High-throughput crowdsourcing mechanisms for complex tasks

Guido Sautter · Klemens Böhm

Abstract Crowdsourcing has been identified as a way to facilitate large-scale data processing that requires human input. However, working with a large anonymous user community also poses 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 research on crowdsourcing mostly focuses on result quality and pays little attention to throughput or even to the tradeoff between the two. One reason is that the number of tasks (atomic units of work) is usually small. A further problem is that the tasks themselves are small as well. In consequence, existing result quality-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 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 demonstrate the effectiveness of the proposed mechanisms.

Keywords Crowdsourcing · Data quality · Throughput

1 Introduction

Recently, crowdsourcing has been identified as a way to work off large numbers of tasks that require human input to improve data quality. Crowdsourcing distributes small pieces of large efforts to many users who make small contributions to the solution. This typically happens over the Internet. Crowdsourcing has been used successfully for many applications, e.g., double-keying individual words for OCR correction (von Ahn et al. 2008), image labeling (Lintott et al. 2008; von Ahn 2006), word-sense disambiguation (Snow et al. 2008), or grading the relatedness of word pairs for ontology construction (Eckert et al. 2010; Siorpaes and Hepp 2007). Crowdsourcing raises a number of unique challenges. In particular, there is no guarantee that user inputs are correct. Here, an input is correct if it is identical to the result respective experts would agree on (Eckert et al. 2010). We see several possible reasons for incorrect inputs, namely:

– Users can accidentally make mistakes due to misjudgment or sloppiness. This may happen even if users contribute solely because of interest in the project, like in Cooper et al. (2010), Lintott et al. (2008).

– Particularly when users receive some reward for their input, they might cheat to reduce their effort. This means that they might contribute random input instead of working thoughtfully. Especially if the reward is external, e.g., monetary, like in Eckert et al. (2010), Snow et al. (2008), and von Ahn et al. (2008), gathering the reward might well be the only motivation to contribute. Eckert et al. (2010) have observed users following this strategy.

– Out of vandalism or other malicious motives, users might generate erroneous input on purpose. Because
this erroneous input exhibits the same properties as that arising from mistakes, we do not consider it separately in our model.

To ease presentation and to increase clarity, we introduce several notions: A Task $T$ is a unit of work assigned to contributing users. Each task consists of one or more Decisions $D_1, \ldots, D_d$ the user has to take. In von Ahn et al. (2008), for example, each task consists of two decisions, namely on the correct transcriptions of two words from images. In Eckert et al. (2010) in turn, tasks consist of 12 decisions, namely on the relatedness of 12 term pairs. The original state of a task is its status before users have worked on it. Further, the final status of a task is its result, i.e., when a crowdsourcing system regards the task as completed. Finally, inputs are the contributions individual users make when working on a task. We further formalize these notions in Sect. 2.

In our context, it is important that incorrect inputs exhibit different properties depending on their reason and thus require different countermeasures. The crowdsourcing projects mentioned before have developed different respective strategies. One measure to counter mistakes is redundancy, i.e., to obtain input from more than one user for each task, usually a lot more than one. Unfortunately, redundancy has a severe impact on throughput. A means to discourage cheating, in turn, is to probe users with tasks the system already knows the correct results for, e.g., CAPTCHAs (von Ahn et al. 2003).

The tasks crowdsourced in previous projects have been relatively simple, e.g., double-keying individual words (von Ahn et al. 2008), grading the relatedness of individual term pairs (Eckert et al. 2010), or generating meaningful labels for individual images (von Ahn 2006). So far, if tasks consist of more than one decision, like in Eckert et al. (2010) and Snow et al. (2008), the individual decisions are mutually independent and can be freely combined into tasks.

Tasks in other applications are a lot more complex, e.g., converting satellite images into maps. Another example is the generation of semantic markup for legacy documents. In general, tasks consist of several dependent decisions that belong together, or decisions are highly complex. In distributed proofreaders (Newby and Franks 2003) for example, decisions are the transcriptions of entire document pages, comprising both the word level and the structure of pages. Tasks of similar complexity arise in the Madagascar Project (Sautter et al. 2009). Because of the complex decisions and the high level of redundancy, throughput is relatively low in distributed proofreaders, around 18,000 documents in 10 years. A more promising approach would be to use a mechanism like reCAPTCHA (von Ahn et al. 2008) for the word-level transcription and to proofread the page structure separately (we argue). Even though structuring document pages can be broken into several decisions of lower complexity, these decisions still form an atomic 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 atomic tasks. Previous crowdsourcing projects have mostly addressed (1), but not in combination with (2) or (3). In addition, previous work has not addressed the tradeoff between data throughput and result quality.

To fill this gap, in paper studies data quality enforcement techniques that are independent of the nature of the tasks, counter both mistakes and cheating, and pay attention to throughput. In particular, we analyze three techniques: The first two increase throughput while maintaining the data quality of static redundancy approaches, the third one discourages cheating:

- **v-Voting** counters mistakes, generalizing a principle used in reCAPTCHA (von Ahn et al. 2008). For every task, it obtains input from several users and aggregates these inputs to an overall result. In contrast to static redundancy, it uses a voting mechanism (controlled by parameter ‘$v$’), which reduces the number of inputs required. A core assumption behind v-Voting is that mistakes are random, so aggregating individual inputs is very likely to yield the correct result.

- **Vote Boosting** extends v-Voting and further increases throughput. It increases the weight of inputs from users who are known to make few mistakes from prior observations, further reducing the number of inputs required. If there is a reward system, the reward can be specified to increase with the weight of the vote. We expect this to foster high-quality inputs.

- **Sampled Probing** counters cheating. It observes when tasks are completed and picks out and stores those whose results significantly differ from their original state. It confronts users with these tasks at random in order to probe them. The probed user passes if his input is more similar to the sampled result than to the original state of the task.

All these mechanisms assume that the majority of the users contribute useful inputs, an assumption common to all crowdsourcing projects. The rationale simply is that if most inputs are arbitrary, there is no chance of obtaining any meaningful data at all.

To assess the effectiveness of the three mechanisms we present in this paper, we have conducted a thorough evaluation, considering both mistakes and cheating. It comprises theoretical analyses of the expected throughput, result quality, and user reward, as well as extensive simulations. The evaluation shows that the three mechanisms
serve their respective purpose well: (1) v-Voting and Vote
Boosting yield the same result quality as static redundancy
with considerably fewer inputs per task, and (2) Sampled
Probing reduces the expected reward for cheating users so
much that cheating does not pay off; this is not the case
without probing.

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

Paper outline: Section 2 introduces basic notions
required for analyzing crowdsourcing mechanisms. Sec-
tion 3 reviews related work. Section 4 provides an in-depth
explanation and mathematical assessment of the data
quality enforcement mechanisms. Section 5 features sim-
ulation results. Sect. 6 concludes. This paper is an extended
version of Sautter and Böhmann (2011), adding in-depth
analyses of user motivation and Sampled Probing as a
means to discourage cheating.

2 Formal notions

To facilitate a mathematical analysis of crowdsourcing,
this section introduces the formal notions required.

2.1 Decisions, tasks and functions

Definition A Decision $D$ is an atomic parameter set by a
user.

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

Notation $\text{Opts}(D) := \{O_1, \ldots, O_n\}$ is the set of options
available for $D$. In addition, $N \notin \text{Opts}(D)$ denotes the null
option, which models the case that $D$ is undecided on so
far.

For instance, the options for a decision can be the
available classes for named entities or the paragraph types.
The null option then indicates that a paragraph or named
entity has not yet been assigned a type or class, respec-
tively. Note that $\text{Opts}(D)$ can be large. In particular, this is
the case when users have to type words into a text field,
like in (von Ahn 2006; von Ahn et al. 2008). $\text{Opts}(D)$ then
contains all strings of a certain length, and practically all
words of the language. $N$ is the empty string in this case.

