HiveMind: A Framework for Optimizing Open-Ended Responses From the Crowd

Preetjot Singh[1,2]*, Walter S. Lasecki*[1]*, Paulo Barelli*[2]*, and Jeffrey P. Bigham*[1]

University of Rochester, Computer Science[1] / Economics[2]
Rochester, NY 14627 USA
{preetjot.singh,paulo.barelli}@rochester.edu
{wlasecki,jbigham}@cs.rochester.edu
Technical Report #978

ABSTRACT

Crowdsourcing has been effectively applied to many difficult problems beyond the capabilities of current automated systems. Many such problems are not only difficult to solve, but their solutions are also difficult to verify. Absent an evaluation metric for automatic verification, a common approach is to have crowd workers not only solve problems but also verify solutions using a collective intelligence model. Optimizing workers' responses in this situation presents difficulties, as does optimizing with respect to both speed and accuracy simultaneously, as one is a constraint on the other. In this paper, we introduce HiveMind, a game-theoretic model of collective-intelligence crowdsourcing that addresses response optimization at the individual worker level with respect to both speed *and* accuracy. In addition to inferring worker commitment level, HiveMind allows task creators to tradeoffs between low crowd participation and excess noise by tuning the level of convergence. We discuss how this model can be used to motivate workers for general continuous real-time tasks of unbounded length using a reputation system and how to identify consistent leaders in these domains. We also explore expert-answer elicitation which requires a significant degree of divergence to attain a set of more varied richer responses, and present a solution using a specialized aggregate function within HiveMind as a framework. This enables systems that seek to elicit different sets of results from crowds (each with their own idiosyncrasies) to all use a single framework.

INTRODUCTION

Recent crowd-powered systems have brought crowdsourcing to bear on problems that require accurate, real-time responses. VizWiz lets visually-impaired people ask the crowd questions about their visual environments (Bigham, Jayant, Ji, Little, Miller, Miller, Miller, Tatarowicz, White & Yeh 2010), Legion puts the crowd in control of existing desktop user interfaces (Lasecki, Murray, White, Miller & Bigham 2011), Scribe enables real-time captioning of speech by non-experts (Lasecki, Miller, Sadilek, AbuMoussa & Bigham 2012*a*), and Adrenaline lets the crowd interactively select the best frame from a video sequence for better photography (Bernstein, Brandt, Miller & Karger 2011). Such systems not only need to encourage workers to respond quickly but also accurately. HiveMind provides a formal theoretical model to address this problem.

Optimizing speed and accuracy is inherently difficult since one is a constraint on the other. We define a *collective intelligence* (CI) metric as the aggregate of the crowd's responses, instead of a static, externally defined benchmark. Using CI metrics in crowdsourcing systems make optimizing both dimensions additionally problematic since responses are evaluated by a fluid metric. Crowdsourcing tasks typically use CI metrics as these play the primary strength and purpose of crowdsourcing: CI tasks include image-labeling(von Ahn & Dabbish 2004*a*), checking websites for qualitative content such as bias or hate speech or just situationally-inappropriate content(Attenberg, Ipeirotis & Provost 2011, Aral, Ipeirotis & Taylor 2011), and relevance of search engine results(Alonso, Kazai & Mizzaro 2012). With HiveMind, we present a theoretical framework for crowdsourcing tasks that solves this problem by motivating workers to optimize their own individual responses with respect to both dimensions. Individual workers are optimally suited to evaluate their own abilities with respect to the task at hand, and hence establish their optimal time for an appropriately accurate answer. HiveMind consists of a selection mechanism that elicits the measure of a worker's commitment to a given task in order to select an optimal set of workers, and an optimizing aggregate mechanism that promotes fast, accurate responses that converge to a collective answer. The application of game theory and mechanism design on problems in crowdsourcing is not without precedent(Chawla, Hartline & Sivan 2012).

Using a CI metric also makes systems especially susceptible to the crowd's idiosyncrasies: Certain crowds could exhibit a greater-than-expected reticence to submit responses and may need encouragement, or you could have the reverse problem with excess noise, and may want the crowd to exercise greater reserve in responding. HiveMind's mechanisms allow task developers to solve these problems by choosing values on a continuum. This also allows developers to measure the effects of a single variable change in the same framework making comparison of different cases more accurate.

HiveMind handles both real-time and non-real-time tasks, and promotes the desired behavior in the crowd – to find a balance between the quality of the responses provided, and the time required to generate them. Additionally, we explore the problem of expert-answer-elicitation in CI systems which, by definition, have a significant democratic component and thus are inherently vulnerable to a *majority effect* where workers tend to veer their responses towards their notion of the collective answer. We solve this problem by using an aggregate mechanism that builds on Pivot, motivating 'expert' workers to fight this majority effect and submit their true responses even when they differ greatly from the collective answer. Finally, we discuss mechanisms that are able to better handle real-time and continuous crowdsourcing domains.

