Human Computation (2014) 1:1:67-93
© 2014, Lasecki et al. CC-BY-3.0
ISSN: 2330-8001, DOI: 10.15346/hc.v1i1.5

# Architecting Real-Time Crowd-Powered Systems

**WALTER S. LASECKI**, UNIVERSITY OF ROCHESTER

**CHRISTOPHER HOMAN**, ROCHESTER INSTITUTE OF TECHNOLOGY

**JEFFREY P. BIGHAM**, CARNEGIE MELLON UNIVERSITY

## ABSTRACT

Human computation allows computer systems to leverage human intelligence in computational processes. While it has primarily been used for tasks that are not time-sensitive, recent systems use crowdsourcing to get on-demand, real-time, and even interactive results. In this paper, we present techniques for building real-time crowdsourcing systems, and then discuss how and when to use them. Our goal is to provide system builders with the tools and insights they need to replicate the success of modern systems in order to further explore this new space.

## 1. INTRODUCTION

Crowdsourcing elicits responses from groups of people via an open call to action (Surowiecki, 2005). Such calls have recently begun being used to elicit people's contribution to human computation processes (von Ahn, 2005). This work might be done for pay, for a sense of accomplishment, or because the worker gets some direct benefit from the final result. Human computation allows systems to go beyond what automated approaches alone are capable of and, using large groups, leverages the "wisdom of crowds" to solve problems that cannot be easily solved by an individual.

In the past, the time needed to both recruit crowds and complete tasks meant that crowdsourcing was primarily an offline mechanism, where the completion time was a low priority. However, more recent research has yielded approaches that leverage the scale and structure of current platforms, such as Amazon Mechanical Turk, to collect answers in seconds, not hours or days (Bernstein et al., 2011; Bigham et al., 2010). This allows task requesters to leverage crowds on demand for interactive applications (Lasecki et al., 2011), e.g., converting speech to text fast enough to provide deaf users with access to their environment (Lasecki et al., 2012). The robustness of these systems also makes it possible to deploy them in real settings, and use the collected data to train automated systems to play increasingly larger roles (Lasecki et al., 2013a).

However, since these systems require the understanding of a wide variety of issues – ranging from

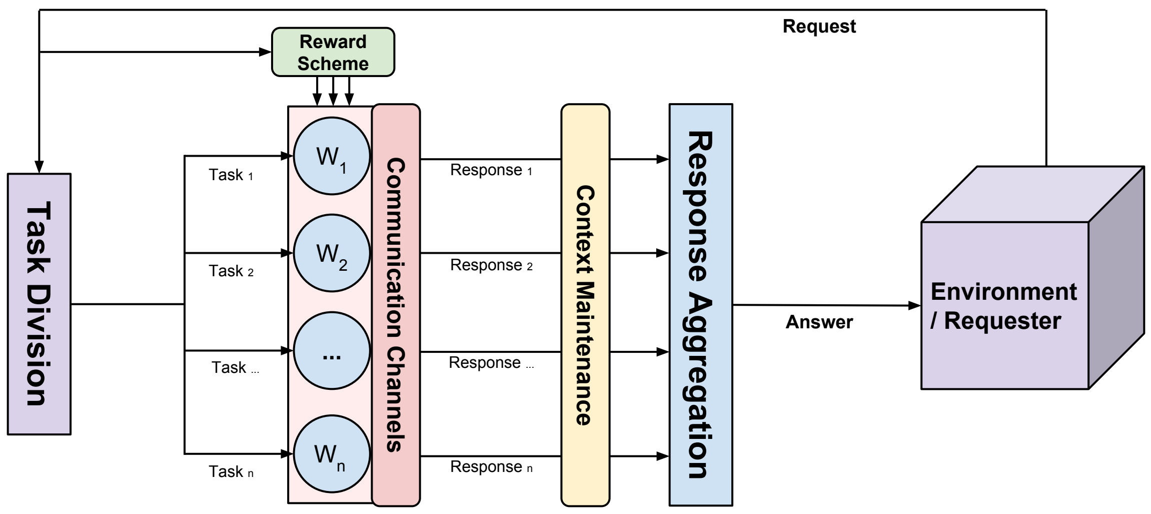

***Figure 1.* Our real-time crowd-powered system architecture.**

interface design to human factors to game theory to aggregation and voting algorithms – the amount of broad and specialized knowledge and experience needed to build them is a major barrier.

Our goal is to inform the design and implementation of real-time crowd-powered systems. We strive to make the problem of integrating real-time crowdsourcing into interactive systems more approachable for system builders by reducing the need for significant prior domain expertise in crowdsourcing in order to develop solutions. We focus on the tools, methods, and paradigms that have been successful in prior work, and discuss when and how different components and approaches should be used in order to be most successful. We also consider recurring patterns of classes and communicating objects that recur in real-time crowdsourcing. This information has not previously been available in a general, interconnected way, since work on most prior systems has been separated into task-focused systems and studies.

In the rest of this paper, we will discuss the following aspects of real-time crowd systems of crowd-powered systems (this architecture is visualized in Figure 1):

– Task division
– Answer aggregation
– Maintaining context
– Facilitating communication between workers
– Rewarding workers
– Integrating automated systems

## 2. BACKGROUND AND RELATED WORK

Using crowdsourcing for human computation requires ideas from a wide range of fields, from human factors and human-computer interactions to decision science and voting theory. In this section, we briefly discuss some of the foundational work tying these fields to crowdsourcing.

## 2.1. Definitions

We begin by defining some of the terms as we use them in this paper:

- **Task:** A job or computation that requires human input to complete.
- **Requester:** A person or system who needs an answer to a *task* from the crowd.
- **Crowd:** A dynamic group of workers available via an online platform to complete tasks.
- **Crowd Worker ("worker"):** A person completing tasks on a crowdsourcing platform.
- **Microtask:** A small, context-free task that is common in crowdsourcing.

## 2.2. Related Work

Crowdsourcing spans many fields: collective intelligence, social computing, economics, and human-computer interaction just to name a few. Collective intelligence studies how the dynamics of groups leads to intelligent and emergent behaviors, and social computing studies how computationally mediated interactions between people arise and their effects on users. It also encompasses the design of purpose-built systems that facilitate and moderate interactions. Human-Computer Interaction (HCI) and other Computer Science disciplines have mainly focused on crowds of workers than can be recruited algorithmically (Little et al., 2010; Bernstein et al., 2010, 2011; Bigham et al., 2010; Lasecki et al., 2012). This work focuses on crowd-powered systems that are able to algorithmically recruit and organize workers for a specific, task-focused objective. A discussion of crowdsourcing and related disciplines can be found in (Quinn and Bederson, 2011) and (Michelucci, 2013).

There are a number of crowdsourcing challenges we explore in this paper, including:

- **Dynamic workforces and worker availability.** One of the greatest benefits of participating in crowd work as a worker is the flexibility to take tasks when it is convenient for them, and stop working when they choose to. However, for requesters, this flexibility means that they cannot rely on particular workers to be available to complete a task when needed, or for workers to remain present over the course of multiple tasks (Mao et al., 2013). Even within a single task, workers may leave if the need arises or simply if they do not enjoy or understand the task.
- **Variable system configurations.** The stability and configuration of worker computers is not as controlled as in a traditional work setting. The setup and capabilities of workers' computers can vary widely even within the same platform. This means that task design must carefully consider what technologies might prevent some workers from participating in a task. Computer glitches can also result in workers disconnecting from a task before completion.
- **Answer reliability.** The crowd's anonymity (leading to minimal accountability), situational and experiential variability (leading to differing worker performance between tasks), and the lack of a strong, quality signal (leading to the requester having no reliable way of vetting workers prior to hiring them), all result in a process that is often unwieldy and potentially unreliable.

It is important to note that some of these issues (those involving worker reliability and availability) arise more frequently when using general-knowledge crowd marketplaces (such as Mechanical Turk), where specific skills are not on display. Longer term interactions are commonly seen on expert crowdsourcing platforms, such as oDesk (https://www.odesk.com/). As general knowledge platforms evolve and grow, it is likely that many of these properties will tend towards those seen in more expert platforms (Kittur et al., 2013).

## 2.3. **Crowd-Powered Systems**

The ability to correct worker input and get a reliable signal makes it feasible to create task-oriented systems powered by the crowd. In this section, we outline some of the work that has led from reliable offline answers from the crowd to systems that can interact with users in real time.

### *2.3.1. Making Crowds Available On-Demand*

Being able to respond quickly, often in one second or less, is a key component of most interactive systems (Bernstein et al., 2011). However, when using people to power the system's responses, reducing latency becomes difficult. Worse, if these workers must be recruited each time an interaction might occur, the problem of reacting in a suitable amount of time becomes difficult.

To address the shortcomings of these systems in interactive settings, some systems have explored methods for ensuring that workers are always available for a task as soon as they are needed. VizWiz (Bigham et al., 2010) recruits workers in advance and keeps them engaged by completing prior tasks in order to train workers while keeping them available when needed. (ready as soon as their most recent task is done). Adrenaline (Bernstein et al., 2011) used a passive waiting approach combined with pop-up alerts to keep people available, not busy completing a separate task. Instead, multiple workers were recruited and asked to keep a browser window open which alerts them as soon as a task arrives. By recruiting a larger crowd, the fastest-to-respond member can be recruited, speeding the response time up even more (Lasecki et al., 2013). By integrating queuing theory and other optimizations involving the size of the potential workforce, it is possible to get workers to join a task in less than a second (Bernstein et al., 2012).