At every point of its time of residence in the crowd-
sourcing system, a decision $D$ has an option $S(D) \in \text{Opts}(D) \cup \{N\}$ assigned to it. We refer to $S(D)$ as
the state of $D$. There are several dedicated states to be
distinguished:

Notation $S_0(D) \in \text{Opts}(D) \cup \{N\}$ is the original state of
$D$, i.e., the state of $D$ when it enters the crowdsourcing
system. $S_i(D, U) \in \text{Opts}(D)$ denotes the input state a user
$U$ has assigned to $D$, i.e., his input for $D$, which is the
option this user has selected. An input state cannot be
$N$. $S_D(D) \in \text{Opts}(D) \cup \{N\}$ is the result of $D$, i.e., the
state of $D$ when leaving the crowdsourcing system. A null result,
i.e., $S_D(D) = N$, indicates that the system could not deter-
mine a meaningful result for $D$. $S_C(D) \in \text{Opts}(D)$ is the
correct state of $D$, i.e., the result that respective experts
would agree on. $\text{Input}(D) = (S_1(D, U_1), \ldots, S_1(D, U_n))$ is the
input list of $D$, comprising the inputs that users $U_1, \ldots, U_n$
have contributed to $D$. $\text{Contribute}(D) = (U_1, \ldots, U_n)$ is the
contributor list of $D$, i.e., the list of all users who have
contributed an input to $D$, in the order they made their
contributions. $\text{Contribute}(D, O) = (U \in \text{Contribute}(D) | S_1(D, U) = O)$, with $O \in \text{Opts}(D)$, denotes the list of the
users whose input for $D$ is equal to a given option $O$.

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

Definition A Task $T = (D_1, \ldots, D_d)$ is the unit of work a
crowdsourcing system assigns to users, consisting of one or
more decisions $D_1, \ldots, D_d$.

The individual decisions that make up a task can be
either connected or independent. In the former case, tasks
are fixed and atomic, and a crowdsourcing system cannot
add or remove decisions. In the latter case, the system can
freely combine 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 $S(T) = (S(D_1), \ldots, S(D_d))$. In analogy to
individual decisions, we make the following distinctions:

Notation $S_0(T) = (S_0(D_1), \ldots, S_0(D_d))$ is the original
state of $T$. $S_i(T, U) = (S_i(D_1, U), \ldots, S_i(D_d, U))$ the input
state user $U$ has assigned to $T$, i.e., his input for $T$.

$S_D(T) = (S_D(D_1), \ldots, S_D(D_d))$ is the result of $T$, i.e., its
state after all user interactions. $S_C(T) = (S_C(D_1), \ldots,
S_C(D_d))$ is the correct state of $T$. $\text{Input}(T) = (S(T, U_1), \ldots,
S(T, U_n))$ is the input list of $T$, comprising the inputs that
users $U_1, \ldots, U_n$ have contributed to $T$.

Definition The distance $\text{Dist}(S_1(T), S_2(T))$ of two states
$S_1(T), S_2(T)$ is the number of decisions for which they differ,
formally $\text{Dist}(S_1(T), S_2(T)) := |\{D \in T | S_1(D) \neq S_2(D)\}|$.

Notation The probability $P(S_3(T) = S_4(T'))$ of two
arbitrary, but distinct states $S_3(T)$ and $S_4(T)$ of a task $T$ to
be equal is the product of the probabilities of the respective
states of the individual decisions \( D \in T \) to be equal, formally:

\[ P\left( S_X(T) = S_Y(T)' \right) = \prod_{D \in T} P\left( S_X(D) = S_Y(D)' \right) \]

To ease presentation, we assume that \( P\left( S_X(D) = S_Y(D)' \right) \) has the same value for all \( D \in T \). Thus, we get a simpler formula:

\[ P\left( S_X(T) = S_Y(T)' \right) = P\left( S_X(D) = S_Y(D)' \right), \quad D \in T \] \[ P\left( S_X(T) = S_Y(T)' \right) \]

Note that this assumption does not incur any loss of generality. This is because we do not make any further assumptions regarding the nature of the individual decisions a task \( T \) consists of.

**Definition** An abstract input-aggregation function \( \text{Result}(D) \) is a function of type \( \text{Input}(D) \rightarrow \{N, S_R(D)\} \) that computes the result of a decision \( D \) from \( \text{Input}(D) \). Analogously, \( \text{Result}(T) \) is the respective function of type \( \text{Input}(T) \rightarrow \{\emptyset, S_R(T)\} \) for a task \( T = (D_1, \ldots, D_d) \). Unless specified otherwise, \( \text{Result}(T) \) is as follows:

\[
\text{Result}(T) := \left\{ \begin{array}{ll}
\emptyset & \text{if } \exists D \in T : \text{Result}(D) = N \\
(\text{Result}(D_1), \ldots, \text{Result}(D_d)) & \text{otherwise}
\end{array} \right.
\]

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

**Example 1** A very simple example of a concrete input-aggregation function is the following: \( \text{Result}_1(D) := N \) if \( \lvert \text{Input}(D) \rvert = 0 \), \( S_1(D, U) \) otherwise. This function defines the result of \( D \) as the first input some user \( U \) contributes for \( D \). A more complex example is \( \text{Result}_2(D) := S_1(D, U) \) if \( \lvert \text{Input}(D) \rvert = 2 \) and \( S_1(D, U_1) = S_1(D, U_2) \), \( N \) otherwise; this function defines the result of a decision as empty/undefined unless there are exactly two agreeing inputs from two users \( U_1 \) and \( U_2 \).

Other concrete input-aggregation functions are the ones used for \( r \)-Redundancy (cf. Sect. 3.1), \( v \)-Voting (cf. Sect. 4.3), and Vote Boosting (cf. Sect. 4.4).

**Definition** The abstract reward function \( \text{Pay}(U, T) \) is a function of type \( U \times T \rightarrow R \) that computes the payoff user \( U \) gets for contributing input to task \( T \).

\( \text{Pay}(U, T) \) facilitates modeling scenarios that involve a reward system. Situations without any reward correspond to a reward function that always returns 0. Note that the number of possible concrete reward functions is unlimited in theory. In consequence, specific functions can have a multitude of additional configuration parameters that are independent of \( U \) and \( T \). To keep the presentation simple, we do not explicitly model such parameters.

**Notation** \( \text{Pay}(D) \) and \( \text{Pay}(T) \) denote the overall payoff a reward system offers for a decision \( D \) and task \( T \), respectively.

Both \( \text{Pay}(D) \) and \( \text{Pay}(T) \) can be fixed values, or they can depend on \( D \) or \( T \), respectively. \( \text{Pay}(D) \) and \( \text{Pay}(T) \) are closely related:

**Observation:** The overall payoff for a task \( T \) is the sum of the overall payoffs for the decisions \( T \) consists of, formally: \( \text{Pay}(T) := \sum_{D \in T} \text{Pay}(D) \)

**Notation** \( \text{WorkExp}(U, T) \) denotes the payoff user \( U \) can expect to receive for contributing an input to a task \( T \). \( \text{WorkExp}(T) \) denotes the expected value of \( \lvert \text{Input}(T) \rvert \) at the moment the input-aggregation function returns a non-null result.

In other words, \( \text{WorkExp}(T) \) is the expected number of inputs to obtain until \( S_R(T) \) emerges.

2.2 Types of errors

This section investigates which errors can occur in the inputs that users contribute to crowdsourced tasks and in the task results. Note that it is not our goal to enable crowdsourcing systems to distinguish between the reasons of errors described in the following. In general, this is not possible. This is because the observation of an error typically does not reveal anything about 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 reasons discussed below, there are two types of errors: (1) Miss errors are errors that remain uncorrected; formally, a miss error exists if \( S_0(D) \neq S_C(D) \), and \( S_R(D) \neq S_C(D) \). (2) Added errors are errors introduced by users; such an error exists if \( S_0(D) = S_C(D) \), and \( S_R(D) \neq S_C(D) \).

2.2.1 Accidental errors

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 \( P(\text{‘accidental miss’}) \) is the average probability across all users that some user accidentally misses an error in a decision \( D \) of a task \( T \). \( P(\text{‘accidental add’}) \) is the average probability that some user accidentally adds an error in a decision \( D \) of a task \( T \).

2.2.2 Cheating errors

Cheating errors occur because users do not bother to contribute thoughtful input. If the original state of a task \( S_0(T) \) is a valid input, we assume that cheating users simply submit \( S_0(T) \) as their input because this is the least possible effort. If the original state of a task consists of null values, like the initially empty text fields in (von Ahn 2006; von Ahn et al. 2008), we assume cheating users to randomly select an element of \( \text{Opts}(D) \) as their input. In the former case, adding an error requires making a change to the original state of a task. Thus, submitting the original state as an input without changing anything does not introduce any error. On the other hand, if the original state of a task contains errors, and modifications are necessary to correct them, they remain part of the input of a cheating user and thus become miss errors. In consequence, cheating errors are always miss errors if the original state of a task is non-empty; a correct original state of a task, in turn, always results in a correct input if a user cheats.