BACKGROUND

HiveMind is related to (i) crowd-powered systems that engage workers for longer periods of time (such as those using real-time work from the crowd) and (ii) game theoretic models that have been applied to crowdsourcing.

Human Computation

Human computation was introduced to integrate people into computational processes to solve problems too difficult for computers to solve alone, but has not been applied to real-time control problems. As convenient means of getting access to human computation, many approaches use *crowdsourcing*, which recruits several *workers* to contribute to the task. Crowdsourcing has been shown useful in writing and editing (Bernstein, Little, Miller, Hartmann, Ackerman, Karger, Crowell & Panovich 2010), image description and interpretation (Bigham et al. 2010, von Ahn & Dabbish 2004*b*), and protein folding (Cooper, Khatib, Treuille, Barbero, Lee, Beenen, Leaver-Fay, Baker, Popovic & Players 2010), among many other areas. Existing abstractions focus on obtaining quality work, and generally introduce redundancy and layering into tasks so that multiple workers contribute and verify results at each stage. For instance, guaranteeing reliability through answer agreement (von Ahn & Dabbish 2004*b*) or the find-fix-verify pattern of Soylent (Bernstein et al. 2010).

Several systems have explored how to make human computation interactive. As an example, VizWiz (Bigham et al. 2010) answers visual questions for blind people quickly. It uses quikTurkit to pre-queue crowds of workers from Amazon's Mechanical Turk so that they will be available when needed. For instance, the ESP Game encouraged accurate image labels by pairing players together and requiring them both to enter the same label, although ESP Game players could also be paired with simulated players (von Ahn & Dabbish 2004*b*). Seaweed reliably got Mechanical Turk workers to be available at the same time to play economic games by requiring the first worker to arrive to wait (generally for a few seconds) (Chilton 2009).

Recent crowd-powered systems target quick responses by pre-recruit workers who are then kept on standby until they are needed (Bigham et al. 2010, Bernstein et al. 2011), which reduces or eliminates the time to recruit workers. Latency is thus determined by how quickly workers choose to complete their task. While current models sacrifice accuracy for latency, HiveMind is designed to encourage optimization of both accuracy and latency.

Continuous Real-time Crowdsourcing

Another approach to soliciting real-time input is to maintain worker engagement in a task. This allows systems to benefit from repeated observation of the same workers and allows workers to exercise longer-term strategies. For instance, the Legion system connects crowd workers to existing desktop user interfaces via a real-time feedback loop for the duration of an interaction (Lasecki et al. 2011). Input from multiple individual workers is aggregated into a single control stream by using worker responses to select the 'best' answer in real-time. Legion has been used to reliably control a variety of interfaces requiring continuous real-time input, ranging from robot navigation to document editing. Similarly, Legion:Scribe uses multiple workers to perform a captioning tasks better than any constituent individual could have by synthesizing the workers partial captions into a single stream (Lasecki et al. 2012*a*). Legion generally motivates the crowd well, but requires tasks to be of fixed length because workers are only rewarded only when the task is completed. In this paper, tasks in HiveMind are framed as atomic, with a single collective answer per task aggregated from one input from each worker. However, HiveMind supports chaining atomic segments together in a continuous-task setting, which enables application to continuous real-time tasks. Segmenting of continuous tasks using the crowd has been performed by Legion:AR, using reliable groups of workers segmented and labeled a video for activity recognition (Lasecki, Song, Kautz & Bigham 2012*b*). HiveMind provides a means of motivating more general crowds for such tasks.

Game-theoretic Models

Mechanism design is an area of game theory that focuses on structuring *games* or situations that elicit specific responses from players (participants in the situation - workers, in our case) thus yielding desirable outcomes. The players are assumed to be *rational*, that is, players will always act to maximize their utility.

The Confidence mechanism in HiveMind accepts workers' bids as a measure of their commitment to the task [1]. Another game-theoretic approach to response-optimization is getting workers to increase their confidence levels by solving repeated tasks (Karger, Oh & Shah 2012).

HIVEMIND

HiveMind is a theoretical framework for crowdsourcing systems that ask workers to answer questions. A worker can agree with (vote for) another worker's proposed answer (hereafter referred to as playing *A*) or propose one of their own (hereafter referred to as playing *P*). We use a *collective intelligence* model where the system uses a voting function to evaluate responses and converge on a collective final answer - as opposed to using an external benchmark. HiveMind incorporates a staggered model, for promoting convergence of similar answers to a single choice. HiveMind implements two mechanisms, working together in a shared memory space: The Confidence mechanism selects the best workers for a task, and the Pivot mechanism motivates them to optimize their answers with respect to accuracy and speed. We assume workers are risk-neutral, rational, have quasi-linear utility functions and have no knowledge of the specific parameters of the task *a priori*.