### *2.3.2. Synchronous Crowds*

Recruiting individual workers quickly is important for making crowd-powered systems feasible in interactive settings, but does not allow for many of the consensus-based approaches common in crowd-powered systems that were discussed above. To enable this type of quality assurance process, we need to recruit multiple workers in real-time, often *synchronously*.

Synchronous crowd are necessary for systems in which workers build on one another's responses to generate a single answer. For example, Adrenaline (Bernstein et al., 2011) asked workers to rapidly refine others' cropped video segments until a single best still image was selected, while other work has used synchronous pairs of workers to collect task-oriented text dialogs using crowd workers (Lasecki et al., 2013a).

Finally, there is a growing body of research on the effects of different task coordination schemes on overall system performance. For example, Seaweed (Chilton et al., 2009) had Mechanical Turk workers play economic games, creating an easy platform for economists to study models with real people. Similarly, TurkServer allows social scientists to run synchronous experiments on Mechanical Turk (Mao et al., 2012), which has previously been shown to be a valuable resource for offline tasks (Mason and Suri, 2012).

## 2.4. Real-Time Crowd-Powered Systems

Being able to recruit workers on demand, to work synchronously on a task, makes it possible to create systems that interact with end users in a way that artificial intelligence alone cannot currently support. VizWiz (Bigham et al., 2010) was one of the first such systems – it allowed blind users to take a picture, ask a verbal question about the picture, and get an answer in under 30 seconds. Adrenaline (Bernstein et al., 2011) allowed users to find the best still image in a short video in a matter of seconds, allowing end users to take better pictures with almost no additional effort.

Legion (Lasecki et al., 2011) was one of the first systems to be able to continuously react to its environment as fast as a person could. Legion allowed users to control any existing user interface (UI) with natural language by having the crowd collectively control the UI remotely. This enabled off-the-shelf systems ranging from word processors to robots to be controlled using natural language, without needing any custom applications per-tool. Legion was the first to introduce the *crowd agent* model of crowdsourcing, where the goal is to coordinate the crowd to act as a single collective individual.

Chorus (Lasecki et al., 2013) went beyond natural language control using single commands, and let users easily find information via a conversational assistant that can hold extended dialog with the end user. Chorus asked workers to build on, refine, and vote for one another's conversational responses to create an intelligent dialog system that provides more accurate and thorough responses than any one worker alone would. Chorus:View (Lasecki et al., 2013b) used the Chorus conversational platform to extend VizWiz into a conversational visual question-answering assistant capable of giving real-time feedback to users regarding how to accurately frame their image.

In this paper, we explore the most successful approaches to mediating crowds for interactive tasks, outline and identify the most-appropriate usage scenarios for different components and system designs, and report new approaches and negative results that have not been reported in prior work. This allows system builders to replicate the success of modern interactive crowd-powered systems by appropriately applying these techniques to their projects.

## 2.5. Applications of Real-Time Crowd-Powered Systems

Real-time crowd-powered systems can have an important impact on many areas, include machine learning and accessibility.

### *2.5.1. Machine Learning*

One of the earliest uses of crowdsourcing was to train automated systems. Image labeling (Russell et al., 2008; von Ahn and Dabbish, 2004), natural language tagging (Snow et al., 2008), language translation (Callison-Burch, 2009), and image sketching (Limpaecher et al., 2013) were among the many settings where crowd input was used to train machines. However, like other crowd tasks, these systems got responses from the crowd in an offline, batch-processing fashion.

Real-time crowdsourcing systems, make it possible to supplement and train the system in-situ by getting just-in-time answers from the crowd. For example, Legion:AR (Lasecki et al., 2013a) provided real-time activity recognition labels to train an Hidden Markov Model when the system was not confident in its own answer. Legion:AR could provide training as needed, and hide the system's training process from the user by providing reliable, direct answers.

#### *2.5.2. Accessibility*

Crowd-powered systems have been used to support accessibility since early systems. In fact, remote human-powered support of people with disabilities has long been used, even before the rise of modern crowdsourcing (Bigham and Ladner, 2011).

Interactive systems such as VizWiz, Chorus:View, and Legion:AR, have stretched the types of services that can be provided by these systems farther than ever before. Many of these services would not be possible to provide without a dynamic workforce such as online crowds provide. Others can be provided, but are either not as available to end users, or might even not be as high quality as the crowdsourced solution. For instance, Scribe (Lasecki et al., 2012) is a system that allows deaf and hard of hearing users to have better access to the world by providing real-time captions with a latency of under 5 seconds on demand.

## 3. TYPES OF CROWDS

There are many different ways to recruit groups of workers to a task, and each comes with a trade-off in terms of their motivations, error rate, abilities, cost, and difficulty to recruit. Selecting a type of crowd is a key issue that arises in any crowd-powered system, but the differences can be particularly pronounced in real-time systems. Prior work has analyzed specific platforms for demographic break-downs of their members, which provides insight into education and engagement on the platform (Ipeirotis, 2010). In this section, we broadly outline the types of crowds that exist, their properties, and the pitfalls and opportunities that are associated with each.

### 3.1. Source of the Crowd

The type of call, as well as the population targeted by the call, play one of the most significant factors in guiding the design of the rest of the system.

*Open crowds*, as the name implies, are open to any (or many) contributor(s). In practice, these workers are often recruited as paid microtask workers on platforms such as Mechanical Turk, or recruited as volunteers interested in the final result of the system (e.g., supporting accessibility systems (Bigham et al., 2010; Lasecki et al., 2013b, 2012)). Due to its position as the most ubiquitously used crowd platform in the crowdsourcing literature, and the fact that it contains many examples of the types of pitfalls we try to address in this paper, we use Mechanical Turk as our default platform for examples, unless explicitly mentioned otherwise.

*Known crowds* are crowds whose members are either known to the requester (friends, associates, full-time employees, etc.), or are otherwise known not to be malicious. Knowing that workers can be trusted – even if they cannot be relied upon to have any special skill set – can make a significant difference in choices made during implementation, particularly when it comes to answer aggregation and communication channels between workers (discussed later).

## 3.2. Expertise of the Crowd

Knowing (or not knowing) who composes the crowd is is only one dimension of selecting a crowd behind a system. Expertise can vary widely between crowd workers, as in any work settings. The source of the crowd often only controls how much we know in advance, rather than the range of skills that may be seen in the group.

*Expert crowds* are composed of people experienced in one or more relevant domains. Online platforms for these types of crowds are increasingly commonplace. For example, platforms such as oDesk (www.odesk.com), Elance (www.elance.com), Guru (www.guru.com), or Freelancer (www.freelancer make it possible to hire easily professional contractors who specialize in anything from programming to design, project management, or advertising, among other domains. The level of expertise such workers bring may varying from a few months to several years. Such workers generally charge rates proportional to experience and ability.

*General crowds* are crowds that are not assumed to have any particular expertise related to the task they will be completing. While novice crowds lack specific, known expertise, they are much easier to find, recruit, and hire than experts. The cost is also typically proportionally lower.

## 3.3. Design Implications

Each different expertise level and source of crowds has a different effect on how the system must coordinate workers' efforts to be successful. Each of these considerations constitutes an axis of a continuous space, and as such can be mixed in various ways, contingent on the resources and platforms available to requesters. The space is roughly

Participant crowds consist of groups of people who are all involved in a specific endeavor. These may mix expert workers with non-experts – as is the case in Wikipedia, or FoldIt (Cooper et al., 2010) and eteRNA (Lee et al., 2014).

Open crowds provide a means of easily accessing the largest possible groups of workers to complete a task, but also introduce the possibility of spammers, workers trying to game the system, and even malicious workers. There is a similar trade-off in terms of expertise: lower expertise requirements leads to the ability to more quickly recruit a larger number of workers to a task (because there is a simpler-to-meet restriction imposed), but each worker might not be able to complete work as well as an expert.

If always-available on-demand access to a crowd-powered system is the goal, then moving more towards lower-expertise, unknown crowds opens the largest set of options (Figure 2). This might be why Mechanical Turk has become the primary resource for real-time systems. Another reason is the ease of access of the platform to these types of requests. Platforms that offer batch-processing services that a delayed by hours or days are clearly not viable options. While many services have properties that are not well suited to real-time work, and to our knowledge none specifically optimize for it, the availability and quality of special-built features for real-time systems is expected to improve as more successful systems arise.

Some of the key platform issues to consider are:

– Latency between a post and workers being able to accept (or be issued) a task.

