task and remaining error

	Base Case	3-Red.	2-Voting	5-Red.	3-Voting	7-Red.	4-Voting
Remaining Error (in %)	4,25	1,11	1,01	0,48	0,46	0,27	0,27
Inputs per Task	1	3	2,36	5	3,57	7	4,75

Vote Boosting. Figure 1 visualizes the impact of Vote Boosting, namely the increase in throughput and in errors. The effect of changes to C and m is similar for all three values of v we have tested: The more liberal the parameter settings, the higher the increase in throughput, but also the number of errors. The dependency seems almost linear for both. For a given result quality required, this predictable behavior allows for tuning to achieve the highest throughput possible.

Cost of High-Quality Results. Table 2 shows the average number of inputs required for each task to achieve at least 99.5% accuracy in the result, broken up across the 9 different user populations. The accuracy actually achieved is given in brackets, with the parameters of the input-aggregation function listed beneath. The input-aggregation function always uses v-Voting (parameter v), mostly with Vote Boosting (parameters m and C). A value of 0 for m indicates that Vote Boosting has

not been used. These results point out the correlation between the capability and honesty of contributing users and crowdsourcing throughput; the latter translates directly into the per-task cost in scenarios with a per-input payoff, e.g., the Amazon Mechanical Turk. With low probabilities for both mistakes and dishonesty, 1.14 inputs per task are sufficient to achieve the desired accuracy. This number increases sharply if either of the two probabilities increases. With pessimistic values for both, even 5.38 inputs per task are not enough to reach the goal. This highlights the importance both of fostering high-quality inputs and of deterring users from cheating.

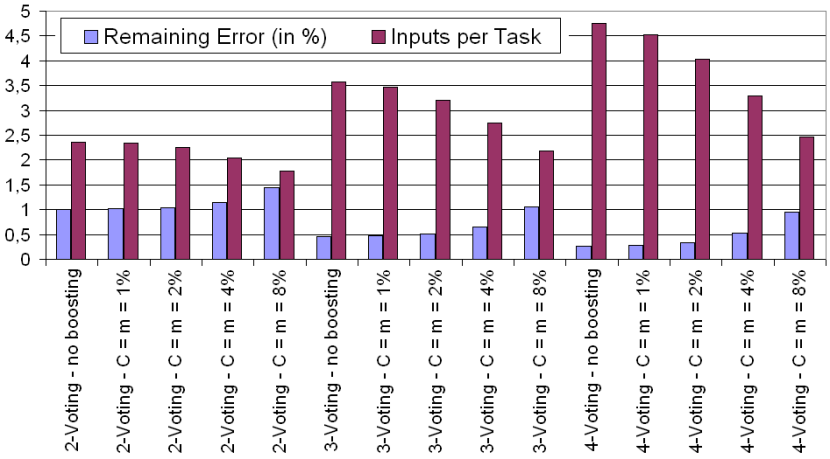

Figure 1. Effects of Vote Boosting

Crowdsourcing Strategy. As our simulations have shown, the best-suited strategy to achieve a desired result quality at a high throughput depends on the exogenous parameters. These parameters are hardly predictable at the start of a crowdsourcing project. Thus, we recommend starting out on pessimistic assumptions, i.e., favoring result quality over throughput. Then, experts can assess the quality achieved (e.g., from a sample of task results) and deduce values of the exogenous parameters. Afterwards, the endogenous parameters can be adjusted to optimize throughput.

Table 2. Inputs required to achieve 99.5% result accuracy

Mean Prob. of	Mistakes		
Cheating	**1%**	**4%**	**15%**
1%	**1.14** (99.51%) v=2 m=8% C=92%	**1.78** (99.63%) v=2 m=4% C=96%	**3.78** (99.55%) v=3 m= 2% C=98%
4%	**1.42** (99.57%) v=2 m=4% C=96%	**1.93** (99.51%) v=2 m=4% C=96%	**4.48** (99.51%) v=4 m=4% C=96%
15%	**3.94** (99.65%) v=4 m=2% C=98%	**4.6** (99.61%) v=4 m=2% C=98%	**not achieved** 5.38 (98.62%) v=4 m=0

6 Conclusions

Crowdsourcing is popular for large-scale data processing endeavors that require human input. However, both potential inability and dishonesty of users threaten the quality of the results. This causes a tradeoff between data throughput and result quality.

In this paper, we have studied mechanisms that enforce data quality with an impact on throughput as small as possible, independent of the actual tasks. In particular, **v-Voting** increases throughput over static redundancy based approaches. **Vote Boosting** further increases throughput by capitalizing on especially capable users.

Extensive simulations over a wide range of exogenous parameters have confirmed the suitability of the mechanisms, substantiating our findings from theoretical analyses. In particular, simulation results show (1) that v-Voting yields higher result quality than r-Redundancy with fewer inputs per task, and (2) that Vote Boosting allows trading off result quality in favor of throughput in a predictable fashion.

References

1. The Amazon Mechanical Turk, http://www.mturk.com
2. Cooper, S., Khatib, F., Treuille, A., Barbero, J., Lee, J., Beenen, M., Leaver-Fay, A., Baker, D., Popovic, Z. Predicting protein structures with a multiplayer online game. Nature 466, 2010.
3. Eckert, K., Niepert, M., Niemann, C., Buckner, C., Allen, C., Stuckenschmidt, H.: Crowdsourcing the assembly of concept hierarchies. In: Proceedings of JCDL 2010, Brisbane, Australia, 2010.
4. Lintott, C. J., Schawinski, K., Slosar, A., Land, K., Bamford, S., Thomas, D., Raddick, M. J., Nichol, R. C., Szalay, A., Andreescu, D., Murray, P. and Vandenberg, J. Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. Monthly Notices of the Royal Astronomical Society, 389, 2008. doi: 10.1111/j.1365-2966.2008.13689.x
5. Newby, G. B., Franks, C. Distributed proofreading. In Proceedings of JCDL 2003. Houston, TX, USA, 2003. doi: 10.1109/JCDL.2003.1204888
6. Sautter, G., Agosti, D., Böhm, K., Klingenberg, C. Creating Digital Resources from Legacy Documents - an Experience Report from the Biosystematics Domain, in Proceedings of ESWC, Heraklion, Greece, 2009.
7. Siorpaes, K., Hepp, M. OntoGame: towards overcoming the incentive bottleneck in ontology building. In Proceedings OTM 2007, Vilamoura, Portugal, 2007.
8. Snow, R., O'Connor, B., Jurafsky, D., Ng, A. Y. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In EMNLP 2008, Morristown, NJ, USA, 2008.
9. Von Ahn, L., Blum, M., Hopper, N., Langford, J. CAPTCHA: Using Hard AI Problems for Security. Advances in Cryptology - EUROCRYPT 2003. Springer Berlin / Heidelberg, 2003. doi: 10.1007/3-540-39200-9_18
10. Von Ahn, L., Games with a Purpose. IEEE Computer, 2006. 29(6): p. 92-94.
11. Von Ahn, L., Maurer, B., McMillen, C., Abraham, D., Blum, M. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. Science 321 (5895), 2008. doi:10.1126/science.1160379