Staggered Model

[1] Note that the model implicitly assumes the existence of a trustworthy online escrow mechanism. Work on this problem has been done by Witkowski, Seuken and Parkes (Witkowski, Seuken & Parkes 2011). Other work also examines the issues affecting trust-inducing mechanisms in an online setting (Witkowski 2011).

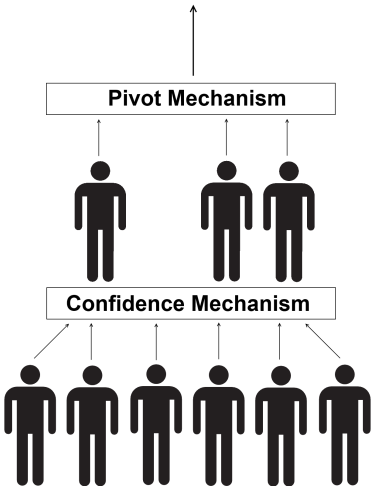

Figure 1. Design of the HiveMind model. The first phase of the default mechanism (Confidence mechanism [CM]) selects appropriate workers, while the second (Pivot mechanism [PM]) aggregates input into a single final answer.

HiveMind uses a staggered or step-wise model for aggregating answers. Every worker entering the task (except those in the initializing set) receive a set of response-choices, called a *step*, which they can agree with or propose a new response to. Votes over these choices invariably weed out responses yielding subsequent steps that culminate in a final collective answer. For purposes of illustration, the voting function is taken to be plurality. An initializing set of workers seed the first step with responses. Each set of choices incorporates two sets: The $main$ set consists of answer proposals which have the greatest number of votes in agreement, and the suggestion set consists of the most recently proposed answers. Choices in the suggestion set get promoted to the main set once they acquire enough votes. Each choice in both sets is timestamped, which is used as a tiebreaker.

The size of the main set (hereafter referred to as N_m) is determined at implementation. Calculation of an optimal N_m depends on a number of factors: too few choices increases the probability of a worker receiving a positive reward by agreeing with an arbitrary choice, but too many choices may hamper reaching a consensus.

The suggestion set acts as a controlled group communication channel. Each time an answer is submitted, it is displayed for at least some period of time in the suggestion set. This ensures workers view all submitted answers and avoids the case of a new answer never being seen because all elements in the main set have more than one vote. The size of the suggestion set can vary, or even include all suggestions that have been made. However, new proposals must always be accommodated for the Pivot mechanism to work. We refer to the size of the suggestion set as N_s. Note that the probability of agents answering at exactly the same time increases with the density of agents per step, which affects the sizes of the main and suggestion sets.

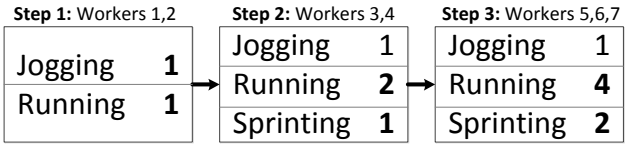

Figure 2. The staggered model used in an activity recognition domain. In the first part, workers w_1 and w_2 create answers. Next, w_3 votes for 'Running', while w_4 creates another new answer. Finally, w_5 and w_7 vote for 'Running', w_6 votes for 'Sprinting'. At the end of the round, 'Running', created by w_2, wins.

Worker Confidence Mechanism

The function of the Confidence mechanism for a given task is to select workers inclined to put in sufficient effort. The problem with implementing such a metric is enforcing commitments. For example, we may ask workers for the number of minutes they agree to put into a task and select those giving the highest numbers; however, the commitment is not enforceable and does not take into account efficiency. This leads to two of our central intuitions: (i) the only person who truly knows the amount of effort going into a task for any worker is the worker himself - thus any commitments must be self-enforced, and (ii) it follows that the best person to optimize the efficiency of any worker is the worker himself - the main tenet of the Pivot mechanism.

The Confidence mechanism captures the first intuition by requiring workers to stake an amount on the outcome of the task. A worker i is required to bid an amount b_i in a range of values specified at implementation. The mechanism parses the bids into three classes: Class A is the highest set of bids (defined here as the top quartile of bids for purpose of illustration). Class B corresponds to workers in the next-highest of bidding range. The last class consists of all others.

Classes A and B are characterized by two values that specify their range. The lower threshold b for any class X is defined such that $\forall b_i \in X, b_t \leq b_i$, also defined as the lowest value in a set which is also the stake for selected workers in a class. The upper threshold, or b_{max} for class A is the range maximum, and for class B, $b_{max(B)} < b_{t_A}$. The Confidence mechanism follows a second-price sealed-bid auction model where no worker can predict, or have any significant control or significant predictive ability over their stake or reward, save the knowledge that their stake is at most their bid. Thus workers bid the highest amount they are willing to stake.

Confidence Mechanism: Worker Strategy

Every worker has knowledge of the bid range (b_{max} for class A) and their own bid. The exact parameters of any class and the bids of other workers are both hidden from workers, but they are aware that the highest bidding workers are selected.