Notation \( P(\text{‘cheat’}) \) is the average probability that some user cheats on a task \( T \) and thereby contributes an input that possibly contains miss errors.

2.2.3 Combined error probability

To simplify subsequent computations, we aggregate the individual error probabilities into universal error probabilities.

Notation/observation \( P(\text{‘miss’}) \) is the average probability of a miss error to occur in a single input, namely:

\[
P(\text{‘miss’}) = (1 - P(\text{‘cheat’})) \times P(\text{‘accidental miss’}) + P(\text{‘cheat’})
\]

\( P(\text{‘add’}) \) is the average probability of an add error to occur in a single input, namely:

\[
P(\text{‘add’}) = (1 - P(\text{‘cheat’})) \times P(\text{‘accidental add’})
\]

2.3 Parameters and 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 making accidental errors and cheating on tasks.

The sole endogenous parameter is the input-aggregation function in use and its parameterization. The payoff function does not have any direct impact on the accuracy of task results, and therefore we do not look at it as an endogenous parameter.

The numbers to optimize are: (1) the expected accuracy of task results, which corresponds to the probability that the result of a task is correct, and (2) the expected number of inputs required to achieve this accuracy, i.e., the expected value of \( \text{Input}(T) \). The latter is particularly important when using third-party crowdsourcing platforms that require a fixed monetary reward per input, like the Amazon Mechanical Turk (AMT). In such a setting, the expected number of inputs per task is proportional to the expected cost.

3 Related work

This section discusses recent crowdsourcing projects, the mechanisms they have deployed to enforce data quality, and the experiences they have gathered.

3.1 \( r \)-Redundancy

Many previous crowdsourcing projects (Eckert et al. 2010; Lintott et al. 2008; Snow et al. 2008) use a rather simple redundancy-based input-aggregation function. We refer to this function as \( r \)-Redundancy, where \( r \) is the parameter that specifies the number of inputs required per task. \( r \)-Redundancy means that, once \( r \) users have contributed an input to a task \( T \), the most frequently given input becomes the result of \( D \), for each Decision \( D \) in \( T \). \( r \) usually is an odd number. In general, \( r \)-Redundancy is suboptimal with regard to throughput. This is because always Work-Exp\( r \)\((T) = r \) (the subscript ‘\( R \)’ meaning ‘\( r \)-Redundancy’), i.e., a task always takes \( r \) inputs to complete, even if the first \((r + 1)/2\) inputs agree in all decisions, and the last \((r - 1)/2\) inputs do not have any influence on the result.

Notation Result\( r \)\((D) \) is the input-aggregation function for \( r \)-Redundancy, formally:

\[
\text{Result}_r(D) = \begin{cases} 
O_0 \in \text{Opts}(D) \text{ such that } & \text{if } |\text{Input}(D)| = r \\
|\text{Contribute}(D, O_0)| \text{ is maximal} & \text{otherwise} \\
N & 
\end{cases}
\]

Observation: even if \( r \) is odd, results can be ambiguous with \( r \)-Redundancy: if there are more than two options for a
decision $D$, i.e., $|\text{Opts}(D)| > 2$, it can happen that more than one option occurs in a simple (non-absolute) majority of inputs:

**Example 2** suppose that $\text{Opts}(D) = \{O_1, O_2, O_3\}$ and $\text{Input}(D) = \{(O_1), (O_2), (O_3), (O_1)\}$ in a 5-Redundancy scenario. Both $O_1$ and $O_2$ occur in 2 inputs. In principle, the input-aggregation function can resolve such cases in several ways: (1) pick the option that occurred first ($O_1$ in this case), (2) pick the option that has achieved the majority first ($O_2$ in this case), \(^1\) or (3) pick one of the two at random. Note that in neither case the result has an absolute majority.

Eckert et al. (2010) use a 5-redundant approach to arrange terms into a concept hierarchy. Each task consists of 12 independent decisions. Each decision is to compare a pair of terms with regard to relatedness and relative generality, i.e., which term is more specific or more general than the other one. To detect cheating, each task includes two dedicated decisions $P$ and $Q$. Namely, $P$ and $Q$ are term pairs for which users can easily determine relatedness and relative generality solely based on common sense. If users get them wrong, this serves as an indicator for them not paying attention. With this mechanism, Eckert et al. (2010) have achieved a degree of data quality comparable to that of a concept hierarchy domain experts have constructed from the same terms. 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. This is not possible with tasks that consist of connected decisions.

Snow et al. (2008) have successfully used 10-Redundancy-based crowdsourcing for detail level NLP tasks like word-sense disambiguation. All tasks consist of 30 independent decisions bundled randomly. The system does not include any mechanisms to detect or prevent cheating. The reported result quality is similar to the figures of Eckert et al. (2010), which suggests that cheating at least has not been pervasive. However, it is questionable if this still holds if users work on many tasks over an extended period of time; the more so as Eckert et al. (2010) have detected cheating attempts in considerable numbers. Thus, it is very unlikely to constantly obtain high-quality results in large crowdsourcing efforts without a mechanism that discourages cheating.

### 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 $S_3(D) = N$. If the two inputs agree, they count as correct, and both users get a reward.

The rationale is that random pairing prevents users from colluding, e.g., from agreeing on some fixed input a priori. The odds of two random inputs to agree is only 1 in $|\text{Opts}(D)|$. Thus, contributing thoughtful input is the only way of increasing the chance of reaching agreement and thus getting a reward larger than 1/|Opts(D)|. Von Ahn has successfully used this approach for image labeling (von Ahn 2006). Siorkpes and Hepp (2007) have shown with their OntoGame that it also works well for ontology construction and alignment, and for named entity disambiguation. Theoretically, the agreement approach also works for tasks that consist of multiple decisions. However, a single mistake of either user invalidates both inputs. Defining agreement in a per-decision fashion can alleviate this if the decisions are independent. The system can collect incomplete decisions and bundle them into new tasks until some pair of users agrees on an input. However, this does not work with tasks that consist of connected decisions. It is unclear how the approach could work in this case.

### 3.3 Other approaches

ReCAPTCHA (von Ahn et al. 2008) is a crowdsourcing project that double-keys images of document pages in a word-by-word fashion, building on the CAPTCHA mechanism (von Ahn et al. 2003). The CAPTCHAs users have to solve consist of two random word images. One of them is the crowdsourcing task $T$, consisting of 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 CAPTCHA is a word image the system already knows the correct transcription $S_C(C)$ for. ReCAPTCHA takes an input for $D$ into account only if the CAPTCHA is solved, i.e., $S_D(C) = S_C(C)$. A task is complete as soon as there are three agreeing inputs. The system has earlier obtained $S_C(C)$ through the same mechanism now used for $D$. This means that as soon as there is a result for $D$, it can serve in the same role as $C$ later on.

The original state for both $C$ and $D$ is empty. Requiring 3 agreeing inputs for $D$ renders mistakes highly unlikely: In practice, reCAPTCHA achieves a word-error rate well below 1%. The presence of the CAPTCHA $C$ that is indistinguishable from the actual task $T (=D)$ counters cheating well.

However, tasks in reCAPTCHA are very small, and users normally need only a few seconds to contribute their input. Tasks that take more time are impractical as CAPTCHAs because they would probably annoy many users. Thus, few web pages would integrate such a mechanism, so throughput would be too low. Furthermore, insisting on agreeing inputs is impractical if tasks consist of multiple decisions, as we will explain.

---

\(^1\) This is the mode we use in our evaluation.
Another crowdsourcing project related to the digitization of legacy documents is Distributed Proofreaders (Newby and Franks 2003). Its purpose is to correct OCR errors by means of repeated review, i.e., redundancy. Tasks consist of one very large decision, namely the transcript of an entire document page. Each user gets to see the changes of the users who worked on the task before him. Data throughput has been relatively low so far. Tens of thousands of volunteers have proofread around 18,000 works in roughly 8 years, as of July 2010. A more sophisticated process separating the pages into smaller chunks might be more promising, e.g., a process using reCAPTCHA on the word level.

The GalaxyZoo (Lintott et al. 2008) project has had over a million galaxy images classified into six basic categories by over 10,000 volunteers in less than 200 days. Their system presents randomly selected images to its users. However, it requires the whole set of tasks to be available from the start. Other efforts like the digitization of legacy documents is Distributed Proofreaders (Newby and Franks 2003). Its purpose is to correct OCR errors by means of legacy documents. Namely, it has to be consistent with the order of the decisions themselves appearing in \( T \).

