Human Intelligence *Needs* Artificial Intelligence

Daniel S. Weld Mausam Peng Dai
Dept of Computer Science and Engineering
University of Washington
Seattle, WA-98195
{weld,mausam,daipeng}@cs.washington.edu

Abstract

Crowdsourcing platforms, such as Amazon Mechanical Turk, have enabled the construction of scalable applications for tasks ranging from product categorization and photo tagging to audio transcription and translation. These vertical applications are typically realized with complex, self-managing workflows that guarantee quality results. But constructing such workflows is challenging, with a huge number of alternative decisions for the designer to consider.

We argue the thesis that "Artificial intelligence methods can greatly simplify the process of creating and managing complex crowdsourced workflows." We present the design of CLOWDER, which uses machine learning to continually refine models of worker performance and task difficulty. Using these models, CLOWDER uses decision-theoretic optimization to 1) choose between alternative workflows, 2) optimize parameters for a workflow, 3) create personalized interfaces for individual workers, and 4) dynamically control the workflow. Preliminary experience suggests that these optimized workflows are significantly more economical (and return higher quality output) than those generated by humans.

Introduction

Crowd-sourcing marketplaces, such as Amazon Mechanical Turk, have the potential to allow rapid construction of complex applications which mix human computation with AI and other automated techniques. Example applications already span the range from product categorization [2], photo tagging [24], business listing verifications [16] to audio/video transcription [17; 23], proofreading [19] and translation [20].

In order to guarantee quality results from potentially error-prone workers, most applications use complex, self-managing workflows with independent production and review stages. For example, iterative improvement [14] and find-fix-verify workflows [1] are popular patterns. But devising these patterns and adapting them to a new task is both complex and time consuming. Existing development environments, *e.g.* Turkit [14] simplify important issues, such as control flow and debugging, but many challenges remain. For example, in order to craft an effective application, the designer must:

- **Choose between alternative workflows for accomplishing the task.** For example, given the task of transcribing an MP3 file, one could ask a worker to do the transcription, or first use speech recognition and then ask workers to find and fix errors. Depending on the accuracy and costs associated with these primitive steps, one or the other workflow may be preferable.
- **Optimize the parameters for a selected workflow.** Suppose one has selected the workflow which uses a single worker to directly transcribe the file; before one can start execution, one must determine the value of continuous parameters, such as the price, the length of the audio file, *etc.*. If the audio track is cut into snippets which are too long, then transcription speed may fall, since workers often prefer short jobs. But if the audio track is cut into many short files, then accuracy may fall because of lost context for the human workers. A computer can methodically try different parameter values to find the best.
- **Create tuned interfaces for the expected workers.** The precise wording, layout and even color of an interface can dramatically affect the performance of users. One can use Fitt's Law or alternative cost models to automatically design effective interfaces [7]. Comprehensive "A-B" testing of alternative designs, automated by computer, is also essential [12].
- **Control execution of the final workflow.** Some decisions, for example the number of cycles in an iterative improvement workflow and the number of voters used for verification, can not be optimally determined *a priori*. Instead, decision-theoretic methods, which incorporate a model of worker accuracy, can dramatically improve on naive strategies such as majority vote [3].

Our long-term goal is to prove the value of AI methods on these problems and to build intelligent tools that facilitate the rapid construction of effective crowd-sourced workflows. Our first system, TURKONTROL [3; 4], used a partially-observable Markov decision process (POMDP) to perform decision-theoretic optimization of iterative, crowd-sourced workflows. This paper presents the design of our second system, CLOWDER[1], which we are just starting to implement. We start by summarizing the high-level architecture of CLOWDER. Subsequent sections detail the AI rea-

[1] It is said that nothing is as difficult as herding cats, but maybe decision theory is up to the task? A *clowder* is a group of cats.

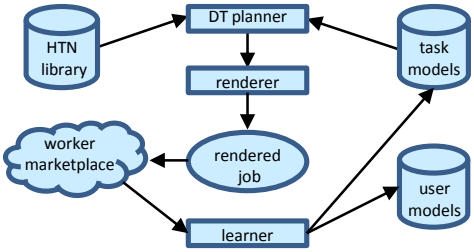

Figure 1: Architecture of the CLOWDER system.

soning used in its major components. We end with a discussion of related work and conclusions.