Three factors lead to our optimal result in selecting workers: (i) the presence of a stake, (ii) the lack of significant control (and thus absence of prediction) over the stake or reward, and (iii) the two-class selection. Workers, if selected, have a staked amount, but in the absence of a 2-class model, the Confidence mechanism selects only those workers who wish to be selected, but are willing (rather, have no choice but) to

stake the maximum. Therefore, these workers will be motivated to bid the range maximum as well. The addition of a second class adds nuance to the rationale behind individual bids: workers who wish to be selected, but who do not wish to bid the range maximum, will bid the highest amount they are willing to stake, since the stake $b \leq b_i$.

Workers cannot make predictions of their exact stake or reward at the time of the bid: this requires incorporating workers' beliefs about their expected payoffs at the time of the bid (at which time they have no knowledge of the specific task, and thus confidence in their abilities to handle it[2]. However, as the selection metric is curved, their chance of being selected over the bid range is an increasing function. This may cause workers to slightly increase or shade their bids upwards though this does not hurt our selection model. The Confidence mechanism thus motivates workers to bid the maximum amount they are willing to stake.

Selection of two classes (with payoffs specific to each class) also allows nuance when dealing with an additional crowdsourcing parameter not incorporated currently in HiveMind - a task budget. While our model does not deal with budget constraints, the related ramifications in a two-class model of selected workers are worthy of exploration.

PIVOT MECHANISM

The Pivot mechanism is an optimizing aggregation mechanism, used to encourage the optimization of both speed and accuracy, that extends the Confidence mechanism. In the HiveMind model, workers can respond to their given task either by agreeing with (voting for) a step-choice or proposing a new choice. Rather than choosing the action that best reflects the most correct answer, workers can play suboptimal actions due to: (i) *noise* introduced when workers indiscriminately propose answers to cause vote splitting and discourage convergence, and (ii) the majority effect, in which workers are discouraged from proposing legitimate responses since they perceive existing responses to have a head start in votes. The Pivot mechanism first defines a disproportionately larger reward for playing propose than playing agree. It attempts to solve these issues by parameterizing threshold values for the feasibility of playing agree or playing propose over staying out, and playing either propose or agree over the other.

Worker actions: The set of actions for a worker i has two dimensions: With respect to choices: Workers can play A , can play P, or can choose to stay out.

Time: Workers can answer immediately (at the worker's entry time, t_0) or wait. In short, they can answer at any time t such that $t_0 < t < task_{end}$ (the ending time). The earliest time a worker can fully evaluate their choices and answer accurately/correctly is designated as t_1.

Confidence: When deciding on their course of action, a worker evaluates their abilities specific to the task as well as external factors such as the payoff functions. The confidence

[2]This holds by assumption, but may not be true in specific implementations. For example, workers may have some task or domain knowledge from prior experience with similar tasks.

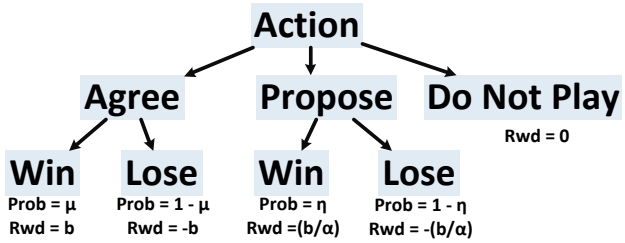

Figure 3. HiveMind action tree for each worker.

of a worker to play an action and win is defined as two variables: (i) $\mu = [0, 1]$ is the confidence level any individual worker has they will play agree and win, and (ii) $\eta = [0, 1]$ is the confidence level any worker has of playing propose and winning. While quantifying confidence for workers is infeasible in general, it is possible for a very narrow case: a worker can quantify only their own confidence level for a specific task. Our approach is sound because our confidence variable only shows motivation for workers' own actions.

Payoff Functions: By the selection mechanism, each selected worker stakes amount b; we use a variable $\alpha \in [0, 1]$ to make the reward for playing P relatively greater than the reward for playing A. For each worker over their actions is the following:

$\text{payoff}_A = \mu\alpha b + (1 - \mu)b$
$\text{payoff}_P = \eta\frac{b}{\alpha} + (1 - \eta)b$
$\text{payoff}_{none} = 0$

Case 1: A is feasible iff $\text{payoff}_A \geq 0$
$\implies \mu b + (1 - \mu)(-b) \geq 0$
$\implies \mu \geq \frac{1}{2}$

Case 2: Play propose is feasible iff $\text{payoff}_P \geq 0$
$\implies \eta\frac{b}{\alpha} + (1 - \eta)(-b) \geq 0$
$\implies \eta \geq \frac{\alpha}{1+\alpha}$