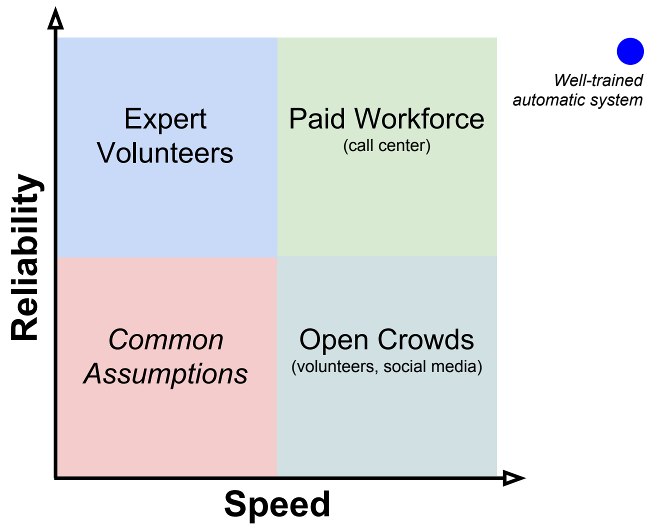

***Figure 2.* Speed and reliability trade-offs for different sources of crowds.**

- The extent to which posts are collapsed into single entries.
- In the case that the platform uses posts that workers can select from: does the platform allow you to make workers aware of the fact that a task needs an immediate response?
- In the case that the platform assigns work directly to workers: does the platform allow you to notify the assignment process that the task needs an immediate response?

## 4. DIVIDING TASKS

Once the workers are recruited, the first issue a system must address is how to assign tasks to them. Depending on the nature of the task, division into synchronous tasks may be appropriate, or coordinated turn-taking might be a better fit. In this section, we discuss the different division strategies available to system builders.

### 4.1. Discrete Versus Continuous Tasks

*Discrete* real-time tasks are self-contained tasks that can be answered entirely by one worker – even if multiple workers may be assigned to increase answer quality or reduce speed. These tasks may include all required context, or may be part of an ordered sequence of tasks in which some context must be passed from one to the next (discussed in Section 6). Single-response tasks are the simplest instance of a discrete task, which require no additional context from prior responses.

*Continuous* real-time tasks are those that engage workers in a closed-loop interaction with the system. This, in turn, allows the system to engage other entities in its environment more fluidly. In many settings, this interaction is key: tasks such as when driving a car or a robot in a changing environment cannot reliably be done as a series of separate pieces, both because of the difficulty of maintaining situational context, and because it would make it hard to react to environmental factors such as pedestrians or falling rocks.

These tasks go outside of the traditional microtask model, but allow for just as much or even more flexibility for workers: they may join at any point, and continue working for as long as they wish. In most cases there will still be a minimum unit or work composing these tasks (e.g., typing a word) – this can be seen as the "microtask" within a continuous task.

However, as with other classes of crowdsourcing algorithms, simply dividing the task does not produce a more effective system – a means of combining these inputs is required (discussed in the next section). Accordingly, the selection of how to divide the task is intimately tied to the recombination strategy that can be used on the input.

## 4.2. Stream Parallelism

*Stream parallelizable tasks* (or *parallelized tasks*) are those that can be split into multiple sub-roles that can be completed by workers in parallel. These roles are carried out simultaneously over the same source-time period.

When workers arrive to a parallelized task with discrete sub-tasks, they are assigned a sub-task. This sub-task may be independent, as may be needed in general image labeling (Bigham et al., 2010; Russell et al., 2008), or part of a context-grouped or even ordered set.

When workers arrive to a parallelized *continuous* task, they are assigned a role and issued a task stream for that role. These roles may be defined statically by the system, e.g., in terms of a subset of controls in a larger interface, or dynamically by the workers themselves. For example, Legion:Gaming (Loparev et al., 2013) allowed each player to selectively control a portion of a video game controller.

The division of control within a task is not required. For instance, to let workers maintain complete situational context, Legion's control interface presented a single feedback stream (video) to all workers (Lasecki et al., 2011). As we will see, many tasks can be trivially stream-parallelized in this way in order to introduce redundancy.

## 4.3. Temporal Division

*Temporally divisible tasks* are those that can be effectively solved by dividing them into multiple subtasks that can be completed by different workers, where each subtask may depend on the output of earlier or later subtasks.

For instance, in order to caption live speech, workers would typically have to type upwards of 250 words per minute, or about 1250 characters per minute – something that exceeds the maximum recorded human typing speed (Wikipedia, 2014). Scribe (Lasecki et al., 2012) divides a streaming set of small ordered segments that are possible for people who can type quickly to caption. These tasks are also redundantly stream-parallelized, asking multiple workers to caption each segment as they arrive in order to produce a more reliable answer after recombination. This type of round-robin turn taking is an example of temporal division used to make the (computer-mediated) group performance better than that of any one constituent member. Another example can be seen in the real-time activity recognition and labeling system Legion:AR (Lasecki et al., 2013a).

Temporally dividing tasks in this way can lead to other benefits to workers' individual performance as well. For instance, TimeWarp (Lasecki et al., 2013b) introduced a collective process to Scribe that allows workers to each listen to audio streams at a reduced playback speed to increase precision and recall, while still decreasing latency.

Generally, the temporal division of tasks is useful when a task can be effectively completed by a worker for short periods of time, but not sustained for extended periods. Turns can also be determined based on semantic roles, with workers trading input control based on their knowledge or role. This can go beyond simple hand-offs and help to avoid context switching costs that arise when a single individual must accomplish multiple orthogonal tasks in order to make the system function as intended.

## 4.4. Correction

*Correctable tasks* are those for which it may not be feasible for human workers to complete in real time, but in which errors can quickly be detected and fixed. Correction is typically a sequential (i.e., non-parallelizable) post-task that is run after a user or machine has generated an initial answer. It can be applied to almost any generation schema, but adds additional latency. This latency is based on the granularity of the task division – larger input segments will increase latency because workers must wait to get access to them. Correctable segments need not be the same size as initial-task segments, e.g., they may be smaller to reduce latency, or larger to provide more context.

As an example of a correction workflow, we can imagine an alternative approach to real-time captioning where one (or more) worker types the initial content, while other workers apply the corrections to the resulting caption stream. Even when content generation is handled by an automated process, the crowd can fill this role, such as in correcting captions generated by automatic speech recognition (Harrington and Vanderheiden, 2013). This example also shows that the correction approach can be used in conjunction with other task division approaches (e.g., stream-parallel or temporally divisible tasks) in order to make even the captioning problem feasible for non-experts.

## 4.5. Combining Approaches

For each of the approaches above, most tasks can use a combination of at least one from each category, assuming the problem is well-suited to each approach individually. There are challenges involved in mixing approaches of course: time scales (segment lengths) may differ between different types of tasks, making them potentially hard to reconcile with one another, and the context needed for different divisions may vary.

There is also the issue of cost. Adding additional workers to any of these roles increases the overall cost of running the system, but adding a new process to a workflow is often a larger increase. For example, going from 4 to 5 workers in the round-robin process in Scribe is much more affordable than parallelizing the task between two workers in all four segment sets.

## 4.6. Task Routing and Mediated Workflows

Regardless of the exact structure of the chosen division, intelligently routing workers can potentially increase the performance of each worker by matching them to a task that they can excel at. This can be done optimally, up to the limits of the information available about the workers' skill sets and task properties. Workers' skills are matched to the task requirements and the configuration yielding the best expected outcome is selected (Jung and Lease, 2013).

Computationally mediated workflows offer the promise of handling much of the coordination work that managers have in classical group-work settings (Retelny et al., 2014). Task routing can efficiently assign tasks to workers who are qualified for, hand-offs can be checked automatically and forwarded with the appropriate context, and new workflows can be generated dynamically based on prior experience. Decision theory can also be used to determine the phases of a workflow. If more workers are needed for a task, they can be routed there until the system is confident in an answer, and the workflow re-adapts accordingly (Dai and Weld, 2011; Lin et al., 2012).

Workflows can also be derived from existing structures of automated computation. For example, CrowdForge (Kittur et al., 2011) uses a Map-Reduce model to divide and recombine subtasks. The crowd might also be able to mediate its own workflows, using systems like TurkoMatic (Kulkarni et al., 2012), which recursively divides tasks based on worker input. The challenge for real-time systems is creating methods of letting workers hand off responsibility and delegate work in a very short amount of time.

### 4.7. Modeling Routing and Decisions

Ho, et al. study the *online task assignment problem*: given a collection of subtasks, they assume that each worker has a skill level for solving each task and a benefit level to the task requester (Ho et al., 2013; Ho and Vaughan, 2012). They present an online algorithm for assigning workers to tasks as they arrive and show the algorithm is, under certain assumptions, close to the optimal offline assignment. Karger also looks at this problem, but from a broader perspective that also considers how to best aggregate answers from competing workers (discussed in Section 5) (Karger et al., 2011). Heidari and Kearns study the problem of using the crowd itself to route discrete subtasks, where each worker, when presented with a subtask, may choose to perform it or pass it to another worker (Heidari and Kearns, 2013).

## 5. ANSWER AGGREGATION

Dividing tasks to help groups of people to provide more complete, accurate, and reliable responses, leads to the need for answer aggregation. Without answer aggregation, users receive multiple, potentially contradictory responses. While some systems, such as VizWiz, simply forward all responses back to a user, this leaves some uncertainty as to the reliability of the responses, especially when answers arrive one by one and could each potentially both be true. In this section, we discuss how answer aggregation algorithms, such as the consensus approaches for batch tasks presented in Section 2, can be applied to real-time systems.

The way the task has been divided determines much of how the aggregation process is structured. However, the aggregation algorithm used can be optimized for specific properties, such as reliability, speed, cost, and consistency.