Overview of CLOWDER

Figure 1 depicts the proposed architecture of CLOWDER. At its core, CLOWDER has the capability to generate, select from, optimize, and control a variety of workflows and also automatically render the best interfaces for a task. It achieves this by accessing a library of pre-defined workflow patterns expressed in a hierarchical task network (HTN)-like representation [18]. Along with each HTN it maintains the relevant planning and learning routines. The learning routine learns task and user models. These parameters aid in controlling the workflow dynamically using a decision-theoretic planner. Finally, it optimizes the task interfaces based on user performance. Overall, it aims to achieve a higher quality-cost-completion time trade-off by optimizing each step of the process. CLOWDER proposes the following novel features:

- **A declarative language to specify workflows.** CLOWDER's language is inspired by the HTN representation. An HTN is a hierarchy of tasks, in which each parent task is broken into multiple children tasks. At the lowest level are the primitive actions that can be directly executed (in our case, jobs that are solved either by machines or are crowd-sourced). Thus, HTN provides a systematic way to explore the possible ways to solve the larger problem. A workflow can be quite naturally represented in an HTN-like representation.
- **Shared models for common task types.** Most crowdsourcing jobs can be captured with a relatively small number of job classes, such as jobs with discrete alternatives, creating/improving content, *etc.* By having a library of job types CLOWDER will be able to share parameters across similar job types. Given a new task, CLOWDER can transfer the knowledge from similar prior tasks, speeding up the learning process. *E.g.*, it could use audio transcription parameters to seed those for the handwriting recognition task, as they are quite similar.
- **Integrated modeling of workers.** CLOWDER models and continually updates its worker's quality parameters. This is especially necessary, since workers often perform poor quality work, so tracking their work and rewarding the good workers is imperative to a healthy functioning platform. While a worker's quality could change based on the task (people not good at writing English descriptions could still be potent audio transcribers), we can seed their task-specific quality parameters based on their average parameters from similar prior tasks.
- **Comprehensive Decision-Theoretic Control.** A workflow has several choices to make including pricing, bonus, number of iterations or voters, and interface layout. Our previous work, TURKONTROL, optimized a subset of these factors for a specific type of workflow. CLOWDER will extend TURKONTROL by allowing a large number of workflows and optimizing for all of these choices.

We now discuss each of these components in detail.

Modeling Worker Performance

Poor quality workers present a major challenge for crowdsourced applications. Although early studies concluded that the majority of workers on Mechanical Turk are diligent [22], more recent investigations suggest a plethora of spam workers. Moreover, the error rates are quite high for open-ended tasks like improving an artifact or fixing grammatical errors [1].

Ipeirotis [9] has suggested several important improvements to the Mechanical Turk marketplace platform, one of which is a *better reputation system* for evaluating workers. He argues that payment should be separated from evaluation, employers should be allowed to rate workers, and the platform should provide more visibility into a worker's history. Worker quality should be reported as a function of job type in addition to aggregate measures. By surfacing limited information, such as percentage acceptance and number of completed hits, Mechanical Turk makes it easy for spam workers to pose as responsible by rank boosting [8; 6]. Yet even if Mechanical Turk is slow to improve its platform, alternative marketplaces, such as eLance, guru, oDesk, and vWorker, are doing so.

But even if Ipeirotis' improved reputation system is widely adopted, the best requesters will still overlay their own models and perform proprietary reasoning about worker quality. In a crowd-sourced environment, the specific workflow employed (along with algorithms to control it) is likely to represent a large part of a requester's competitive advantage. The more an employer knows about the detailed strengths and weaknesses of a worker, the better the employer can apply the worker to appropriate jobs within that workflow. Thus, knowledge about a worker provides a proprietary advantage to an employer and is unlikely to be fully shared. Just as today's physically-based organizations spend considerable resources on monitoring employee performance, we expect crowd-sourced worker modeling to be an area of ongoing innovation. TURKONTROL devised a novel approach to worker modeling, which CLOWDER extends.