Case 3: A player will (weakly) prefer playing propose over agree iff $\text{payoff}_P \geq \text{payoff}_A$
$\implies \eta\frac{b}{\alpha} + (1 - \eta)(-b) \geq \mu b + (1 - \mu)(-b)$
$\implies \eta \geq 2\mu\alpha$

The variable α sets the threshold gap between μ and η for playing propose over agree, as well as the threshold value for η for playing propose to be feasible. The example below shows, for various values of α, corresponding threshold gap values between μ and η and threshold values for η:

$$\alpha = \begin{cases} 0 & \eta \geq 0; \eta \geq 0 \\ \frac{1}{4} & \eta \geq \frac{1}{2}\mu; \eta \geq \frac{1}{5} \\ \frac{1}{2} & \eta \geq \mu; \eta \geq \frac{1}{3} \\ \frac{3}{4} & \eta \geq \frac{2}{3}\mu; \eta \geq \frac{3}{7} \\ 1 & \eta \geq 2\mu; \eta \geq \frac{1}{2} \end{cases}$$

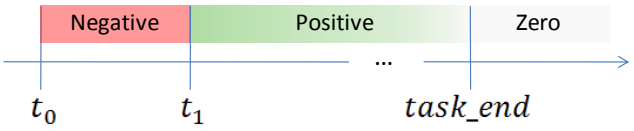

Figure 4. Expected payoffs over time. From time t to t_1 the agent has negative expected payoff, the optimal payoff comes at time t_1 then declines over time until the end of the task, at which point it is 0.

Therefore, α can adjust for *confidence inflation* of one confidence variable with respect to the other, or, in particular, the lead μ may have over η due to the majority effect[3]. Noise can be eliminated as well for higher values of α that increase the threshold gap between η and μ, thus playing P is feasible only for high values of η (Note the gap values subsume the threshold values for η for all cases of $\mu \geq \frac{1}{2}$).

Equilibria: We have multiple symmetric Nash equilibria such as the following for a crowd of n workers, $\alpha = \frac{1}{2}$ and a step-size of 2:

$$\alpha = \frac{1}{2} \begin{cases} \eta \geq \mu & \textit{play P} \\ \eta \leq \mu \; and \; \frac{1}{2} \leq \mu & \textit{play A} \\ else & none \end{cases}$$

$\sigma(1) = P$
$\sigma(2) = P$
$\vdots$
$\sigma(n) = A$

Worker Strategy Across Time

Consider the actions of any worker with respect to time: The worker can play at any time t such that $t_0 \leq t \leq task_{end}$. We take three possible cases with respect to t_1:

Case 1: At any time $t < t_1$ the worker receives a negative payoff (refer to section Freeloader's Payoff), thus playing at $t < t_1$ is a dominated strategy and so is never played.

Case 2: playing at $t = t_1 < task_{end}$, the worker has the following payoffs:

$\text{payoff}_{A_{t_1}} = \mu_a \alpha b_m + (1 - \mu_a) b_t$ where $\alpha \in [0, 1]$

$\text{payoff}_{P_{t_1}} = \mu_p b_k + (1 - \mu_a) b_t$

For the purpose of illustration we assume time to be in discrete units t_1,t_2,$task_{end}$.

Case 3: We use a value $\delta \in (0, 1)$ to discount future payoffs for times $t_n > t_1$ as follows:

$\text{payoff}_{A_{t_n}} = \delta^{n-1} \; \text{payoff}_{A_{t_1}}$

$\text{payoff}_{P_{t_n}} = \delta^{n-1} \; \text{payoff}_{P_{t_1}}$

As $n \to \infty, \delta \to 0$

Other cases: For any $t \geq task_{end}$ the worker cannot play, and therefore their payoff is always 0.

[3]Any step-choice comes with at least one vote and hence any player's confidence levels over these answer choices is inflated.

The discount factor δ captures the notion of the worker's uncertainty of when the task ends. Thus, for any value, the worker maximizes their payoff at $t = t_1$. Therefore playing at t_1 is a weakly dominant over any other strategy.

Freeloader's Payoff

For our model to be robust, we cannot allow workers to gain rewards without any effort. That is, workers cannot *freeload*, or select answers arbitrarily as a feasible strategy. We define freeloaders as workers who select an answer (to agree with) arbitrarily from the list of available choices at any given step. In agreeing with an answer, a freeloader can agree arbitrarily with any of (total number of choices at a step $c = num_{mainset} + num_{suggset}$) answers and can answer at any time between t_0 and $task_{end}$. However, calculating a worker's expected payoff for freeloading at any step in the task would depend on knowing not only the number of proposed answers in the entire task, but their distribution across all future steps, which are both unlikely to be known. Thus, not only is this information not available to the worker, but it is not available to the mechanism. The best case for the freeloader occurs when they are fortunate enough to vote on an answer in a step where the winning answer is included as a choice. We again define b as the amount staked by the worker, c as the number of choices and r as the maximum reward value for agreeing with the correct answer: $r = b\dot{x}$ where $x > 0$. We then select c and r such that the freeloader's expected payoff is $r/c - |b(c-1)/c| < 0$ making freeloading a dominated strategy for any worker. Adjusting values of c and r in the mechanism implementation follows intuition as well: If c is usually low coupled with r being high relative to the worker's bet b, then the expected payoff may well be positive. That is, we want to ensure $0 < x < c - 1$. In other words, if the number of choices in a given step is low, the probability of choosing a winning answer increases and, if the reward is decidedly large compared to the amount the worker bids, his or her expected payoff may be positive.