### 5.1. Optimizing for Reliable Responses

The most common approach to ensuring reliable answers in crowd-powered systems is to increase the level of answer redundancy. This relies on the *wisdom of crowds*, which says that a group of people generally tend collectively towards the correct answer (e.g., after averaging (Surowiecki, 2005)), subject to the bias of the crowd towards a given task. The fine-grain distinction as to when and how people can be biased is beyond the scope of this paper.

However, the level of agreement alone is not the only available means of increasing answer reliability. In many cases, such as when soliciting open-ended responses, agreement between two different workers given a large answer space is a strong signal of correctness. This concept was used with synchronous workers for discrete tasks in The ESP Game (von Ahn and Dabbish, 2004), where workers each guessed a label for an image and the answer was accepted when both workers answered with the same label.

For continuous real-time systems, determining the quality of responses on the fly can be a challenge (Lasecki and Bigham, 2012b; Mashhadi and Capra, 2011). Assigning additional workers to a workflow in order to get redundant responses can have negative impacts on the system. If a worker set that is just large enough to ensure high confidence is chosen in advance to complete tasks in parallel, then the work can be completed as fast as the slowest response. However, it is hard to know just how large the worker set needs to be and, often, reliable agreement with fewer workers than the pre-determined level is possible. On the other hand, if an iterative approach is used to terminate execution as soon as a high-reliability answer is found, it is likely to add undue latency.

Trading off between reliability and speed has been explored in previous work (Singh et al., 2012; Horton and Chilton, 2010), but remains an open problem. Incentives based on early, later-confirmed responses have been one effective means of incentivizing workers in real-time systems to optimize this tradeoff (Lasecki et al., 2013; Singh et al., 2012). Algorithmic optimization approaches to balancing quality under time constraints have also been developed (Boutsis and Kalogeraki, 2013).

## 5.2. Optimizing for Rapid Responses

If a quick answer is the most important aspect of a system, first-response approaches are usually the most effective. VizWiz provided the first result available to users, relying on the fact that end users can typically identify incorrect answers easily for verification (e.g., the answer “A can” to the question “How many calories are in this package of food?” is incorrect). Another contributing factor is that, anecdotally, workers provide more reliable answers when presented with tasks that do social good. More generally, larger sets of workers benefit from natural variance yielding at least one faster worker in the typical case than any single worker making repeated contributions.

## 5.3. Optimizing for Affordable Answers

Cost is a critical factor in making a deployed system sustainable. To minimize the cost, the total worker time used must be reduced, which often means that the total number of task touches needs to be decreased, even if as a consequence each worker must complete a slightly longer task. Doing so can reduce the context switching overhead that workers incur when joining a new task.

To minimize the number of tasks that need to be issued by the system in settings where providing even occasional inaccurate responses to users can be detrimental effects, measures of response confidence can be used. Response sets can vary widely between tasks. For simple tasks with clear correct answers, nearly all workers may be in agreement from the beginning. On the other hand, edge cases, or otherwise difficult or ambiguous tasks might yield conflicting sets of responses with no clear winner, meaning only a response from a larger crowd might be considered reliable. Since this is sometimes unavoidable in collective intelligence systems, our goal becomes to answer each question in as few responses as possible with a given level of confidence.

Iterative workflows offer an effective structure for approaching this problem: by evaluating worker agreement at each iterative step, we can cut off processing when a confidence threshold is reached (Karger et al., 2011). This threshold can have based on standard statistical confidence measures, such as p value or confidence interval. However, as mentioned above, iterative workflows can be too slow for many real-time tasks. As such, they offer a solution only in cases where the delay incurred is acceptable, or where the initial response provided to a user can be updated as new answers arrive (e.g., data visualization or captioning).

## 5.4. Optimizing for Consistent Responses

Optimizing for quick, reliable, or affordable responses can enable systems that respond helpfully to users. But not all problems require only a single answer. Thus far, we've focused on performance criteria that are largely assuming that answers each exist in isolation. In practice, some tasks require the system to maintain state or understand the implications of prior actions (e.g., when creating a virtual agent (Rossen and Lok, 2012; Borish et al., 2014; Lasecki et al., 2013). This idea introduces a new measure of answer quality: *consistency*.

Consistency can be defined as producing responses that do not conflict with prior responses. This definition heavily depends on the task at hand. For conversational interaction for instance, responses should not disagree with prior claims about the system's beliefs, or repeat parts of a conversation that a single person would not (for instance, saying "hello" again after initial introductions have already been completed).

Legion explored a number of approaches to real-time interface control, and found that one of the most reliably fast and accurate was selecting a representative leader from the crowd once each small time unit (Lasecki et al., 2011). By doing this, responses best represent the will of the collective, while avoiding self-contradicting actions from turn to turn.

More generally answer consistency be maintained by ensuring the crowd is aware of prior context, and have the tools (usually communication paths) to avoid conflicting or repeating the answers generated by other workers in settings, especially when their answer will not be aggregated automatically. We discuss both context maintenance and communication channels later in this paper.

## 5.5. Formal and Theoretical Approaches

The main theoretical framework for studying information aggregation among heterogeneous agents is social choice theory. This framework generally consists of two collections: one of agents (i.e., the workers in a crowdsourced setting) and one of *alternatives*. Each agent is assumed to have a (partial) ranking of the alternatives. Often the agent must also report these rankings to an aggregation system. This can often be an arduous task, because the number of alternatives can be very large. Social choice theory typically addresses various ways in which the preferences of individual users can be aggregated into a single preference list. Such methods are called *social welfare functions* or, when they return merely the top choice, *social choice functions*. Typical questions on social choice functions involve whether the function satisfies certain fairness criteria. Famously, Arrow showed that no social choice function must fail at least one reasonable fairness criterion (Arrow, 2012).

Typically, social choice theory considers only one election at a time and assumes that all agents vote simultaneously, as is the case in standard political elections. Recently, Parkes and Procaccia use Markov decision processes to extend elections into multiple rounds in which the preferences of the individuals represent the state of the system at any given time, and each election changes the preferences of each individual (Parkes and Procaccia, 2013). (Hemaspaandra et al., 2012, 2014) consider elections in which agents vote *sequentially*, rather than in parallel.

Another approach to achieving consensus is to treat the task as a machine learning problem in which workers provide partial or noisy answers and the aggregation mechanism filters the best answer from the workers' input. For instance, Snow et al. study the accuracy of Mechanical Turk workers in text annotation tasks. They propose a method for improving accuracy using expectation maximization and a training set of data annotated by task experts (Snow et al., 2008). Kamar et al. used partially-observeable Markov decision processes (POMDPs) in *consensus tasks* (Kamar et al., 2012; Kamar and Horvitz, 2012), which they define as tasks in which multiple workers propose answers to a single question and the goal of the system is to come up with an optimal policy that either recruits a new worker or selects one of the answers already proposed, by considering the cost of adding more workers against the expected improvement in the quality of the answers. See also (Dalvi et al., 2013; Karger et al., 2011; Carpenter, 2011; Liu et al., 2012; Raykar et al., 2010; Sheng et al., 2008; Zhou et al., 2012)

## 6. MAINTAINING CONTEXT

When dividing and later aggregating tasks from a larger job, we need to ensure that workers have enough context to complete their task. Context might be critical, such as remembering previously traversed paths in a robot navigation setting (Lasecki et al., 2011, 2012) – or a tool to improve efficiency and user experience, such as maintaining a consistent tone and topic history over multiple interactions with a conversational assistant (Lasecki et al., 2013). For both of these cases, the context shared between the crowd might be *explicit* content that must be read by each worker, or *implicit* in the observations of prior events.

### 6.1. Explicit

The most direct means of maintaining context between multiple interactions is to show relevant information to later workers. This is conceptually simple, but presents a number of challenges: how to prevent this information from becoming overwhelming to process over many interactions? How do we determine what is most important or worth passing on? How can we prevent creating and consuming this information from adding latency to our task?

Note taking is a widely used tool for remembering key details and pieces of information for later. Many strategies are evolved on an individual basis for best tracking this information (Van Kleek et al., 2009). However, scaling this to a collective, where different people might be more adept at certain strategies, and there is no one single overseer who understands all aspects of the problem in order to judge what is most important.

Existing crowd-powered systems have seen successful approaches based around special purpose note-taking systems (Lasecki et al., 2013). Automated systems might also act to augment the crowd's ability to remember and process large sets of information over long periods of time.In a puzzle-solving setting, message passing between workers has also been observed as a means of preventing thrashing between states, leading to more efficient solutions (Zhang et al., 2012). If the system has enough information about the task, it may be able to explicitly direct workers to hand off certain information to future workers (even when the information cannot be extracted automatically), such as with expert crowds in flash teams, where links between modular tasks define deliverables for use by future teams, and parallel tasks require more open-ended coordination and message passing during the task execution (Retelny et al., 2014).

However, this is an understudied problem. Few crowd-powered systems exist that meaningfully support the same user over multiple sessions. The most important elements to consider are:

– **What is important?** Predicting if a fact will be important is within the capability of workers in many (but not all) situations. Forecasting future events is even an active area of research in crowdsourcing (Forlines et al., 2014).
– **Who will record the information?** Crowd-curation can involve a separate task and workflow, but can yield better results than automated summaries. By combining human and automated approaches, such as having people identify what is important and having the machine summarize textual information, the cognitive effort and time required for human workers can be reduced.
– **How much can a worker process quickly?** This varies significantly per-task, but it is important to consider the latency impact of having to process and understand the information captured about prior sessions. Often, this load can be distributed over time, or even included as part of the existing tutorial phases to reduce the direct overhead.
– **How will the task be structured?** Depending on the task and type of information that needs to be recorded, the same workforce might be used to complete the core task and record information, or two separate groups might be used. Separating the task makes the problem of increasing latency and worker cognitive load more manageable, but increases operating costs.