Learning a Model of Simple Tasks: Let us focus on the simplest tasks first – predicting the worker behavior when answering a binary question. The learning problem is to estimate the probability of a worker x answering a binary ballot question correctly. While prior work has assumed all

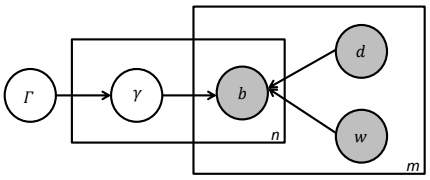

Figure 2: A plate model of ballot jobs; b represents the ballot outcome; γ, a worker's individual error parameter; d, the difficulty of the job and w, truth value of the job. Γ is the prior on workers' errors. Shaded nodes represent observed variables.

workers to be independent, we realize that worker inputs are actually correlated – their errors often depend on the intrinsic difficulty of the question (d). We assume conditional independence of workers given the difficulty. We model a random worker's accuracy by a parametrized distribution: $a_x(d) = \frac{1}{2}[1-(1-d)^{\gamma_x}]$. Lower γ represents a better performing worker. We seek to continually estimate γ values for a worker working on our jobs.

Figure 2 presents our generative model of such jobs in plate notation; shaded variables are observed. Here b represents the ballot by one worker, and v represents the true value of the question.

We seek to learn $\vec{\gamma}$. Moreover, we use the mean $\bar{\gamma}$ as an estimate for future, unseen workers. To generate training data for our task we select m questions and post n copies of ballot jobs. We use $b_{i,x}$ to denote x^{th} worker's ballot on the i^{th} question. Let $w_i = true(false)$ be the ground truth for i^{th} question and let d_i denote the difficulty of answering this question. We take human expert labels for true answer and difficulty of a question. Assuming uniform prior over γ, we can estimate γ_x parameters using maximum likelihood.

Alternatively, we could also use a pure unsupervised approach (akin to [27]) using the EM algorithm to jointly estimate the true labels, difficulties and γ parameters of a worker. Supervised learning leads to better learning, so that will be the algorithm of choice, in case getting some labeled data isn't too costly.

For a new worker, we initially use the mean $\bar{\gamma}$, and as we obtain more information about them continually update their parameter using a simple update rule. At the question answering time, we can use the existing ballots $\vec{\vec{b}}$, uniform prior on Γ, and our conditional independence assumption to estimate the true values $\vec{w}$.

$$\begin{aligned} P(\vec{\gamma}|\vec{\vec{b}},\vec{w},\vec{d}) &\propto P(\vec{\vec{b}}|\vec{\gamma},\vec{w},\vec{d}) \\ &= \Pi_{i=1}^{m}\Pi_{x=1}^{n} P(b_{i,x}|\gamma_x,d_i,w_i). \end{aligned}$$

Learning Complex Worker Models: In addition to simple binary questions, TURKONTROL also studies a specific case of learning more complex worker models that arise in the iterative improvement workflow – learning a worker's improvement model. Here, we wish to estimate the probability distribution of the quality of a new artifact (q') when a worker x tries to improve an artifact of quality q. Learning such two dimensional distributions is a challenging problem. TURKONTROL assumes specific distribution shapes and applies parameter fitting techniques to make the problem tractable [4].

Similar ideas apply to learning other more complex models. In CLOWDER we propose to enable learning capability for a wide variety of worker models. We list a few below based on the job type. We anticipate that existing models from other tasks will aid in seeding the worker models for a new task, which can be continually updated as we gain more information about a worker on the task at hand.

- *Discrete alternatives.* Workers may be asked to choose between more than 2 discrete alternatives. A simple extension of our ballot model suffices for this.
- *Find jobs.* The Soylent word processor popularized a crowd-sourcing design pattern called *Find-Fix-Verify* which "splits complex crowd intelligence tasks into a series of generation and review stages that utilize independent agreement and voting to produce reliable results." [1]. Find jobs typically present a worker with a sequence of data, *e.g.*, a textual passage, and ask the worker to identify *flaw* locations in that sequence, *e.g.* the location of grammatical errors. Since only a few locations can be returned, we can learn models of these jobs with an extension of the discrete alternatives framework.
- *Improvement jobs.* This class of job (also known as a "Fix" job) requires the worker to revise some piece of content, perhaps by fixing a grammatical error or by extending a written description of a picture. We can employ and extend curve fitting ideas in [4] to learn such models.
- *Content creation jobs.* This class of job would be used when initializing an iterative improvement workflow or in the first step of a transcription task. It can be modeled as a degenerate case of an improvement job.