Expert Mechanism

The Pivot mechanism works for convergence-centric tasks, but cannot be applied for tasks where workers are expected to go against the crowd. The confidence levels μ and η incorporate both the belief that a choice (or proposal) is correct as well as that the crowd will support it, with the latter factor outweighing the former in importance. That is, $\eta = f(\eta_{int}, \eta_{ext})$ where for each worker, confidence levels over answers attributed to internal metrics and external beliefs (about the crowd) are presented by η_{int} and η_{ext} respectively. Similarly, $\mu = f(\mu_{int}, \mu_{ext})$. For non-convergence-centric crowdsourcing tasks where the worker is to propose his or her best response irrespective of his or her beliefs about the crowd, we can employ a Kindred Expert mechanism: Consider such a task where the worker currently plays agree (or has high μ) in two cases: (1) When the worker believes an answer choice is correct and will probably win and (2) when the worker believes an answer choice is incorrect but will probably win. Consider having an expert check the collective answer as well as all individual answers proposed: the expert is called in for any given task with a probability p (say,

$p = \frac{1}{2}$ for purposes of illustration). The expert can choose to reject the collective answer on evaluation and reward another individual proposed answer. If the worker believes the expert will be consulted for the current task, he believes his proposed answer will be recognized as correct by the expert (even if the crowd chooses not to do so), and so plays propose. Thus we counteract the belief individual workers have about their crowd with beliefs about another crowd (the expert or panel of experts). However, this could go horribly wrong: It may be that in addition to negating case(2), beliefs about the expert could negate case(1) (i.e., the worker could believe the crowd is correct, but may believe the expert will not recognize it). Thus, for the expert solution to work, a worker's belief about the expert must be consistent with the worker's notion of a better answer (this is our assumption). That is, we provide no other information on the expert (no definition) except to state that he or she is an expert. The definition of the expert, then, is left to the worker, which we assume will be consistent with the worker's notion of a better answer. Thus a phantom expert must be used so no further beliefs can be formed about them.

Using such deception mechanisms in proposal-elicitation tasks is not sustainable. Given enough time, even significant incomplete information, deception mechanisms will eventually cease to work thus requiring fresh solutions to proposal-elicitation tasks in a CI model.

PROPOSAL-ELICITATION MECHANISM

Extracting collective answers from the crowd provides a reliable means of finding correct answer given that a majority of workers can generate or identify the correct answer. However, for many domains, knowledgeable crowds are hard to find. In these cases, we want to be able to leverage a smaller set of workers who may still know the correct answer. Eliciting, then recognizing, such answers is difficult in CI systems, since workers are generally encouraged to agree with the majority in order to be rewarded as is the case in the Pivot mechanism. We present a solution to this elicitation problem: Our solution enables workers to fight the majority effect and give his or her best response - all within the framework of a democratic aggregate function. In task-models with variable entries and exists for workers, our approach attempts to correct the initial advantage workers may obtain in proposing at the early stages of the task, which acts as a deterrent for workers entering in the later stages to propose.

We follow the staggered model of Pivot where workers are presented with a step (set of response-choices), and can either agree with (vote for) a choice, or add to the choice-set by proposing a new answer.

Structure: Workers in this model are divided into three sets - The *head set*, an initializing set, where workers may only propose new answers, the *main set* where the responses-choices evolve by workers agreeing with or proposing choices, and the *tail set*, where workers provide only votes over the final set of choices. The actions of playing agree and playing propose are denoted by A and P respectively.

In the head set, workers can only propose responses. These responses are used to seed the first step of choices and are given a fixed amount of time and a fixed reward for any non-null response.In the main set, the actions available to workers are $\{$A, P, none$\}$. The choice-set of responses evolve by actions of workers here, and the expert(s) are assumed to be in this set, thus the main set is chosen to be large relative to the size of the headset and tail set, so the probability of an expert existing in this set is high. Unless specified otherwise, references to workers refer to those in the main set.

All proposals with their scores are recorded, and every worker is presented with a step that is, a snapshot in time of the top k choices, initialized for every worker entering the task. Every worker sees only one such step and can submit only one response. The worker can choose to act any time until the end of the task. A payoff function, similar to one used in Pivot, is used where winning when playing P yields a disproportionately large reward relative to winning when playing A.