## 6.2. Implicit

Implicit context might be shared by direct observation of other workers, or the results of their actions. It relies heavily on workers being willing to observe and assess their surroundings for clues to prior events that have an impact on their potential responses. While this may seem impractical for microtask workers who must complete tasks at a quick pace to earn a livable wage, prior work has observed workers waiting to see what others do before proceeding to propose their own answers in the presence of others (Lasecki et al., 2012). Crowd workers also remember some task details between different sessions spanning hours or days, so providing incentives for returning can afford some level of consistency between sessions (Lasecki et al., 2012; Bigham and Lasecki, 2014).

Explicit context maintenance is easier to design for than implicit mechanisms in most cases, because the tools used usually have more predictable uses and outcomes. However, this comes at the cost of increased latency to interact with the mechanism, or increased cost for additional workers to complete tasks in parallel. Thus, if implicit cues can be used, they can frequently provide a much smoother and less costly means of transferring information.

## 7. COMMUNICATION CHANNELS

Crowd-curated context can be seen as a case of one-way (potentially mediated) worker-to-worker communication with large time delays. As discussed in the previous section, communication can be very helpful for communicating context and other task needs to later workers. In this section, we explore how communication during a task can help tasks get completed quicker, and more reliably.

Communication in collaborative settings has been well studied in management theory, organizational behavior and psychology, human-computer interaction, and other fields. However, within crowdsourcing, the collaborative nature of tasks is often overlooked, perhaps because of a combination of the lack of native platform support (on almost any existing crowd marketplace) and the potential for collusion between workers to inhibit the wisdom-of-crowds effect.

### 7.1. Design Implications of Communication

Allowing workers to communicate with one another can be beneficial in multiple ways. Since workers are generally well-intentioned, but sometimes do not understand specific task rules, guidelines, or objective, other workers can help increase adherence to these rules and actually improve the performance of workers who might otherwise contributed noise (Lasecki et al., 2013). Extending this idea to a formal feedback framework has been shown to be effective at helping workers improve over time (Dow et al., 2012) – an important consideration when systems will be deployed and rely on a perpetual crowd workforce.

Communication can also allow workers to build on one another's work so far, and prevents conflicts from arising between workers. This can be of benefit in settings such as the collaborative search process in Chorus: workers could each claim a specific portion of a task, and then focus on that sub-task without worrying that another worker might complete it before them and nullify their contribution. The tradeoff here is that this could reduce answer speed and diversity if only one user searched for an answer, but reduces the potential for uncertain payoff and redundant responses to some aspects of a task while starving others. In an ideal case, communication can be used to effectively parallel parts of the task, while helping workers extend each other's efforts.

However, allowing for direct communication also has drawbacks. Workers might collude with one another to increase their odds of payment when majority decisions are used, or more vocal workers might persuade others to agree with their answer without regard to its correctness. An additional problem is well known to online platforms of all types: spammers.

Paralleling our discussion of context, we will look at creating both explicit and implicit channels for worker communication in this section.

### 7.2. Explicit

Creating explicit communication channels can range from text to audio or video, though the latter is far more rare because it's harder to host and mediate. This leaves text as the dominant form of direct communication behind most any crowd-powered system. By creating an explicit channel for communication, the worker experience can be more carefully designed, and workers will have a known resource to look to, rather than having to go without support or miss discovering an implicit communication channel.

To address the drawbacks stemming from direct communication, workers can be split into sub-groups. For tasks that would otherwise work without communication, this provides workers with a support group, while not biasing the entire crowd by any one individual. Similarly, it provides for speed and diversity in sub-tasks by mixing workers from different groups, while also ensuring that starvation is minimized since workers within a sub-group will be able to avoid conflict. To prevent spamming within sub-groups, rate limiting or feedback from other users can be used to deter over-posting.

In settings where crowds large enough to be divided into effective sub-groups are not practical, other mediation strategies can be used to prevent collusion and bias. For example, random spot-checking responses can provide confirmation that workers are giving helpful, differentiated answers. Spot-checking uses the crowd to act as a force-multiplier for the user, rather than fully-autonomous control (discussed more in the next section). In some cases, tasks with known gold-standard answers can also be inserted into a workflow to detect when workers might be cooperating to the detriment of quality. If this is observed, then communication might need to be reduced between workers in favor of more indirect means.

Not all channels of direct communication need be intentional. Chorus observed workers effectively using 'proposed' messages (which are only visible to other workers) to chat amongst one another without forwarding the response to users by voting on them. This was used for good (e.g., helping other workers understand the task) in some cases, and to the determent of the system in others (e.g., spammers). Interestingly though, this method of communication required workers to coordinate and trust one another enough to not forward the internal messages to users, and to actually pay a small cost (in terms of lost bonus) to propose an answer they knew would not be voted through. This suggests that potential mechanisms for incentivizing workers to contribute only the most important comments to one another may be effective.

## 7.3. Implicit

Implicit communication channels do not allow direct worker-to-worker contact, but instead display some feedback to workers about what others are doing. Implicit communication channels can frequently be overlooked when designing a system, or even might be impossible to remove. For example, Legion explored a visual feedback mechanism in a robot control task that let workers know what others in the crowd had done in the previous time-step if it differed from theirs. However, this quickly resulted in workers all trying to follow the crowd's previous decision, leading to many experimental trials ending in a longer series of repeated actions (such as spinning in circles).

On the other hand, using only the observation of the outcome of the task (instead of the exact input), which is seen by workers in the movements of the robot in the video and is essentially impossible to completely remove, led to a learned group behavior that let the crowd collective control the robot for over an hour without forgetting the intended path, despite the high level of turnover in the crowd and the lack of instructions to anyone but the first small set of workers who had begun the task.

Indirect cues such as showing the crowd's final response are not without their use however. Showing aggregate information about the crowd's collective decision also provides a means of allowing more adept workers a chance to compensate for the actions of inattentive or lazy workers (Lasecki and Bigham, 2012a). This also requires an attention to the incentives provided to workers. Even more subtle, indirect forms of feedback include displaying results from workers (individual or aggregate) to other workers (Lasecki et al., 2011, 2013, 2014).

### 7.4. Theoretical Aspects of Communication

Communication problems can be roughly divided into two categories: collusion and coordination. Collusion in the form of voter *manipulation* has been widely studied in social choice theory. See (Faliszewski et al., 2010) for a discussion of this and related topics.

Coordination is when the workers collaborate on a shared goal and individually have incomplete information about the situation, but can benefit from sharing information. We can characterize such problems in terms of Markov decision problems and their extensions to multiagent settings. When each worker is aware of all the observations and actions available to all the other workers, the variant is called a *multiagent POMDP* (MPOMDP), otherwise, it is a *decentralized POMDP* (Dec-POMDP). In both cases, workers are assumed to share a single reward and thus benefit from some communication. To model the case where each worker has a distinct reward, partially-observable stochastic games (POSGs) are used. While workers in practice almost always have individual incentives for participating, when studying the benefits of communication from a theoretical perspective, it is common to focus on collective benefits. We discuss the problem of structuring incentives to get desired outcomes from the workers with selfish motives in section 8.

The basic question asked of POMDP models is: what are the optimal or equilibrium policies? Determining an optimal policy in this case is NEXP-hard for Dec-POMDPs. To address this computational complexity barrier, the usual approach is to adopt the same approach taken to heuristically solve POMPDPs. Often, to improve the tractability of the problem, additional restrictions are assumed, say, in terms of *coordination graphs* (Proper and Tadepalli, 2009) or branch-and-bound heuristics (Spaan and Oliehoek, 2012). See also (Messias et al., 2011; Goldman and Zilberstein, 2004; Boutilier, 1999).

## 8. REWARDING WORKER CONTRIBUTIONS

The ultimate goal of a reward/penalty scheme is to provide workers with incentives that guarantee a desired level of effort, quality, or task completion time. However, in practice it is often hard to evaluate the quality of the work or to know what truly motivates workers. If there was a right answer to compare to, then there would be no need for worker input.

A lot of work has been done in the area of studying how to incentivize work in different scenarios. The literature in crowdsourcing has also followed suit. However, many of the most effective incentive mechanisms rely on task configurations that are either too difficult or require too much time to understand, or are not compatible with real-time systems in their original implementation. We discuss strategies that can be used in real-time to reward helpful work.

### 8.1. Payment Limitations

One issue that reduces flexibility in reward mechanisms is the limitation of the crowd platform itself. For example, Mechanical Turk (as well as most others) does not provide a means of marking a task as “completed in good faith, but incorrect”. This means that workers who try and fail to get a correct answer can only be either rejected or paid the full amount of the task without feedback. This leads to a culture where requesters generally just pay for completed tasks regardless of quality, and rejections are seen as an exceptionally strong response to incorrect work.