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

\[
P_{BC}(\text{"add"}) = 5\%,
\]

\[
P_{BC}(\text{"miss"}) = 10\%,
\]

This means that we assume that the original state of a given decision \( D \) has an 80% chance of being correct, and that users cause miss errors and add errors with 10 and 5% probability when contributing an input for \( D \), respectively.

### 4.2 Base case

As the reference point for the individual data quality enforcement mechanisms, we first formalize the base case, i.e., a setting where exactly one user works on each task \( T \) and contributes a respective input, which immediately becomes the result of \( T \). Then, the probabilities \( P_{BC}(\text{"miss"}) \) of a miss error (‘BC’ is short for ‘Base Case’) and \( P_{BC}(\text{"add"}) \) of an add error occurring in a decision \( D \) of a task \( T \) are

\[
P_{BC}(\text{"miss"}) = P(\text{"miss"}),
\]

\[
P_{BC}(\text{"add"}) = P(\text{"add"}).
\]

This results in the following probability of a correct result:

\[
P_{BC}(\text{"add"}) - P_{BC}(\text{"miss"}) = P(\text{"add"}) - P(\text{"miss"})
\]

This means the following: There are two paths to an incorrect result for a decision \( D \): either original state of \( D \) is correct and the contributing user falsifies it (incurs an add error), or the original state of \( D \) is erroneous and the contributing user fails to correct it (incurs a miss error). If neither of this happens, the result of \( D \) is correct.

Note that always \( \text{WorkExp} = 1 \), representing optimal throughput. With the values from the running example, we obtain

\[
P_{BC}(\text{"add"}) = 0.94 \text{ and } P_{BC}(\text{"miss"}) = 0.06.
\]

### 4.3 v-Voting

v-Voting is a mechanism countering accidental errors. Like r-Redundancy, it does so by obtaining and aggregating
several inputs for each task. As opposed to $r$-Redundancy, however, it uses an agreement-based input-aggregation function, controlled by the so-called vote-majority parameter $v$. That is, there is a fixed level of agreement to reach, but no fixed number of inputs to obtain. Von Ahn et al. (2008) use 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** $\text{Result}_v(T)$ (the subscript ‘$v$’ indicating the use of ‘$v$-Voting’) is the input-aggregation function for $v$-Voting. Formally, this is:

\[
\text{Result}_v(T) := \begin{cases} 
O_0 & \text{if } \exists O_0 \in \text{Opts}(D) \text{ such that } \text{Contribute}(D, O_0) \geq v \\
N & \text{otherwise}
\end{cases}
\]

Note that $\text{Result}_v(T)$ avoids the ambiguous cases that can occur with $r$-Redundancy, as discussed in Sect. 3.1. Another advantage of $\text{Result}_v(T)$ is that it requires fewer inputs than $r$-Redundancy for the same expected result quality. Namely, tasks of low difficulty require fewer inputs because user inputs easily agree. Further note that $\text{Result}_v(T)$ computes the result decision-wise and does not require entire inputs to agree, in contrast to von Ahn et al. (2008).

**Example 3** 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:

\[
\begin{align*}
S_1(T, U_1) &= \text{('page header', 'main text', 'main text', 'footnote')} \\
S_1(T, U_2) &= \text{('main text', 'main text', 'main text')} \\
S_1(T, U_3) &= \text{('page header', 'main text', 'caption', 'main text')}
\end{align*}
\]

Even though no two inputs are equal as a whole, and all deviate from $S_c(T)$ in one decision, at least two inputs agree on 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 has actually provided this input. Had users $U_1$ and $U_2$ given the same overall input, the system would not have obtained an input for $T$ from $U_3$ at all.

Observation: decision-wise voting can considerably decrease the number of inputs required for a consensus result, as illustrated in the example. The larger the number of decisions a given task consists of, the higher the advantage.

**Formal analysis:** what is the overall probability of a correct result for a task $T$, i.e., $P_V(S_R(T) = S_c(T))$? We compute this in the following. For ease of presentation, we assume that $v = 2$. To keep the computation simple, we further assume the worst case, that is, 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 is the case only for binary decisions (i.e., $|\text{Opts}(D)| = 2$). In non-binary decisions like the classification task from the running example, it is an assumption that increases the error probability. However, it also ensures that $|\text{Input}(T)| \leq 3$. Namely, with $|\text{Opts}(D)| = 2$, a decision is reached after at most three inputs. This helps us because it reduces the number of cases to consider in our computations. Our simulation results show that the average value of $|\text{Input}(T)|$ barely increases for $|\text{Opts}(D)| > 2$, in the range of a few percent, over a wide range of values for the other exogenous parameters. Thus, the throughput according to our model will not differ much from the actual one.

**Notation** $P_V(\text{‘miss’})$ and $P_V(\text{‘add’})$ denote the probabilities of a miss error and an add error occurring in the result of a decision $D \in T$, respectively.

Informally, if $v = 2$, an error in the result of a decision $D$ occurs if the first two inputs are erroneous, and also if one of the two first and the third input are erroneous.

Formally, $P_V(\text{‘miss’})$ and $P_V(\text{‘add’})$ are as follows:

\[
\begin{align*}
P_V(\text{‘miss’}) &= 3 \times P(\text{‘miss’})^2 - 2 \times P(\text{‘miss’})^3 \\
P_V(\text{‘add’}) &= 3 \times P(\text{‘add’})^2 - 2 \times P(\text{‘add’})^3
\end{align*}
\]

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

\[
P_V(S_R(D) = S_c(D)) = 1 - P(\text{‘miss’})^2 \times P_V(\text{‘add’})
\]

The overall probability for the result of a task $T$ with $d$ decisions $D_1\ldots D_d$ to be correct then is:

\[
P_V(S_R(T) = S_c(T)) = P_V(S_R(D) = S_c(D), D \in T)^d
\]

**Example 4** 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:

\[
P_V(S_R(D) = S_c(D)) = 0.9886 \text{ and } P_V(S_R(T) = S_c(T), D \in T) \approx 0.9552
\]

In the base case, things are different.

\[
P_{bc}(S_R(D) = S_c(D)) = 0.94 \text{ and } P_{bc}(S_R(T) = S_c(T), D \in T) = 0.7807.
\]

With no correction at all, i.e., with fully automated NLP and no user interaction, it 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 95.5 % from about 78 % in the base case. This corresponds to a reduction of error by a factor of about 4, for the at most threefold effort.

Note that accuracy, for instance that of classification algorithms, is usually measured for individual objects, which 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 about 94 % to about 99 %. This corresponds to a reduction of error by a factor of almost 6 in comparison to the base case.

\( \text{Discussion: note that in reality both } P(\text{‘miss’}) \text{ and } P(\text{‘add’}) \text{ will be far lower than the rather pessimistic values from the example computations. Further, the probability of a correct original state } P(S_0(D) = S_c(D)) \text{ is often higher, resulting in a higher probability of the first two inputs to agree, i.e., a higher } P(S_0(D, U_1) = S_0(D, U_2)) \text{. On the other hand, tasks can consist of far more decisions, so the exponent in the computation of } P(S(T, U_1) = S(T, U_2)) \text{ increases, resulting in lower values. Depending on the actual numbers, the effect can go either way:} \\
\)

**Example 6** if \( P(S_0(D, U_1) = S_0(D, U_2)) \) is 99 % in a task with 20 decisions, \( P(S(T, U_1) = S(T, U_2)) \) is 82 %; in a task with 50 decisions in turn, it drops to 61 %.

User motivation: with an appropriately designed payoff function, v-Voting can also foster high-quality inputs. Suppose that the total payoff \( \text{Pay}(T) \) for each task \( T \) is shared between all users who have contributed inputs to \( T \).

**Definition** \( \text{Pay}_v(U, T) \) is the payoff function for v-Voting, namely:

\[
\text{Pay}_v(U, T) := \text{Pay}(T)/|\text{Input}(T)|
\]

Then the expected payoff a user \( U \) receives for contributing an input to \( T \) is:

\[
\text{PayExp}_v(U, T) = \text{Pay}(T)/\text{WorkExp}_v(T)
\]

\( \text{PayExp}_v(U, T) \) increases if the expected number of required inputs \( \text{WorkExp}_v(T) \) decreases. Thus, such a payoff function incentivizes users to seek agreeing inputs, i.e., to increase \( P(S(T, U_1) = S(T, U_2)) \) of the first two inputs to agree on all decisions in \( T \), namely:

\[
\text{WorkExp}_v(T) = 2 \times P(S_1(T, U_1) = S_1(T, U_2)) + 3 \times P(S_1(T, U_1) \neq S_1(T, U_2))
\]

Further, \( \text{WorkExp}_v(T)/\text{WorkExp}_p(T) \) is the overhead 2-Voting incurs in comparison to the base case. Likewise, \( 1 - \text{WorkExp}_v(T)/\text{WorkExp}_p(T) \) is the reduction in effort 2-Voting yields in comparison to 3-Redundancy.