Our model does not require a specific voting rule for assignation of points, but we use plurality for purposes of illustration. For every worker, agreement with or proposal of any answer in the step assigns 1 additional point to it, with an implicit assignment of 0 additional points for all other step-choices. The score for a proposal submitted by any worker i is: $score_i = \frac{points\ received}{points\ possible}$. All initial proposals are added to the current step, but not scored until a predetermined number of workers vote over it. We refer to this number as the block size, or bl. This solves the problem of all proposals initially having a score of $\frac{1}{1}$. Depending on the number of votes received by each initial proposal during their first bl votes, they are either voted out or voted in. The final set of choices at the end of main set is presented to the tail set, whose function is to cast a final set of votes over the available proposals. For proposals cast with $< bl$ workers remaining in main set, all such proposals are added to the final step of choices with a score equal to the smallest score in the final step. The tail set consists of bl workers, each having a vote equal to $\frac{g}{bl}$, where $g = max_{final} - min_{final}$. Thus it is possible for every choice in the final step to win.

Analysis: Our scoring function utilizes relative vote points instead of a traditional absolute vote count. Thus a choice received 20 out of 30 possible votes scores higher than one receiving 50 out of a 100 possible votes (for $bl \leq$30) even though the latter proposal has more absolute votes. This also negates any advantage earlier entrants have over later entrants, which is especially relevant in task-models with variable entries and exits. Additionally, the notion of a block size, bl, solves the problem of an initial score of $\frac{1}{1}$ propelling the proposal to the top when tallying the scores of all choices - with a block size, a score for any proposal is not calculated until $points\ possible = bl$. This does yield an additional fairness problem that may affect workers' motivation: very late entrants may face an audience numbering less than the block size, and thus have no hope of ever being selected. The existence of the tail set solves this: It allows every worker's proposal, entering at any time, the possibility of winning. Within this model, we can make the case that the expert worker will act as follows: He or she will propose if (i) their prior η_{int} and η_{ext} are both high, and (ii) their η_{int} is high but their

prior η_{ext} is low. In other words, any worker's confidence level in their proposals no longer depends on the absolute number of workers backing other answers or their proposal. A minimum number of workers voting over their proposal is required, which is guaranteed by the existence of the $tail$ set. Thus, every proposal, even those entering late, is given the opportunity to win.

This functionality is crucial to answer-elicitation - as only proportions of votes cast are used to compute the score (as opposed to the traditional approach of counting votes), you could conceivably have a proposal with a small number of votes score higher than proposals with far greater number of votes, the crux of the worker's motivation in combating the majority effect.

CONTINUOUS REAL-TIME TASKS

Many human computation tasks are not easily divisible *a priori*. Many are better completed as continuous processes, but engaging workers over indefinite periods introduces new challenges. For instance, workers need to be compensated for their work, but doing so fairly requires knowing how much work was actually done. One way is to divide work into subtasks of approximately equal size and compensate accordingly. Subtasks can be labeled by adding an operation to continuous tasks for users to signal the end of a subtask. Naively implemented, this would motivate workers to create short subtasks that can be done frequently.

Using a reputation mechanism can prevent workers from shortening the task, even when they are paid per-response. We first give a reputation value to each worker, and set a cut-off, below which workers may be removed from the task. As mentioned earlier, the value of a task increases over time as our confidence in a worker increases, thus workers are motivated to avoid being removed. We define the threshold σ to be an acceptable distance from the correct answer (a) as a fraction of the total time for the subtask. Reputations are then updated using a weight learning function. Workers are penalized for the difference in their submission time and the average of all other workers by a reward function:

$$\text{reward} = 1 - \begin{cases} 0 \text{ if } |ans - ans_w| < \sigma \cdot ans \\ \frac{||ans - ans_w||}{ans} \quad otherwise \end{cases}$$

Since workers cannot observe other workers and each is motivated to finish as soon as possible, workers will be unable to reliably agree to submit the end of response task until the task is actually finished. Workers will be motivated to submit the end of task response as soon as possible to increase their chance of answering the next task before its end because their completion time t_0 is shifted by any additional time spent on the previous task. Thus, workers are motivated to correctly delimit the subtasks by signaling their true ends, but are not motivated to take longer than needed.

Finding Leaders

Legion shows that using a single worker elected by the crowd, called a *leader*, to directly control a task is an effective paradigm in continuous control domains. We can use HiveMind to identify leaders by first applying the Pivot mechanism to a continuous task, then use the following weight learning formula to rank workers:

$$W_{t+1} = (1 - \alpha) \cdot W_t + \alpha \cdot task_{score}$$

where α determines how much history should be considered. We can then select the leader by picking the highest weight. The final answer selected in each subtask is the one selected or generated by the leader.