From a payment side, this can be partially mitigated using bonuses, if supported by the platform – a base payment for taking the task and genuinely trying to provide a valid answer, and a bonus can be paid for higher quality work. This also allows for much more flexibility in the incentive mechanisms that can be used. In this section, we will assume that the platform being used allows for delayed rewards.

## 8.2. Design Implications of Reward Strategies

As we discuss reward strategies (incentive mechanisms) it is important to keep in mind that these payments are not just a means of motivating workers to do good work – they often do that even in the face of poorly designed rewards – but also to provide workers fair pay for the effort they have put into the task. The key is knowing when this effort has really been put in, and how much to pay.

"Correctness" is typically determined in one of two ways: either by using agreement between a sufficient number of workers, or using gold-standard tasks to periodically inspect worker responses. While these methods often include some delays in order to determine the right answer, heuristics can be used to provide approximate immediate feedback to workers that they are on the right path, even if the result does not turn out to be perfect. In Scribe, automatic pre-checks (i.e., spell checking) and agreement are used to give immediate feedback as to whether the system thinks the user is correct in typing a word, even before the system is really confident in the final answer. Delayed rewards (bonuses) can then be used to compensate for the cases when the heuristic fails to reward a worker for helpful input immediately.

To maintain speed, microtasks (as with any fixed-value work) push workers to balance between speed and efficiency – spending more time on a task increases the chances that it will be considered well-done, while spending less time increases the hourly rate of the task and allows for higher overall income. To maximize economic value, the worker's goal then becomes to do the task as quickly as it can be done correctly. The problem lies in the worker's ability to estimate all of the factors in this balance, which is difficult even if all of the information required has been made available to the worker.

HiveMind (Singh et al., 2012) is an incentive mechanism that used this basic balance to vary the rate at which answers are elicited by rewarding early answers relative to later ones by a larger or smaller factor, and rewarding answers more than votes of agreement. Larger payment for early answers relative to late answers leads workers to contribute fewer answers, and agree with existing answers more often. More even payments over time lead workers to continue to contribute new answers for the chance to earn a higher pay. The requester can thus adjust the rewards depending on what is of higher value to the system.

Another frequent, and perhaps more fundamental issue is that the reward mechanisms being used are not clear to workers, meaning that workers cannot correctly account for them when considering what actions to take. To make them clear, strategies with direct mappings to visible actions should be used. If behind-the-scenes information has to be used to compute the reward (e.g., basing rewards on the input of other workers), some indication of this should be shown. Explanations of the process used, as well as tutorials that allow workers to experiment with how rewards are given in practice have also proven to be effective ways to help workers understand the task and its rewards better (Lasecki et al., 2013,b, 2014).

## 8.3. Joint Rewards

Joint rewards are needed when the collective behavior cannot just be the sum of individually optimized behaviors. For example, encouraging workers to give the correct answers to a navigation problem can be done using individual rewards (because we assume we can compare to other workers), but using self-correction (Lasecki and Bigham, 2012a) requires a joint success reward to see if workers were able to collectively able to correct for the group action (discussed in Section 7).

Another example of joint rewards is meta-rewarding. For instance, Legion used a global time limit with a reward multiplier for faster times. By creating a two-tier optimization, workers can be encouraged to agree and rewarding them for the time they spent on the task, while also rewarding the group for finishing faster.

More generally, rewards help to provide workers with not only a wage, but a means of getting feedback on their performance (even if that feedback is often very noisy, such as on Mechanical Turk). There are a number of non-monetary ways to provide feedback in real-time settings, many of which fall under the umbrella of gamification.

## 8.4. Theoretical Models

The most common theory for reasoning about such problems is *mechanism design*. Game theory asks what the best strategy is for an individual to take when the outcome depends on how the other individuals (who themselves are acting strategically) play, whereas mechanism design asks how to best design the game so that the workers' optimal strategies lead to the outcomes that the designer desires. It is most commonly used in the context of auctions, and there is a substantial body of work that looks at crowdsourcing from this perspective (DiPalantino and Vojnovic, 2009; Boudreau et al., 2011; Moldovanu and Sela, 2001, 2006; Archak and Sundararajan, 2009; Chawla et al., 2012). However, this work does not focus on real-time settings, and generally assumes that the response timescale is much longer and that finding high-quality solutions is a matter of time and effort. Also, such settings typically involve a single round. In real-time settings, workers have little time to respond, meaning one must regard the knowledge and beliefs of the workers as relatively static. A real-time task often involves multiple rounds.

Models in which the actions, observations, and rewards of multiple actors are explicitly defined are called partially observable stochastic games (POSGs) (Kuhn, 1953). In POSGs there may be no well-defined globally optimal (i.e., dominant) joint strategy. Instead, strategic equilibria must be found, but it can be NEXP-hard (Bernstein et al., 2002) to compute these, even offline and with complete knowledge of the model. TurKontrol (Dai and Weld, 2011) used a POMDP model in order to optimize iterative task completion online.

## 9. ADDING AUTOMATION

Thus far, we have discussed how to create systems that use computer-mediated groups of people (crowds) to go beyond what automated systems can do by themselves. However, beyond mediating interactions between workers and routing tasks, there are many automated solutions that target these problems. Their solutions may not work in a wide enough variety of settings to create the desired reliability, or may only give partial solutions, but they can be integrated into crowd-powered systems as a means of making human workers' tasks easier.

## 9.1. Design Implications of Automated Assistance

Crowdsourcing can be seen as a *scaffold* for intelligent systems: a way to support systems in situations where they cannot operate independently. This provides users with a seamless experience, while providing a means of training the automated system to be more capable over time (learning by demonstration (Osentoski, Crick, Jay, and Jenkins, Osentoski et al.; Lasecki et al., 2011).

Computers can also go beyond the limitations of people. While people are great sources of general-purpose processing – able to understand context and disambiguate situations using many different facets of a problem – they are not nearly as fast as computers at well-defined data-processing tasks. Using computers to contribute information, such as search results or patterns found in a dataset, allows groups containing both human and machine contributors to outperform either individually. These *hybrid* systems leverage the fact that the collaboration methods used in crowdsourcing are often flexible enough to handle automated systems as just another set of workers, with a different set of mistakes and biases.

Automating systems over time also provide a compelling argument for the scalability of crowd-powered systems: people are sources of "live" training data for future fully-automated systems. With this in mind, the input mechanisms for and workers interfaces can be designed in such a way that they elicit information that not only answers the question posed, but also provides a formal enough notion of the information for computers to parse. For example, ARchitect (Lasecki et al., 2014) extended Legion:AR (Lasecki et al., 2013a) with an interface that let workers convey the dependency structure of a set of actions by simply answering a set of yes-no questions. This allows the system to generalize observed and labeled activities the crowd produces and helps the system to learn accurate patterns with fewer examples.

## 9.2. Roles of Automation

To help provide a framework for integrating automated systems into crowd-powered systems, we highlight the following potential roles that automation can commonly fill:

### *9.2.1. Support*

One of the easiest means of integrating automated input into a system's workflow is to use AI as a "first pass". By producing an initial guess automatically, many problems can potentially be solved immediately, and at most use worker input to confirm their accuracy. This reduces the time and effort that workers must contribute, thus making the system cheaper to run. However, editing large responses can itself become more challenging than simply creating a new response if the error rate is high enough. As such, when no algorithm is available that can achieve high accuracy consistently, or has a reliable confidence metric for selecting the cases where error is expected to be low, then it might be more effective to abstain from providing this input altogther.

#### 9.2.2. Augment

Even when approaches are not available to wholly solve any instances of the problem facing the system, automated systems can be made available as support tools. Web search tools can help workers access information, even before they explicitly request it, and prediction systems can make response suggestions based on other workers' input. Even simple tools such as spellcheck can improve the quality and speed of responses significantly.

#### 9.2.3. Guide

Beyond directly helping workers with completing their task, automated systems can be used to highlight portions of the task that need the most attention. This might be based on features within the task instance itself, based on other workers' input to the same task, or based on aggregate information about previously observed work. This type of guidance could also be provided by a human worker, but the cost of giving a worker context and the higher response latency of a human contributor typically makes an automated approach a more viable choice. Note the distinction between support via in-task guidance, and task-level routing (discussed in Section 4) where tasks are assigned to workers based on system needs and, if known, the workers' skill sets.

#### 9.2.4. Complete Tasks

Automated systems can also be used to entirely solve some tasks without involving a human at all. This use-case is tightly tied with the task routing method being used, which must have a means of assessing the capability of the automated system relative to the task it is faced with. If the systems' confidence is high enough that the problem can be solved automatically, then the task is not routed to a human worker. When it is not, the system employs an active learning approach to get answers from workers. We include this role here because it relies on the idea that automated systems can fill equivalent roles as people and have work selectively routed to them when they are capable. The capabilities of each AI can vary (just as with people), and the routing algorithms used at the hybrid system level should be able to remain almost entirely unchanged.

New methods for combining these roles (and more) continue to be created. The exact ideal mix depends on the task at hand, the capabilities of current automated approaches, and the cost trade-off between using human input and developing a custom automated solution.

## 10. CONCLUSION

In this paper, we have discussed an architecture for crowd-powered intelligent systems that can respond to users in real time. For each component in the architecture, we outline key approaches from relevant literature and anecdotal experience, design implications and trade-offs for each approach, and examples of systems that have effectively used the approach. We then ground our discussion in the models underpinning these components in order to provide more context for future work to build upon.