**Example 5** with the values from the running example, this is:

\[
P(S_{1,11}(T) = S_{1,12}(T)) = 0.6162 \text{ and thus} \\
\text{WorkExp}_v(T) = 2.3838
\]

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 \( P(\text{‘miss’}) \text{ and } P(\text{‘add’}) \) will be far lower than the rather pessimistic values from the example computations. Further, the probability of a correct original state \( P(S_0(D) = S_c(D)) \) is often higher, resulting in a higher probability of the first two inputs to agree, i.e., a higher \( P(S_0(D, U_1) = S_0(D, U_2)) \). On the other hand, tasks can consist of far more decisions, so the exponent in the computation of \( P(S(T, U_1) = S(T, U_2)) \) increases, resulting in lower values. Depending on the actual numbers, the effect can go either way:**

**Example 6** if \( P(S_0(D, U_1) = S_0(D, U_2)) \) is 99 % in a task with 20 decisions, \( P(S(T, U_1) = S(T, U_2)) \) is 82 %; in a task with 50 decisions in turn, it drops to 61 %.

User motivation: with an appropriately designed payoff function, v-Voting can also foster high-quality inputs. Suppose that the total payoff \( \text{Pay}(T) \) for each task \( T \) is shared between all users who have contributed inputs to \( T \).

**Definition** \( \text{Pay}_v(U, T) \) is the payoff function for v-Voting, namely:

\[
\text{Pay}_v(U, T) := \text{Pay}(T)/|\text{Input}(T)|
\]

Then the expected payoff a user \( U \) receives for contributing an input to \( T \) is:

\[
\text{PayExp}_v(U, T) = \text{Pay}(T)/\text{WorkExp}_v(T)
\]

\( \text{PayExp}_v(U, T) \) increases if the expected number of required inputs \( \text{WorkExp}_v(T) \) decreases. Thus, such a payoff function incentivizes users to seek agreeing inputs, i.e., to increase \( P(S(T, U_1) = S(T, U_2)) \) of the first two inputs to agree on all decisions in \( T \), namely:

\[
\text{WorkExp}_v(T) = 2 \times P(S_1(T, U_1) = S_1(T, U_2)) + 3 \times P(S_1(T, U_1) \neq S_1(T, U_2))
\]

Further, \( \text{WorkExp}_v(T)/\text{WorkExp}_p(T) \) is the overhead 2-Voting incurs in comparison to the base case. Likewise, \( 1 - \text{WorkExp}_v(T)/\text{WorkExp}_p(T) \) is the reduction in effort 2-Voting yields in comparison to 3-Redundancy.

**Example 5** with the values from the running example, this is:

\[
P(S_{1,11}(T) = S_{1,12}(T)) = 0.6162 \text{ and thus} \\
\text{WorkExp}_v(T) = 2.3838
\]

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.

---

**4.4 Vote Boosting**

Vote Boosting is a mechanism that aims at increasing throughput. It increases the weight of inputs from users who are known to make few mistakes. It exploits that likely not all users make mistakes with the same probability, and that v-Voting allows for observing the frequency of a user making mistakes. If \( U \) has made very few mistakes recently, Vote Boosting gives higher weight to an input from \( U \) in the aggregation function, referred to as vote boost. By doing so, it reduces the number of inputs required for computing a result and thus increases throughput. We formalize Vote Boosting in the following:
**Definition** CoinFlip($c$) is a random function that returns 1 with a probability of $c$ and 0 with a probability of $(1 - c)$.

**Notation** $\text{BoostProb}(U, T)$ is the function that computes the probability that the input $S_i(T, U)$ 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. Note that more sophisticated definitions of $\text{BoostProb}(U, T)$ than the one below are conceivable, with more configuration parameters. Again, we do not explicitly model these parameters here, to keep the presentation simple.

**Definition** $\text{Result}_{\text{VB}}(T)$ is the input-aggregation function for Vote Boosting (indicated by the subscript ‘VB’), as follows:

\[
\text{Result}_{\text{VB}}(T) = \begin{cases} 
S_i(T, U) & \text{if } |\text{Input}(T)| = 1 \text{ and } \\
\text{CoinFlip}(\text{BoostProb}(U, T)) = 1 & \text{otherwise}
\end{cases}
\]

With this definition of $\text{Result}_{\text{VB}}(\text{Input}(T))$, the weight of input $S_i(T, U)$ becomes $v$ with a probability of $\text{BoostProb}(U, T)$. Then $S_i(T, U)$ becomes the result of $T$. This circumvents the $v$-Voting mechanism and thus reduces $\text{WorkExp}_{\text{VB}}(T)$ to 1, the baseline level, completely eliminating the overhead. However, it also abandons the error-preventing effect of $v$-Voting. Thus, $\text{BoostProb}(U, T)$ should return a value considerably above 0 only for users who are very unlikely to make mistakes, or, conversely, for users who are very likely to provide an error-free input for $T$. We formalize $\text{BoostProb}(U, T)$ as follows:

**Notation** $\text{minCorrProb}$ denotes the minimum probability required for the result of a task $T$ to be correct.

$\text{minCorrProb}$ is an endogenous parameter. Its value is to be determined by the operators of a crowdsourcing system that implements Vote Boosting, based on the result accuracy they aim at. We devise a respective configuration strategy in our evaluation.

**Notation** $P(\frac{\text{‘}S_i(D, U) = S_c(D)\text{’}}{\text{‘}T\text{’}})$ is the probability that a user $U$ provides a correct input for a decision $D$. The respective probability for a task $T$ is $P(\frac{\text{‘}S_i(T, U) = S_c(T)\text{’}}{\text{‘}T\text{’}}) = P(\frac{\text{‘}S_i(D, U) = S_c(D)\text{’}}{\text{‘}D\text{’}})$.

The actual value of $P(\frac{\text{‘}S_i(T, U) = S_c(T)\text{’}}{\text{‘}T\text{’}})$ is unknown. We thus estimate it from the number of decisions and tasks a given user has provided correct inputs for since last making a mistake. To achieve this, we test the hypothesis “$P(\frac{\text{‘}S_i(T, U) = S_c(T)\text{’}}{\text{‘}T\text{’}}) \geq \text{minCorrProb}”$, referred to as the boostability hypothesis,\(^2\) based on the number of observed error-free inputs from user $U$. The lower the significance level we can accept this hypothesis with, the higher the boost probability for $U$. Three further notions help formalizing this:

**Notation** $\text{Correct}(U)$ denotes the observed number of correct inputs from user $U$ since his last erroneous input. $\text{CorrSig}(U)$ is the post hoc significance level for accepting the boostability hypothesis based on $\text{Correct}(U)$ correct observed inputs. The quotient of $\text{maxFalseBoostProb}$ and $\text{CorrSig}(U)$ then becomes an upper bound for the boost probability for $U$. Formally, this is:

\[
\text{BoostProb}(U, T) \leq \frac{\text{maxFalseBoostProb}}{\text{CorrSig}(U)}
\]

\[
\Leftrightarrow \text{BoostProb}(U, T) \leq \frac{\text{maxFalseBoostProb}}{P(\frac{\text{‘}S_i(U(T) = S_c(T)\text{’}}{\text{‘}T\text{’}} < \text{\text{minCorrProb}’’})}
\]

\[
\Leftrightarrow \text{BoostProb}(U, T) \leq \frac{\text{maxFalseBoostProb}}{\frac{\text{\text{minCorrProb}’’}}{\text{\text{minCorrProb}’’}}}
\]

\[
\text{maxFalseBoostProb} \geq \text{BoostProb}(U, T)
\]

Note that this upper bound increases exponentially with $\text{Correct}(U)/T$. To prevent $\text{BoostProb}(U, T)$ to grow to or beyond 1, which would factually deactivate voting for user $U$ and thus might foster cheating, we use $(1 - \text{maxFalseBoostProb})$ as an additional upper bound for the boost probability. The rationale behind this bound is that the lower $\text{maxFalseBoostProb}$ is, the slower is the growth of $\text{BoostProb}(U, T)$. We thus can afford a higher maximum boost probability, as it takes a very large number of correct inputs to achieve. Further, we want $\text{BoostProb}(U, T)$ to be 0 for $\text{Correct}(U) = 0$ and therefore subtract $\text{maxFalseBoostProb}$.