Optimizing & Controlling Workflows

Every given workflow has one or more parameters whose value needs to be set before execution. The most obvious examples are the price offered to worker for completion of the job and HTML interface provided to workers, but other workflows have additional parameters. For example, the length of individual audio files in transcription or the number of distinct examples in a labeling task.

There are two basic methods for optimizing the parameters: blind and model-based search. If nothing is known about the effect of the parameter, then blind search — enumerating different possible values and measuring the effect on worker performance — will be necessary. The designer will likely wish to specify the range of values for the system to consider. If the system has been given or has learned a model of the parameter's effect, it can use direct optimization or branch and bound to find good parameter values before issuing any jobs to actual workers.

Execution Control: Probably the biggest benefit of decision-theoretic modeling of a workflow is the ability to automatically control the different pieces of the task and dynamically allocate the resources to the sub-tasks that are expected to yield largest benefits. The benefits are evaluated in terms of the utility that is given as the input by the requester.

CLOWDER will extend the decision-theoretic control methodology used in TURKONTROL [3]. Consider an

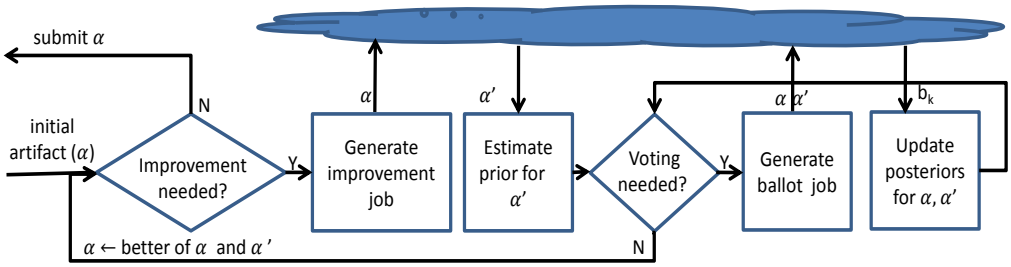

Figure 3: Decision-theoretic Computations needed to control an iterative-improvement workflow.

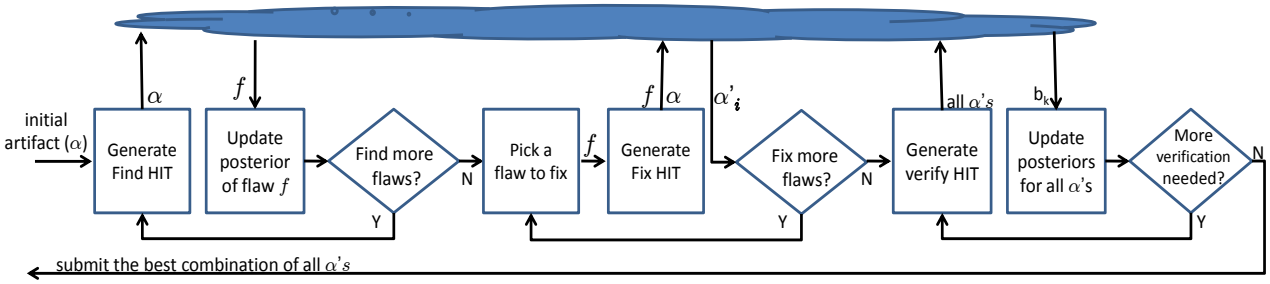

Figure 4: Decision-theoretic Computations needed to control the Soylent word processor.

iterative-improvement workflow in which an artifact (*e.g.*, an English description of an image) created by the first worker goes through several improvement iterations; each iteration comprising an improvement and a ballot phase [14]. An *improvement job* solicits α', an improvement of the current artifact α. In the ballot phase, zero or more workers complete a *ballot job*, voting whether the most recent 'improvement' really is better than the predecessor. The best artifact is promoted to the next iteration.

TURKONTROL controls this workflow based on its *belief* about the quality of the two artifacts as informed by priors and the noisy ballots. Figure 3 shows the relevant decision points for iterative improvement: Which artifact is the best so far? Is it good enough to return or should a worker be tasked to improve it? Should another worker be tasked to compare two artifacts? These decisions are answered using a Partially Observable Markov Decision Process (POMDP) formulation, where the (latent) world state includes the quality of the two artifacts. Initial experiments show that this form of decision-theoretic control can save as much as 28% of the cost to achieve a given quality of artifact compared to previously-proposed policies [4].