As with any design space, a complete set of options cannot be enumerated. Instead, our goal is to inform system builders who wish to augment interactive systems with human intelligence from the crowd about how to design systems in this complex space that is only beginning to be explored.

## 11. ACKNOWLEDGEMENTS

This work has been supported by the National Science Foundation (under awards #IIS-1149709, #IIS-1218209, and #SES-1111016), Google, Yahoo, a Sloan Foundation Fellowship, and a Microsoft Research Ph.D. Fellowship.

The authors would like to thank all those who have contributed to this line work both via tool creation and insightful discussion, including Christopher Miller, Mitchell Gordon, Donato Borello, Rachel Wesley, Ece Kamar, Matthew Murphy, and others.

## 12. REFERENCES

Archak, N and Sundararajan, A. (2009). Optimal Design of Crowdsourcing Contests.. In *ICIS*. 200.

Arrow, K. J. (2012). *Social choice and individual values*. Vol. 12. Yale university press.

Bernstein, D. S, Givan, R, Immerman, N, and Zilberstein, S. (2002). The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

Bernstein, M. S, Brandt, J, Miller, R. C, and Karger, D. R. (2011). Crowds in Two Seconds: Enabling Realtime Crowd-powered Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 33–42. DOI:http://dx.doi.org/10.1145/2047196.2047201

Bernstein, M. S, Karger, D. R, Miller, R. C, and Brandt, J. (2012). Analytic Methods for Optimizing Realtime Crowdsourcing. *CoRR* abs/1204.2995 (2012).

Bernstein, M. S, Little, G, Miller, R. C, Hartmann, B, Ackerman, M. S, Karger, D. R, Crowell, D, and Panovich, K. (2010). Soylent: A Word Processor with a Crowd Inside. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 313–322. DOI:http://dx.doi.org/10.1145/1866029.1866078

Bigham, J. P, Jayant, C, Ji, H, Little, G, Miller, A, Miller, R. C, Miller, R, Tatarowicz, A, White, B, White, S, and Yeh, T. (2010). VizWiz: Nearly Real-time Answers to Visual Questions. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 333–342. DOI:http://dx.doi.org/10.1145/1866029.1866080

Bigham, J. P and Ladner, R. E. (2011). What the Disability Community Can Teach Us About Interactive Crowdsourcing. *interactions* 18, 4 (July 2011), 78–81. DOI:http://dx.doi.org/10.1145/1978822.1978838

Bigham, J. P and Lasecki, W. S. (2014). Crowd Storage: Storing Information on Existing Memories. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 601–604. DOI:http://dx.doi.org/10.1145/2556288.2557159

Borish, M, Cordar, A, Foster, A, Kim, T, Murphy, J, and Lok, B. (2014). Utilizing Real-time Human-Assisted Virtual Humans to Increase Real-world Interaction Empathy. In *Kansei Engineering & Emotion Research (KEER '14)*. 15.

Boudreau, K. J, Lacetera, N, and Lakhani, K. R. (2011). Incentives and problem uncertainty in innovation contests: An empirical analysis. *Management Science* 57, 5 (2011), 843–863.

Boutilier, C. (1999). Sequential optimality and coordination in multiagent systems. In *IJCAI*, Vol. 99. 478–485.

Boutsis, I and Kalogeraki, V. (2013). Crowdsourcing under Real-Time Constraints. In *Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on*. 753–764. DOI:http://dx.doi.org/10.1109/IPDPS.2013.84

Callison-Burch, C. (2009). Fast, Cheap, and Creative: Evaluating Translation Quality Using Amazon's Mechanical Turk. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1 - Volume 1 (EMNLP '09)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 286–295.

Carpenter, B. (2011). A Hierarchical Bayesian Model of Crowdsourced Relevance Coding.. In *TREC*.

Chawla, S, Hartline, J. D, and Sivan, B. (2012). Optimal crowdsourcing contests. In *SODA*. 856–868.

Chilton, L. B, Sims, C. T, Goldman, M, Little, G, and Miller, R. C. (2009). Seaweed: A Web Application for Designing Economic Games. In *Proceedings of the ACM SIGKDD Workshop on Human Computation (HCOMP '09)*. ACM, New York, NY, USA, 34–35. DOI:http://dx.doi.org/10.1145/1600150.1600162

Cooper, S, Khatib, F, Treuille, A, Barbero, J, Lee, J, Beenen, M, Leaver-Fay, A, Baker, D, Popović, Z, and others, . (2010). Predicting protein structures with a multiplayer online game. *Nature* 466, 7307 (2010), 756–760.

Dai Peng, M and Weld, D. S. (2011). Artificial intelligence for artificial artificial intelligence. In *Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2011)*.

Dalvi, N, Dasgupta, A, Kumar, R, and Rastogi, V. (2013). Aggregating crowdsourced binary ratings. In *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 285–294.

DiPalantino, D and Vojnovic, M. (2009). Crowdsourcing and all-pay auctions. In *Proceedings of the 10th ACM conference on Electronic commerce*. ACM, 119–128.

Dow, S, Kulkarni, A, Klemmer, S, and Hartmann, B. (2012). Shepherding the Crowd Yields Better Work. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1013–1022. DOI:http://dx.doi.org/10.1145/2145204.2145355

Faliszewski, P, Hemaspaandra, E, and Hemaspaandra, L. A. (2010). Using complexity to protect elections. *Commun. ACM* 53, 11 (2010), 74–82.

Forlines, C, Miller, S, Guelcher, L, and Bruzzi, R. (2014). Crowdsourcing the Future: Predictions Made with a Social Network. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 3655–3664. DOI:http://dx.doi.org/10.1145/2556288.2556967

Goldman, C. V and Zilberstein, S. (2004). Decentralized control of cooperative systems: Categorization and complexity analysis. *J. Artif. Intell. Res.(JAIR)* 22 (2004), 143–174.

Harrington, R. P and Vanderheiden, G. C. (2013). Crowd Caption Correction (CCC). In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*. ACM, New York, NY, USA, Article 45, 2 pages. DOI: http://dx.doi.org/10.1145/2513383.2513413

Heidari, H and Kearns, M. (2013). Depth-Workload Tradeoffs for Workforce Organization. In *First AAAI Conference on Human Computation and Crowdsourcing*.

Hemaspaandra, E, Hemaspaandra, L. A, and Rothe, J. (2012). Online Voter Control in Sequential Elections.. In *ECAI*. 396–401.

Hemaspaandra, E, Hemaspaandra, L. A, and Rothe, J. (2014). The complexity of online manipulation of sequential elections. *J. Comput. System Sci.* 80, 4 (2014), 697–710.

Ho, C.-J, Jabbari, S, and Vaughan, J. W. (2013). Adaptive task assignment for crowdsourced classification. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 534–542.

Ho, C.-J and Vaughan, J. W. (2012). Online Task Assignment in Crowdsourcing Markets.. In *AAAI*.

Horton, J. J and Chilton, L. B. (2010). The Labor Economics of Paid Crowdsourcing. In *Proceedings of the 11th ACM Conference on Electronic Commerce (EC '10)*. ACM, New York, NY, USA, 209–218. DOI:http://dx.doi.org/10.1145/1807342.1807376

Ipeirotis, P. G. (2010). Analyzing the amazon mechanical turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students* 17, 2 (2010), 16–21.

Jung, H. J and Lease, M. (2013). Crowdsourced Task Routing via Matrix Factorization. *CoRR* abs/1310.5142 (2013).

Kamar, E, Hacker, S, and Horvitz, E. (2012). Combining Human and Machine Intelligence in Large-scale Crowdsourcing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS '12)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 467–474.

Kamar, E and Horvitz, E. (2012). Incentives for truthful reporting in crowdsourcing. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems, 1329–1330.

Karger, D. R, Oh, S, and Shah, D. (2011). Iterative learning for reliable crowdsourcing systems. In *Advances in neural information processing systems*. 1953–1961.

Kittur, A, Nickerson, J. V, Bernstein, M, Gerber, E, Shaw, A, Zimmerman, J, Lease, M, and Horton, J. (2013). The Future of Crowd Work. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW '13)*. ACM, New York, NY, USA, 1301–1318. DOI:http://dx.doi.org/10.1145/2441776.2441923

Kittur, A, Smus, B, Khamkar, S, and Kraut, R. E. (2011). CrowdForge: Crowdsourcing Complex Work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 43–52. DOI:http://dx.doi.org/10.1145/2047196.2047202

Kuhn, H. (1953). Extensive Games and the Problem of Information, Contributions to the Theory of Games II, Kuhn, HW and AW Tucker, eds. 193-216. (1953).