**Definition** $\text{BoostProb}(U, T)$

\[
:= \min \left( \left(1 - \text{\text{maxFalseBoostProb}}\right), \left(\frac{\text{\text{minCorrProb}’’}}{\text{\text{minCorrProb}’’}} - 1\right) \right)
\]

**Example** 7 this example illustrates how the boost probability of a user increases as he contributes more and more correct inputs: Suppose that a given task $T$ consists of

\(^{2}\) Note that ‘hypothesis’ does not mean ‘a research hypothesis of ours’ in this current context; it means the hypothesis that a user has a sufficiently low error probability to be eligible for a vote boost.
three decisions. Further, suppose user $U$ has contributed a correct inputs to the previous $\text{Correct}(U)=100$ decisions. Finally, let $\max\text{FalseBoostProb} = 1 \%$ and $\min\text{CorrProb} = 99 \%$. Then the probability of boosting the vote of $U$ is:

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

For the boost probability to exceed 50 \% for the given task $T$ and values of $\max\text{FalseBoostProb}$ and $\min\text{CorrProb}$, $\text{Correct}(U)$ has to exceed 1.173. This means that, for tasks consisting of three decisions, $U$ has to contribute inputs to 391 tasks without making a mistake. When $\text{Correct}(U)$ becomes 1.374, i.e., after 458 tasks of the size of $T$, the boost probability reaches its upper limit of $(1 - \max\text{False BoostProb}) = 99 \%$.

For values that are less strict, e.g., $\max\text{FalseBoostProb} = 5 \%$ and $\min\text{CorrProb} = 95 \%$, the boost probability is much higher:

$$\text{BoostProb}(U, T) = 0.05 \times (0.95^{\frac{T}{100}} - 1) = 22.6 \%$$

When $\text{Correct}(U)$ exceeds 141, i.e., after 47 tasks the size of $T$, $\text{BoostProb}(U, T)$ exceeds 50 \% with this second set of values. The upper limit of 95 \% is reached when $\text{Correct}(U)$ exceeds 176, i.e., after 59 tasks.

User motivation: in addition to increasing throughput, Vote Boosting can stimulate high-quality inputs if combined with a payoff function of appropriate design. For instance, let there be a fixed total payoff $\text{Pay}(T)$ for each task $T$, and let this payoff be shared between all users who have contributed an input for $T$, as in the previous section. If the input of a user $U$ receives a vote boost, his reward is equal to $\text{Pay}(T)$; otherwise, he only receives $\text{Pay}(T)/\text{WorkExp}_V(T)$, his share of the reward. Formally, with Vote Boosting, the expected payoff for a user $U$ on a task $T$ is:

$$\text{PayExp}_V(U, T) = \text{Pay}(T) \times \text{BoostProb}(U, T) + (1 - \text{BoostProb}(U, T)) \times \text{Pay}(T)/\text{WorkExp}_V(T)$$

Observation: note that $\text{WorkExp}_V(T) > 2$. This means that receiving a vote boost at least doubles the payoff a user receives for his input. Thus, it is desirable for users to first achieve and then maintain a high boost probability. As the only way to do so is to contribute correct input to each task, it fosters high-quality inputs.

4.5 Sampled Probing

Sampled Probing is a generic measure to discourage cheating. In particular, it tests users for their honesty and penalizes them if they fail such a test. The penalties reduce the expected payoff when cheating to such a degree that making thoughtful inputs becomes the dominant strategy. This holds even if simply submitting the original state of a task as an input takes considerably less time than contributing thoughtfully. Namely, the penalty factor (see below) allows for adjusting the penalization so that cheating is disadvantageous, irrespective of how much it reduces the working time per task.

Sampled Probing generalizes the approaches used by Eckert et al. (2010) and von Ahn et al. (2008). Like the latter mechanism, it uses tasks that are already complete, and whose results are therefore known, to probe users for their reliability. It does not do so in every task, as this is impractical if tasks are larger than in von Ahn et al. (2008) and consist of connected decisions. Instead, Sampled Probing inserts tasks with already-known results in the stream of tasks with a certain probability (the probe rate $pr$).

**Notation** A Probe $P$ consists of a probe task $T_P$, its original state $S_O(T_P) = \text{S1}(T_P)$, and its result $S_R(T_P) = \text{S2}(T_P)$. $pr$ is the probe rate, i.e., the probability of a crowdsourcing system probing a user $U$ with a probe task $T_P$ belonging to a probe $P$ instead of a task the result is yet unknown for.

**Definition** $\text{Pass}(P, S_I(T_P, U))$ is the function that decides whether or not a user $U$ who has submitted $S_I(T_P, U)$ as his input to $T_P$ passes the probe $P$, formally:

$$\text{Pass}(P, S_I(T_P, U)) := \begin{cases} true & \text{if } \text{Dist}(S_I(T_P, U), S_R(T_P)) \leq \text{Dist}(S_I(T_P, U), S_O(T_P)) \\ false & \text{otherwise} \end{cases}$$

Probing users: whenever a user $U$ retrieves a task to work on it, the crowdsourcing system instead returns a probe task $T_P$ belonging to a probe $P$ with probability $pr$. Because $T_P$ has been sampled, it is similar to other tasks. Thus, $U$ cannot recognize that he is being probed and submits his input $S_I(T_P, U)$ as normal. He passes the probe if $\text{Pass}(P, S_I(T_P, U))$ returns true. Otherwise, his input is considered a cheating attempt.

Sampling Probes: to obtain appropriate probes, Sampled Probing inspects each task $T$ after its result $S_R(T)$ is available. If the result differs substantially from the original state of the task, i.e., $\text{Dist}(S_O(T), S_R(T))$ is large, $T$ is a good probe, for two reasons: (1) If $S_O(T)$ was free from errors, i.e., $\text{Dist}(S_O(T), S_R(T)) = 0$, users would not have to correct anything to generate a correct input; such tasks are ineffective to detect cheating. (2) If users make a mistake in a probe task, this has to be distinguishable from cheating. Thus it is better to have several errors to correct in a probe, so users can still pass if they make few mistakes.
task from the running example (see Sect. 4.1) is a good probe task $P$ because $\text{Dist}(S(O(P),S(P))) = 3$. Suppose user $U_1$ from the illustration in Sect. 4.3 is probed with $P$ and submits the input $S_{1,U_1}(P) = \{\text{‘page header’, ‘main text’, ‘footnote’}\})$. Even though this input contains a miss error in decision $D_4$, it still passes the probe because $\text{Dist}(S_{1,U_1}(P), S_{2}(P)) = 1$ is smaller than $\text{Dist}(S_{1,U_1}(P), S_{O}(P)) = 2$.

Throughput: users cannot contribute input to real tasks while they are working on probe tasks. Thus, probing does decrease throughput to a degree equal to the probe rate $pr$, while they are working on probe tasks. Thus, probing does decrease throughput to a degree equal to the probe rate $pr$, so $pr$ should be low.

Analysis of user motivation: in order for probing to serve its purpose, the penalty for a user failing a probe has to be sufficiently high. In particular, the expected reward decrease throughput to a degree equal to the probe rate $pr$, respectively.

We generally assume that $\text{Time}_{H}(U,T) > \text{Time}_{C}(U,T)$.

**Notation** $\text{Time}_{H}(U,T)$ and $\text{Time}_{C}(U,T)$ denote the time it takes a user $U$ to honestly provide input for a task $T$ and the time it takes him to submit an input when cheating, respectively.

For cheating to be inefficient, the following must hold:

$$\frac{\text{PayExp}_{H}(U,T)}{\text{Time}_{H}(U,T)} > \frac{\text{PayExp}_{C}(U,T)}{\text{Time}_{C}(U,T)}$$

The relation of $\text{PayExp}_{H}(U,T)$ and $\text{PayExp}_{C}(U,T)$ depends on the input-aggregation function in use: With $r$-Redundancy, the two values are equal, so cheating is always advantageous. With v-Voting, $\text{PayExp}_{H}(U,T)$ is $\text{Pay}(T)/\text{WorkExp}_{H}(T)$, thus at most $\text{Pay}(T)/v$, so cheating is advantageous if $v \times \text{Time}_{H}(U,T) > (v+1) \times \text{Time}_{C}(U,T)$. Finally, with Vote Boosting, the maximum value for $\text{PayExp}_{H}(U,T)$ is close to $\text{Pay}(T)$. Because users who cheat regularly tend to have frequent errors in their inputs, they will hardly receive a vote boost. Thus, $\text{PayExp}_{C}(U,T)$ is the same with or without Vote Boosting. Consequently, with Vote Boosting, cheating is only advantageous if $\text{Time}_{H}(U,T) > v \times \text{Time}_{C}(U,T)$.