Indeed, iterative improvement is not the only workflow that can benefit from decision-theoretic control. In Figure 4 we propose the initial design of a controller for Soylent [1], a word processor that uses a Find-Fix-Verify workflow to shorten and rewrite text written by the user. There are several decision points, such as whether to request more flaws, fixes or votes; and also which flaws to ask the fixes for, and how to combine the various artifacts to submit the final version. CLOWDER will implement general purpose routines to control several workflows that can be expressed in its general representation language (see next section).

Because of it's high-dimensional and continuous state space, solving a POMDP is a notoriously hard problem. For the case of iterative-improvement workflows a simple k-step lookahead greedy search performed remarkably well; however, more sophisticated methods may be necessary as we increase the number of decision points made by the agent. We will investigate a variety of strategies, including discretization and the Monte Carlo methods pioneered in UCT [11]). Our prior experience with approximate and optimal, MDP and POMDP algorithms (*e.g.*, [3; 13]) will come in handy in scaling to the larger problems.

Pricing Jobs: There are several ways to compute the best price for a job. Once a concrete interface has been selected, it is easy to measure the time required to complete a job; multiplying by an expected hourly rate produces a price. But money is not a worker's only motivation; the intellectual challenge of a job and even attractiveness of a UI can reduce the necessary wage [25].

Mason and Watts [15] showed that increasing the payment for a task on Mechanical Turk increased the quantity of tasks performed by a worker but not the quality. So, if the task comes with a deadline then the variation of the price with the rate of completion of tasks could determine the pay. Moreover, there are methods for improving worker performance and these may be explored by CLOWDER.

Awarding Bonuses: Paying an optional bonus for higher quality (or more rapidly-completed) work is a tried and tested way to motivate better submissions from the workers. For an automated agent the decision question will be (1) When to pay a bonus? (2) Under what quality conditions should a bonus be paid? and (3) What magnitude bonus should be paid? Intuitively, if we had an expectation on the total cost of a job, and we ended up saving some of that money, a fraction of the agent's savings could be used to reward the workers who did well in this task.

Other Parameters: CLOWDER will provide support to

learn a variety of workflow parameters. First, we will study common workflows to abstract away the typical classes of parameters. For example, a common parameter type is the size of the decomposition. Whether we wish to recognize a long handwriting task, or transcribe a large audio file, or ask people to identify grammatical errors in a long essay, there is an optimum length in which to subdivide the problem. A short length may result in poor quality output because the relevant context will be missing, whereas a long length could also result in poor quality, since the workers may lose focus in the middle. Moreover, workers do not like really long jobs, so the completion rates may fall too. An easy blind way to learn the optimum length is by using binary search. However, if other tasks have solved similar problems in the past then we can adapt their parameters and explore regions close to the existing parameters for a more efficient learning.

We propose to add support for typical parameter types, which can greatly boost a workflow's performance. This will also be a subroutine in selecting one workflow over another, since each workflow needs to be evaluated in their best configuration to compare against another workflow.

Optimizing Interfaces: The layout of an interface and choice of widgets for entering data can make an enormous difference in the productivity of the interface's users. Leading Internet companies perform detailed quantitative analysis of the user-behavior and completion rates of Web-based product-search and checkout workflows [12] — small changes can have dramatic effects on productivity. It stands to reason, therefore, that the UI layout of crowd-sourced jobs is a critical factor in how quickly they can be completed and hence the payment necessary to motivate this completion. Interestingly, the job interfaces for many jobs on Mechanical Turk appears poor — ignoring various guidelines and tips. Furthermore, many workers post suggestions on how to improve workflows to help requesters improve the flow.

CLOWDER will extend our previous work on automatic generation of personalized user interfaces, which uses a functional representation for interfaces and the SUPPLE algorithm for decision-theoretic interface personalization [7]. We developed learning algorithms for inducing the user-preference models necessary to drive the optimization process, but one could also use Fitt's law for the common case of able-bodied workers. Finally, we demonstrated that our automatic personalized interfaces significantly decreased the time required to perform a wide class of tasks [7]. These methods are perfectly suited to the task of optimizing Web-based workflows. After using SUPPLE to shortlist a few interfaces with good expected performance, CLOWDER will generate trial runs of the tasks using each interface thus automating traditional "A-B" testing to pick the interface which performs best in practice [12].