Kulkarni, A, Can, M, and Hartmann, B. (2012). Collaboratively Crowdsourcing Workflows with Turkomatic. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work (CSCW '12)*. ACM, New York, NY, USA, 1003–1012. DOI: http://dx.doi.org/10.1145/2145204.2145354

Lasecki, W and Bigham, J. (2012)a. Self-correcting Crowds. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems (CHI EA '12)*. ACM, New York, NY, USA, 2555–2560. DOI:http://dx.doi.org/10.1145/2212776.2223835

Lasecki, W, Miller, C, Sadilek, A, Abumoussa, A, Borrello, D, Kushalnagar, R, and Bigham, J. (2012). Real-time Captioning by Groups of Non-experts. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 23–34. DOI:http://dx.doi.org/10.1145/2380116.2380122

Lasecki, W. S and Bigham, J. P. (2012)b. Online Quality Control for Real-time Crowd Captioning. In *Proceedings of the 14th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '12)*. ACM, New York, NY, USA, 143–150. DOI: http://dx.doi.org/10.1145/2384916.2384942

Lasecki, W. S, Gordon, M, Koutra, D, Jung, M, Dow, S, and Bigham, J. P. (2014). Glance: Rapidly Coding Behavioral Video with the Crowd. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 12.

Lasecki, W. S, Kamar, E, and Bohus, D. (2013)a. Conversations in the Crowd: Collecting Data for Task-Oriented Dialog Learning. In *First AAAI Conference on Human Computation and Crowdsourcing*.

Lasecki, W. S, Miller, C. D, and Bigham, J. P. (2013)b. Warping Time for More Effective Real-time Crowdsourcing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2033–2036. DOI: http://dx.doi.org/10.1145/2470654.2466269

Lasecki, W. S, Murray, K. I, White, S, Miller, R. C, and Bigham, J. P. (2011). Real-time Crowd Control of Existing Interfaces. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. ACM, New York, NY, USA, 23–32. DOI:http://dx.doi.org/10.1145/2047196.2047200

Lasecki, W. S, Song, Y. C, Kautz, H, and Bigham, J. P. (2013)a. Real-time Crowd Labeling for Deployable Activity Recognition. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work (CSCW '13)*. ACM, New York, NY, USA, 1203–1212. DOI:http://dx.doi.org/10.1145/2441776.2441912

Lasecki, W. S, Thiha, P, Zhong, Y, Brady, E, and Bigham, J. P. (2013)b. Answering Visual Questions with Conversational Crowd Assistants. In *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility (ASSETS '13)*. ACM, New York, NY, USA, Article 18, 8 pages. DOI:http://dx.doi.org/10.1145/2513383.2517033

Lasecki, W. S, Weingard, L, Ferguson, G, and Bigham, J. P. (2014). Finding Dependencies Between Actions Using the Crowd. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 3095–3098. DOI:http://dx.doi.org/10.1145/2556288.2557176

Lasecki, W. S, Wesley, R, Nichols, J, Kulkarni, A, Allen, J. F, and Bigham, J. P. (2013). Chorus: A Crowd-powered Conversational Assistant. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*. ACM, New York, NY, USA, 151–162. DOI:http://dx.doi.org/10.1145/2501988.2502057

Lasecki, W. S, White, S, Murray, K. I, and Bigham, J. P. (2012). Crowd Memory: Learning in the Collective. In *Proceedings of Collective Intelligence*. 8.

Lee, J, Kladwang, W, Lee, M, Cantu, D, Azizyan, M, Kim, H, Limpaecher, A, Yoon, S, Treuille, A, and Das, R. (2014). RNA design rules from a massive open laboratory. *Proceedings of the National Academy of Sciences* 111, 6 (2014), 2122–2127.

Limpaecher, A, Feltman, N, Treuille, A, and Cohen, M. (2013). Real-time Drawing Assistance Through Crowdsourcing. *ACM Trans. Graph.* 32, 4, Article 54 (July 2013), 8 pages. DOI:http://dx.doi.org/10.1145/2461912.2462016

Lin, C. H, Weld, D. S, and others, . (2012). Dynamically switching between synergistic workflows for crowdsourcing. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Little, G, Chilton, L. B, Goldman, M, and Miller, R. C. (2010). TurKit: Human Computation Algorithms on Mechanical Turk. In *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology (UIST '10)*. ACM, New York, NY, USA, 57–66. DOI:http://dx.doi.org/10.1145/1866029.1866040

Liu, Q, Peng, J, and Ihler, A. (2012). Variational inference for crowdsourcing. In *Advances in Neural Information Processing Systems*. 692–700.

Loparev, A, Lasecki, W. S, Murray, K. I, and Bigham, J. P. (2013). Introducing Shared Character Control to Existing Video Games. (2013).

Mao, A, Chen, Y, Gajos, K. Z, Parkes, D, Procaccia, A. D, and Zhang, H. (2012). TurkServer: Enabling Synchronous and Longitudinal Online Experiments. In *Fourth Workshop on Human Computation (HCOMP 2012)*.

Mao, A, Kamar, E, and Horvitz, E. (2013). Why Stop Now? Predicting Worker Engagement in Online Crowdsourcing. In *First AAAI Conference on Human Computation and Crowdsourcing*.

Mashhadi, A. J and Capra, L. (2011). Quality control for real-time ubiquitous crowdsourcing. In *Proceedings of the 2nd international workshop on Ubiquitous crowdsouring*. ACM, 5–8.

Mason, W and Suri, S. (2012). Conducting behavioral research on AmazonâĂŹs Mechanical Turk. *Behavior research methods* 44, 1 (2012), 1–23.

Messias, J. V, Spaan, M, and Lima, P. U. (2011). Efficient offline communication policies for factored multiagent POMDPs. In *Advances in Neural Information Processing Systems*. 1917–1925.

Michelucci, P. (2013). Synthesis and Taxonomy of Human Computation. In *Handbook of Human Computation*. Springer, 83–86.

Moldovanu, B and Sela, A. (2001). The optimal allocation of prizes in contests. *American Economic Review* (2001), 542–558.

Moldovanu, B and Sela, A. (2006). Contest architecture. *Journal of Economic Theory* 126, 1 (2006), 70–96.

Osentoski, S, Crick, C, Jay, G, and Jenkins, O. C. Crowdsourcing for closed loop control. (????).

Parkes, D. C and Procaccia, A. D. (2013). Dynamic Social Choice with Evolving Preferences. (2013), 767–773.

Proper, S and Tadepalli, P. (2009). Solving multiagent assignment markov decision processes. In *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems, 681–688.

Quinn, A. J and Bederson, B. B. (2011). Human Computation: A Survey and Taxonomy of a Growing Field. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, New York, NY, USA, 1403–1412. DOI: http://dx.doi.org/10.1145/1978942.1979148

Raykar, V. C, Yu, S, Zhao, L. H, Valadez, G. H, Florin, C, Bogoni, L, and Moy, L. (2010). Learning from crowds. *The Journal of Machine Learning Research* 11 (2010), 1297–1322.

Retelny, D, Robaszkiewicz, S, To, A, Lasecki, W, Patel, J, Rahmati, N, Doshi, T, Valentine, M, and Bernstein, M. (2014). Expert Crowdsourcing with Flash Teams. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. ACM, New York, NY, USA, 12.

Rossen, B and Lok, B. (2012). A crowdsourcing method to develop virtual human conversational agents. *International Journal of Human-Computer Studies* 70, 4 (2012), 301 – 319. DOI:http://dx.doi.org/10.1016/j.ijhcs.2011.11.004

Russell, B. C, Torralba, A, Murphy, K. P, and Freeman, W. T. (2008). LabelMe: A Database and Web-Based Tool for Image Annotation. *Int. J. Comput. Vision* 77, 1-3 (May 2008), 157–173. DOI:http://dx.doi.org/10.1007/s11263-007-0090-8

Sheng, V. S, Provost, F, and Ipeirotis, P. G. (2008). Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 614–622.

Singh, P, Lasecki, W. S, Barelli, P, and Bigham, J. P. (2012). *HiveMind: A Framework for Optimizing Open-Ended Responses From the Crowd*. Technical Report. URCS Technical Report.

Snow, R, O'Connor, B, Jurafsky, D, and Ng, A. Y. (2008). Cheap and Fast—but is It Good?: Evaluating Non-expert Annotations for Natural Language Tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08)*. Association for Computational Linguistics, Stroudsburg, PA, USA, 254–263.

Spaan, M. T and Oliehoek, F. A. (2012). Tree-Based Solution Methods for Multiagent POMDPs with Delayed Communication. In *Proc. of 24th Benelux Conference on Artificial Intelligence*. 319–320.

Surowiecki, J. (2005). *The wisdom of crowds*. Random House LLC.

Van Kleek, M. G, Bernstein, M, Panovich, K, Vargas, G. G, Karger, D. R, and Schraefel, M. (2009). Note to Self: Examining Personal Information Keeping in a Lightweight Note-taking Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1477–1480. DOI:http://dx.doi.org/10.1145/1518701.1518924

von Ahn, L. (2005). *Human Computation*. Ph.D. Dissertation. Carnegie Mellon University, Pittsburgh, PA.

von Ahn, L and Dabbish, L. (2004). Labeling Images with a Computer Game. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 319–326. DOI:http://dx.doi.org/10.1145/985692.985733

Wikipedia (2014). Words per minute. (2014). http://en.wikipedia.org/wiki/Words_per_minute Accessed: 2014-06-05.

Zhang, H, Law, E, Miller, R, Gajos, K, Parkes, D, and Horvitz, E. (2012). Human Computation Tasks with Global Constraints. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 217–226. DOI:http://dx.doi.org/10.1145/2207676.2207708

Zhou, D, Basu, S, Mao, Y, and Platt, J. C. (2012). Learning from the wisdom of crowds by minimax entropy. In *Advances in Neural Information Processing Systems*. 2195–2203.