**Notation** $P(‘S_{O}(D) = S_{R}(D)’) \text{ denotes the probability that the original state of a decision } D \text{ equals its result;}$ $P(‘S_{O}(T) = S_{R}(T)’) \text{ denotes the same probability for an entire task } T$.

Using standard combinatorics, we obtain the following probability of a correct result for a decision $D$:

$$P(‘S_{O}(D) = S_{R}(D)’) \text{ computes as:}$$

$$P(‘S_{O}(D) = S_{R}(D)’)= P(‘S_{O}(D) = S_{C}(D)’) \times (1 – P(‘\text{‘add’})\text{‘}$$

$$+ P(‘S_{O}(D) \neq S_{C}(D)’) \times P(‘\text{‘miss’}\)$$

Note that even though this formula looks very similar to the one for the first two inputs for a decision $D$ to agree, it is different because it refers to the equality of the original state of a decision $D$ to its result.

**Observation:** For 2-Voting, the expected payoff PayExp$_{C}(U,T)$ then is as follows:

$$\text{PayExp}_{C}(U,T) = ((1 – P(‘\text{‘cheat’})\text{‘}) \times P(‘S_{O}(D)$$

$$= S_{R}(D’), D \in T)^{|T|} + P(‘\text{‘cheat’})\text{‘} \times \text{Pay}(T)/2$$

$$+ (1 – (1 – P(‘\text{‘cheat’})\text{‘}) \times P(‘S_{O}(D)$$

$$= S_{R}(D’), D \in T)^{|T|} – P(‘\text{‘cheat’})\text{‘} \times \text{Pay}(T)/3$$

The rationale behind this is the following: Assume user $U$ cheats on a task $T$, i.e., he submits $S_{O}(T)$ as his input. One more input is required for 2-Voting to yield the result $S_{R}(T)$. Let a second user $V$ provide this input. Now the payoff of user $U$ as follows: If $V$ submits $S_{O}(T)$ as his input as well, be it due to cheating as well or due to checking honestly and not making any changes, $S_{R}(T)$ emerges after two inputs, so the payoff for both $U$ and $V$ is $\text{Pay}(T)/2$. If $V$ submits something different, i.e., $S_{O}(T, U) \neq S_{O}(T, V)$, a third input is required for $S_{R}(T)$ to emerge. Consequently, the payoff for $U$, $V$ and the third contributor) is $\text{Pay}(T)/3$.

**Example 9** this example illustrates that a user can in fact increase his overall payoff by cheating in a scenario with 2-Voting and Vote Boosting, but without Sampled Probing: let the global probability of cheating be as high as 20%; with the values of our running example, the expected payoff turns out to be:

$$P(‘S_{O}(D) \neq S_{R}(D)’) = 0.2128$$

$$\text{PayExp}_{C}(U,T) = 0.4179 \times \text{Pay}(T)$$

Due to Vote Boosting, the expected payoff for contributing a thoughtful input, PayExp$_{H}(U,T)$, is at most slightly more than twice this value. This means that if cheating on a given task $T$ by submitting its original state as an input takes less than half as long as contributing thoughtfully, cheating is advantageous.

The example shows that cheating remains to be penalized. To achieve this, Sampled Probing introduces a penalty factor into the payoff function to reduce the payoff for a user who has recently been caught cheating, i.e., has recently failed a probe. Formally, this means the following:
Notation LastFail(U) is the number of tasks user U has contributed inputs to since last failing a probe.

Definition PenaltyFactor(U) is the function that provides the penalty factor for user U. penSev is a configuration parameter that controls the severity of the penalization. PenaltyFactor(U) is:

\[
\text{PenaltyFactor}(U) = \begin{cases} 
pr \times \text{penSev} & \text{if } \text{LastFail}(U) < (1/pr) \\
1 & \text{otherwise}
\end{cases}
\]

To influence, based on the penalty factor, the payoff a user U receives for contributing an input to a task T, Sampled Probing replaces any given payoff function Pay0(U, T) with one that enforces penalties:

Definition PaySP(U, T) is the payoff function for Sampled Probing:

\[
\text{PaySP}(U, T) := \text{Pay0}(U, T) \times \text{PenaltyFactor}(U)
\]

Observation: This means that after failing a probe, a user U has to contribute inputs to the next \((1/pr - 1)\) tasks, which statistically are not probes, to make up for the failed probe. He then gets an overall reward that is at most penSev times the reward for submitting thoughtful input for the probe in the first place. The probability to get caught cheating is equal to the probe rate pr. The expected reward for cheating on task T then is:

\[
\text{PayExpSP}(U, T) = (1-pr) \times (pr \times p) \times \text{PayExpC}(U, T)
\]

The lower penSev, the harder the penalization. After a user U has failed a probe, different values for penSev mean the following: penSev = 0 denies user U any payoff for the next \((1/pr - 1)\) tasks. penSev = 1 in turn grants user U exactly the same overall payoff for the next \((1/pr - 1)\) tasks that he would have received for providing thoughtful input on the probe he failed. Finally, penSev = \((1/pr)\) pins PenaltyFactor(U) to 1 and thus alleviates the penalization completely.

Thus, penSev values between 0 and \((1/pr)\) represent different severities of penalization. Values above \((1/pr)\) would turn penalization into a reward, as such values increase the penalization factor beyond 1 after a user has failed a probe.

Example 10 for a probe rate pr = 10% and a penalty factor p = 1, for instance, Sampled Probing reduces the expected reward for continuous cheating to 9% of what it would be without Sampled Probing. Thus, cheating does not pay off any longer. This is the case even if cheating on a task is considerably less effort/less time-consuming than contributing honestly.

Discussion: Sampled Probing is promising to discourage cheating at the cost of some throughput. Due to its parameters pr and penSev, it is sufficiently flexible. Namely, the value of the configuration parameter penSev can be chosen to render cheating inefficient for any time advantage it may have. The lower the probe rate is, the lower is the expected reward for cheating, and the lower is the impact of the probes on throughput. However, users also have to perceive a realistic threat of being probed, so pr should not be too low.

5 Evaluation

To evaluate our mechanisms, we have run extensive simulations. For all parameter combinations, we have tested many variations of input-aggregation and payoff functions.

5.1 Experimental setup

Our experimental setup covers the whole space of exogenous parameters, namely the properties of the tasks and the competence and honesty of the users. The sets of tasks we have used have two parameters: the number of options per decision, and the accuracy of the original states, i.e., the probability that the original state of a decision is correct. We have generated 9 sets with 1,000,000 tasks each, with 2, 3, or 4 options per decision and 80, 90, and 95 as the accuracy for the original states. Each task consists of 5–10 decisions, normally distributed over that interval.

The user populations we have tested have two parameters: their mean probabilities of cheating and of making mistakes, respectively. We have used values of 1, 4, and 15 for both, generating populations of 1,000 users for each of the resulting nine combinations. For the individual users, the probabilities of cheating and of mistakes have been exponentially distributed over [0, 1] around the respective mean values.

Users are implemented as follows: When mistakenly making an add error on a decision that has more than two options, a user randomly selects one of the erroneous options. Users take a fixed time t per decision when contributing thoughtfully. Changing the state of a decision increases this time to \(2 \times t\); cheating decreases it to \(t/2\). At runtime, each user is a separate thread, so users work concurrently and as independent of each other as possible.

In all, we have run simulations for 181 input-aggregation functions: One is the base case, i.e., each task requires one input to complete. The other 180 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 is to deactivate it, the other 13 are different combinations of values for minCorrProb and...
maxFalseBoostProb; all those setups without Sampled Probing, and with probe rates of 1, 4, and 15. The penSev parameter was fixed to 1 in all setups, as its effect is entirely predictable.

There is a fixed payoff Pay(D) per decision; the payoff function distributes Pay(D) equally among all users who have contributed an input. Users also receive a payoff for providing input for probe tasks if they pass it. The rationale behind this design decision is that a passed probe is a thoughtful input and deserves a payoff. In addition, crowdsourcing services like the Amazon Mechanical Turk do require a per-task payoff, so our decision reflects reality.