Generating Complex Workflows

Different tasks require very different kinds of workflows to achieve quality output. Often, these are carefully engineered after significant human effort and extensive testing. We propose to automate part of the process with CLOWDER performing the testing and workflow selection.

For instance, in the audio transcription task, one could just use the output from the speech recognition system. Or one could improve the machine output in a Find-Fix-Verify like workflow. Or, one may decide to completely do away with machine recognition and use workers to directly transcribe the audio. In this situation, one worker could transcribe the whole audio or one may divide the audio into smaller audio files. There could be additional quality control workers that score previous transcription quality. Indeed, CastingWords employs a proprietary workflow for audio transcription.

To realize the vision of an intelligent crowd-sourcing agent capable of dealing with a variety of tasks, CLOWDER will facilitate the process of searching through these different workflows and thus assisting the human designer in the selection process. However, to initiate this search, CLOWDER needs a language to represent the potential workflows that solve the task. We anticipate that the language will resemble the declarative hierarchical task network representation [18] from the automated planning literature. This could be augmented with workflow-specific primitives for data input (such as observation actions) using a variant of the SUPPLE representation [7] .

A hierarchical task network, which is commonly used in domain-specific planning literature, is a natural representation for our purposes. An HTN subdivides each task hierarchically into a set of subtasks, and also maintains the preconditions and effects of applying each subtask. Thus, given a workflow represented as an HTN, CLOWDER can easily enumerate all the workflows that solve the given task.

The next challenge is to select the best workflow. A better workflow maximizes the utility obtaining the best quality-cost-completion time tradeoff. However, enumerating each potential workflow and evaluating it individually can be time-consuming as well as financially wasteful. We will develop routines that, given an HTN, will evaluate each workflow-component in their hierarchical order. CLOWDER will compute the expected utility of each subtask, which can then be used to select the best workflow for the whole task.

Automatically selecting workflows is a rather challenging direction for CLOWDER, but one that will go a long way in realizing the potential of crowd-sourcing by making it convenient for requesters to create complex workflows without needing to understand the mathematical calculations necessary to execute them efficiently.

Related Work

Shahaf and Horvitz [21] also use an HTN-style decomposition algorithm to find a coalition of workers, each with different skill sets, to solve a task. Our workflow selection ideas are inspired by their work.

Other researchers have studied modeling worker competencies. Whitehill *et al.* [27] also model problem difficulty, though they use the unsupervised EM algorithm. Wellinder *et al.* [26] add annotation bias and multi-dimensional parameters. Kern *et al.* [10] also make predictions on whether to elicit new answers. Donmez *et al.* study worker accuracies as a function of time [5].

Recently Zhang *et al.* [28] argue that *all* aspects of the workflow right from designing to controlling can be crowd-

sourced. In many ways, our thesis is in direct contrast with theirs. We believe that all tasks are not well-suited for unskilled crowd-sourced workers – often they do not have a global picture, humans are not best at numeric optimizations, in which computers excel. Thus, mixed-initiative systems that collaborate machine intelligence with humans are essential to success of complex, crowd-sourcing tasks.

Conclusions

Amazon Mechanical Turk has the tagline 'Artificial Artificial Intelligence' emphasizing the ability of the crowd in performing several tasks commonly attributed for AI systems. In this paper we argue that AI techniques are rather essential in managing, controlling, executing, and evaluating the tasks performed on crowd-sourcing platforms.

We outline the design of our system, CLOWDER, that (1) uses machine learning to continually refine the models of worker performance and task difficulty, (2) is able to optimize the parameters and interfaces in a workflow to achieve the best quality-cost-completion time trade-off, (3) can dynamically control a workflow to react to and anticipate the effect of better or worse workers finishing a task, and (4) has the ability to select one among the multiple possible workflows for a task based on automatic evaluation of the different optimized workflows.

CLOWDER combines core ideas from different subfields of artificial intelligence, such as decision-theoretic analysis, model-based planning and execution, machine learning and constraint optimization to solve the multitude of subproblems that arise in the design. The implementation of the system is in progress.