FUTURE WORK

In the future, we plan to evaluate HiveMind in a real-world setting where workers may have differing motivations and definitions of rationality. This will require implementations which effectively communicate the parameters of the task, and find effective reward values in a variety of domains (since results will be instance specific). These tests can demonstrate two key factors: i) workers will complete the tasks with higher quality per unit of time spent, and ii) adjusting the value of alpha will be correlated with the size of the resulting answer-set (number of responses).

It is also possible to probabilistically model workers' knowledge of their environment and assumptions over incomplete information. We can then analyze how closely the resulting Bayesian-Nash Equilibrium matches the HiveMind model.

CONCLUSION

We have described HiveMind, a theoretical framework for collective intelligence crowdsourcing tasks that optimizes the tradeoff between speed and accuracy in worker responses, while allowing task developers to easily adjust the extent to which the crowd converges to a set of answers by tuning the value of α. HiveMind incorporates a two-part mechanism which first finds an optimal group of workers, then uses those workers to generate and select an answer. The model motivates individual workers to optimize their personal tradeoff of time and accuracy, thus generating correct answers as quickly as possible. HiveMind presents a game-theoretic approach to designing methods of combining worker input in real-time, which can be useful in a variety of domains where workers were previously difficult to motivate properly, such as continuous real-time tasks.

From the point of view of task developers crowds are intrinsically idiosyncratic, never responding quite the way you require. The primary contribution of HiveMind is the increased flexibility it offers task developers in recognition of this: With the alteration of just one variable, developers can finetune collective answers received, promoting convergence or divergence, balancing noise with worker participation, inducing simple responses or rich, complex answers.

REFERENCES

Alonso, O., Kazai, G. & Mizzaro, S. (2012), *Crowdsourcing for Search Engine Evaluation*, Springer.

Aral, S., Ipeirotis, P. & Taylor, S. (2011), Content and context: Identifying the impact of qualitative information on consumer choice, *in* D. F. Galletta & T.-P. Liang, eds, 'ICIS', Association for Information Systems.

Attenberg, J., Ipeirotis, P. G. & Provost, F. J. (2011), Beat the machine: Challenging workers to find the unknown unknowns, *in* 'Human Computation', Vol. WS-11-11 of *AAAI Workshops*, AAAI.

Bernstein, M. S., Brandt, J., Miller, R. C. & Karger, D. R. (2011), Crowds in Two Seconds: Enabling Realtime Crowd-powered Interfaces, *in* 'Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2011)', Santa Barbara, CA, pp. 33–42.

Bernstein, M. S., Little, G., Miller, R. C., Hartmann, B., Ackerman, M. S., Karger, D. R., Crowell, D. & 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, pp. 313–322.

Bigham, J. P., Jayant, C., Ji, H., Little, G., Miller, A., Miller, R. C., Miller, R., Tatarowicz, A., White, B., W. S. & Yeh, T. (2010), VizWiz: nearly real-time answers to visual questions, *in* 'Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2010)', New York, NY, pp. 333–342.

Chawla, S., Hartline, J. D. & Sivan, B. (2012), Optimal crowdsourcing contests, *in* Y. Rabani, ed., 'SODA', SIAM, pp. 856–868.

Chilton, L. (2009), Seaweed: A web application for designing economic games, Master's thesis, MIT.

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

Karger, D., Oh, S. & Shah, D. (2012), Budget-Optimal Task Allocation for Reliable Crowdsourcing Systems, *in* 'Proceedings of the 26th Conference on Neural Information Processing Systems (NIPS 12)', Lake Tahoe, NV, p. To Appear.

Lasecki, W. S., Miller, C. D., Sadilek, A., AbuMoussa, A. & Bigham, J. (2012*a*), Real-time captioning by groups of non-experts, *in* 'In Submission'. http://hci.cs.rochester.edu/pubs/pdfs/legion-scribe.pdf.

Lasecki, W. S., Murray, K. I., White, S., Miller, R. C. & Bigham, J. P. (2011), Real-time Crowd Control of Existing Interfaces, *in* 'Proceedings of the ACM Symposium on User Interface Software and Technology (UIST 2011)', Santa Barbara, CA, pp. 23–32.

Lasecki, W. S., Song, Y., Kautz, H. & Bigham, J. P. (2012*b*), Real-Time Crowd Labeling for Deployable Activity Recognition, *in* 'Pervasive Computing', Newcastle, UK, p. In Submission.

von Ahn, L. & Dabbish, L. (2004*a*), 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, pp. 319–326.

von Ahn, L. & Dabbish, L. (2004*b*), 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, pp. 319–326.

Witkowski, J. (2011), Trust Mechanisms for Online Systems, *in* 'Proceedings of the 22nd International Joint Conference on Articial Intelligence (IJCAI 11)', Barcelona, Spain, pp. 2866–2867.

Witkowski, J., Seuken, S. & Parkes, D. C. (2011), Incentive-Compatible Escrow Mechanisms, *in* 'Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI 11)', San Francisco, CA, pp. 751–757.