5.2 Results

From a total of 14,661 scenarios simulated (9 task lists × 9 user populations × 181 input-aggregation functions), we report on the four analyses we deem most interesting.

v-Voting vs. r-Redundancy: Table 1 shows the average accuracy of results and the average number of inputs 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, task sets, and settings of Sampled Probing.

Clearly, v-Voting is superior to r-Redundancy in terms of throughput, requiring significantly fewer inputs for the same result accuracy. This substantiates the results of the analytical assessment. Interestingly, result accuracy 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 the ambiguous decisions that can occur with r-Redundancy (see Sect. 3.1).

From a different angle, v-Voting increases the data quality achievable with a given maximum number of inputs/at a given maximum cost: 4-Voting requires even less inputs per task than 5-Redundancy, yet halves the number of remaining errors.

Sampled Probing: Figure 1 graphs the cheating probability of users against their expected payoff per task and overall, without probing and for different probe rates. We excluded setups with Vote Boosting from this aggregation in order to isolate the effects of probing. Sampled Probing turns out to serve its purpose well: The overall payoff for users who cheat frequently is less than for users who contribute thoughtfully. This holds even though the former submit a considerably higher number of inputs. The fact that the per-task payoff increases for higher probe rates is due to the payoff for probe tasks.

Vote Boosting: Figure 2 visualizes the impact of Vote Boosting, namely the increase in throughput and in errors. The effect of changes to minCorrProb (abbreviated as c in the figures) and maxFalseBoostProb (abbreviated as m in the figures) is similar for all three values of the vote-majority parameter v we have tested: The more liberal the parameter settings, the higher the increase in throughput, but the number of errors is higher as well. The dependency

| 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 |

Fig. 1 Per-task and overall payoff depending on cheating

Fig. 2 Effects of Vote Boosting
Table 2  Inputs required for achieving 99.5 % result accuracy

| Mean probability of cheating | Accidental errors |
|-----------------------------|-------------------|
| 1 %                         | 4 %               | 15 %             |
| v = 2, pr = 0              | v = 2, pr = 0     | v = 3, pr = 0    |
| m = 8 %, c = 92 %          | m = 4 %, c = 96 % | m = 2 %, c = 98 %|
| 1.14 (99.51 %)             | 1.78 (99.63 %)    | 3.78 (99.55 %)   |
| v = 2, pr = 0              | v = 2, pr = 0     | v = 3, pr = 0    |
| m = 8 %, c = 92 %          | m = 4 %, c = 96 % | m = 2 %, c = 98 %|
| 1.42 (99.57 %)             | 1.93 (99.51 %)    | 4.48 (99.51 %)   |
| v = 2, pr = 0              | v = 2, pr = 0     | v = 4, pr = 1 %  |
| m = 4 %, c = 96 %          | m = 4 %, c = 96 % | m = 4 %, c = 96 %|
| 3.94 (99.65 %)             | 4.6 (99.61 %)     | Not achieved     |
| v = 4, pr = 0              | v = 4, pr = 0     | 5.38 (98.62 %)   |
| m = 2 %, c = 98 %          | m = 2 %, c = 98 % | v = 4, pr = 0 m = 0 |
| 4.48 (98.62 %)             |                  |                  |

seems almost linear for both. For a given result accuracy required, this predictable linear behavior allows system designers to tune the parameters 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 an average result accuracy of at least 99.5%, broken up across the nine 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 maxFalseBoostProb/m and minCorrProb/c), and some also use Sampled Probing (parameter pr). A value of 0 for pr or maxFalseBoostProb/m indicates that Sampled Probing or Vote Boosting were not used, respectively. These results point out the correlation between the capability and honesty of contributing users and crowdsourcing throughput; the latter directly translates into the per-task cost in scenarios with a per-input payoff, e.g., the Amazon Mechanical Turk (AMT). With low probabilities for both mistakes and dishonest behavior, 1.14 inputs per task are enough to achieve the desired accuracy. This number increases sharply if either of the two probabilities increases. With the most pessimistic values for both, even 5.38 inputs per task are not enough to reach the goal. This emphasizes the importance of fostering high-quality inputs and of deterring users from cheating.

Crowdsourcing Strategy: as our simulations have shown, the strategy suited best to achieve a desired result quality at as much throughput/little cost as possible strongly depends on the exogenous parameters. These parameters are impossible to predict at the beginning of a crowdsourcing project. Thus, we recommend starting with pessimistic assumptions, i.e., initially choosing a setup that favors result quality over throughput. Later, experts can assess the result quality actually achieved (e.g., from a sample of task results) and deduce more accurate estimates of the exogenous parameters. Afterwards, the endogenous parameters can be adjusted to optimize throughput. We recommend periodic repetition of this cycle of assessment and adjustment.

6 Conclusions

Crowdsourcing is becoming a popular approach to large-scale data processing efforts that require human input. But working with a large and anonymous user community also raises new challenges. Namely, both possible inability and potential dishonesty of the users threaten the quality of the results. This incurs a harsh tradeoff between 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 the static redundancy-based approaches used in previous work by means of more sophisticated aggregation of the individual inputs. Vote Boosting builds upon v-Voting; it further increases throughput by capitalizing on users who are particularly capable, and it rewards these users. Sampled Probing tests users for their honesty and punishes them in case of a failed test.

Extensive simulations over a wide range of exogenous parameters have confirmed the suitability of all three mechanisms, substantiating our findings from theoretical analyses. In particular, simulation results show (1) that v-Voting yields somewhat higher result quality than r-Redundancy with considerably fewer inputs per task, (2) that Vote Boosting allows trading off result quality in favor of throughput in a predictable fashion, and (3) that Sampled Probing turns cheating into a losing strategy even in the long haul.

In future work, we plan to verify the findings presented in this paper in large-scale real-world experiments. Further,
we plan to experiment with additional variants of Vote Boosting, namely with alternative definitions of the function that computes the boost probability, and with input-aggregation functions that can boost the weight of an input in a more fine-grained fashion. The goal is to further reduce the number of inputs per task required to achieve a given accuracy in the result.

References

AMT. The Amazon Mechanical Turk, http://www.mturk.com
Cooper S, Khatib F, Treuille A, Barbero J, Lee J, Beenen M, Leaver-Fay A, Baker D, Popovic Z (2010) Predicting protein structures with a multiplayer online game. Nature 466:756–760
Eckert K, Niepert M, Niemann C, Buckner C, Allen C, Stuckenschmidt H (2010) Crowdsourcing the assembly of concept hierarchies. In: Proceedings of JCDL 2010, Brisbane, Australia
Lintott CJ, Schawinski K, Slosar A, Land K, Bamford S, Thomas D, Raddick MJ, Nichol RC, Szalay A, Andreescu D, Murray P, Vandenberg J (2008) Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. Monthly Notices of the Royal Astronomical Society, 389. doi: 10.1111/j.1365-2966.2008.13689.x
Newby GB, Franks C (2003) Distributed proofreading. In Proceedings of JCDL 2003. Houston, TX, USA. doi:10.1109/JCDL.2003.1204888
Sautter G, Böhm K (2011) High-throughput crowdsourcing mechanisms for complex tasks. In: Proceedings of SocInfo 2011, Singapore
Sautter G, Agosti D, Böhm K, Klingenberg C (2009) Creating digital resources from legacy documents—an experience report from the biosystematics domain. In: Proceedings of ESWC, Heraklion, Greece
Siourpaes K, Hepp M (2007) OntoGame: towards over-coring the incentive bottleneck in ontology building. In: Proceedings OTM 2007, Vilamoura, Portugal
Snow R, O’Connor B, Jurafsky D, Ng AY (2008) Cheap and fast — but is it good?: evaluating non-expert annotations for natural language tasks. In: EMNLP 2008, Morristown, NJ, USA
Von Ahn L (2006) Games with a purpose. IEEE Comput 29(6):92–94
Von Ahn L, Blum M, Hopper N, Langford J (2003) CAPTCHA: using hard ai problems for security. Advances in cryptology—EUROCRYPT 2003. Springer Berlin/Heidelberg. doi:10.1007/3-540-39200-9_18
Von Ahn L, Maurer B, McMillen C, Abraham D, Blum M (2008) reCAPTCHA: Human-Based Character Recognition via Web Security Measures. Science 321 (5895). doi:10.1126/science.1160379