We believe that a mixed-initiative system that combines the power of artificial intelligence with that of *artificial* artificial intelligence has the potential to revolutionize the ways of business processes. We already see several innovative crowd-sourcing applications; we can easily anticipate many more, as CLOWDER reduces the requester skills and computational overhead required to field an application.

Acknowledgments

This work was supported by the WRF / TJ Cable Professorship, Office of Naval Research grant N00014-06-1-0147, and National Science Foundation grants IIS 1016713 and IIS 1016465.

References

[1] M. Bernstein, G. Little, R. Miller, B. Hartmann, M. Ackerman, D. Karger, D. Crowell, and K. Panovich. Soylent: A word processor with a crowd inside. In *UIST*, 2010.

[2] *http://crowdflower.com/solutions/prod_cat/index.html*.

[3] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI10*, 2010.

[4] Peng Dai, Mausam, and Daniel S. Weld. Artificial intelligence for artificial, artificial intelligence. In *AAAI*, 2011.

[5] Pinar Donmez, Jaime G. Carbonell, and Jeff Schneider. A probabilistic framework to learn from multiple annotators with time-varying accuracy. In *SDM*, pages 826–837, 2010.

[6] C. Eickhoff and A. de Vries. How crowdsourcable is your task? In *Proceedings of the WSDM Workshop on Crowdsourcing for Search and Datamining*, 2011.

[7] Krzysztof Z. Gajos, Jacob O. Wobbrock, and Daniel S. Weld. Automatically generating personalized user interfaces with SUPPLE. *Artificial Intelligence*, 174:910–950, August 2010.

[8] P. Ipeirotis. Be a top Mechanical Turk worker: You need $5 and 5 minutes. *http://behind-the-enemy-lines.blogspot.com/2010/10/be-top-mechanical-turk-worker-you-need.html*.

[9] P. Ipeirotis. Plea to Amazon: Fix Mechanical Turk! *http://behind-the-enemy-lines.blogspot.com/2010/10/plea-to-amazon-fix-mechanical-turk.html*.

[10] Robert Kern, Hannes Thies, and Gerhard Satzger. Statistical quality control for human-based electronic services. In *In Proc. of ICSOC*, pages 243–257, 2010.

[11] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.

[12] Ron Kohavi, Randal M. Henne, and Dan Sommerfield. Practical guide to controlled experiments on the web: listen to your customers not to the hippo. In *KDD*, 2007.

[13] A. Kolobov, Mausam, and D. Weld. ReTraSE: Integrating paradigms for approximate probabilistic planning. In *Procs. of IJCAI 2009*, 2009.

[14] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: tools for iterative tasks on mechanical turk. In *HCOMP*, 2009.

[15] W. Mason and D. Watts. Financial incentives and the "performance of crowds". In *Human Computation Workshop (HComp2009)*, 2009.

[16] *http://crowdflower.com/solutions/blv/index.html*.

[17] *http://castingwords.com*.

[18] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN planning system. *JAIR*, 20:379–404, 2003.

[19] *http://www.serv.io/edit*.

[20] *http://www.serv.io/translation*.

[21] Dafna Shahaf and Eric Horvitz. Generlized markets for human and machine computation. In *AAAI*, 2010.

[22] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and A. Ng. Cheap and fast — but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP'08*, 2008.

[23] *http://speakertext.com*.

[24] *http://www.tagasauris.com*.

[25] Michael Toomim, Travis Kriplean, Claus Portner, and James A. Landay. Utility of human-computer interactions: Toward a science of preference measurement. In *CHI*, 2011.

[26] Peter Welinder, Steve Branson, Serge Belongie, and Pietro Perona. The multidimensional wisdom of crowds. In *In Proc. of NIPS*, pages 2424–2432, 2010.

[27] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, and Javier Movellan. Whose vote should count more: Optimal integration of labels from laberlers of unknown expertise. In *In Proc. of NIPS*, pages 2035–2043, 2009.

[28] Haoqi Zhang, Eric Horvitz, Rob C. Miller, and David C. Parkes. Corwdsourcing general computation. In *CHI Workshop on Crowdsourcing and Human Computation*, 2011.