Execution Control for Crowd-Sourcing

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

ABSTRACT
Crowdsourcing marketplaces enable a wide range of applications, but constructing any new application is challenging — usually requiring a complex, self-managing workflow in order to guarantee quality results. We report on the CLOWDER project, which uses machine learning to continually refine models of worker performance and task difficulty. We present decision-theoretic optimization techniques that can select the best parameters for a range of workflows. Initial experiments show our optimized workflows are significantly more economical than with manually set parameters.

ACM Classification Keywords: H5.2. Information interfaces and presentation: User Interfaces.

General terms: Algorithms, performance, experimentation.

Author Keywords: Human computation, decision-theory.

INTRODUCTION

Amazon Mechanical Turk and similar crowd-sourcing marketplaces enable applications that seamlessly mix human computation with AI and other automated techniques. Example applications already span the range from product categorization and photo tagging to A/V transcription and interlingual translation. In order to guarantee quality results from variable competency workers, most applications use complex, self-managing workflows with independent production and review stages. *E.g.*, iterative improvement [7] and find-fix-verify workflows [2] 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 [7] simplify important issues, such as control flow and debugging, but many challenges remain. In order to craft an effective application, the designer must:

- Choose between alternative workflows for the same task.
- Optimize the parameters for a selected workflow.
- Create tuned interfaces for the expected workers.
- Control execution of the final workflow.

We argue that AI methods such as machine learning, decision-theory, optimization can solve these problems, facilitating the rapid construction of effective crowd-sourced workflows. Our first system, TURKONTROL [4, 5], uses decision-theoretic control to optimize iterative workflows on Amazon Mechanical Turk. It automatically learns task-

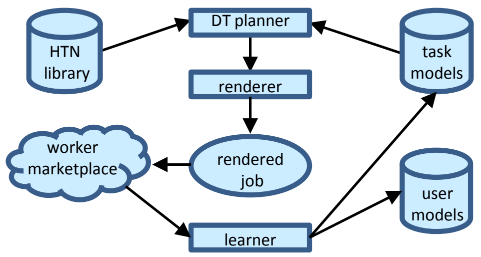

Figure 1: Architecture of the CLOWDER system.

dependent models of typical workers and refines this model for individuals over the course of interaction. More recently, we present the architecture of a successor system, CLOWDER (Figure 1), which we are starting to implement [8]. This poster focuses on our methods for optimizing execution of two new workflow patterns: *find-fix-verify* [2] and the *retainer-bonus* model [1] for real-time crowd creation [3].

OPTIMAL WORKFLOW CONTROL

Decision-theoretic modeling of workflows allows CLOWDER to automatically control the different pieces of the task and dynamically allocate 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. For example, in a find-fix-verify workflow invoked by Soylent, the user's utility function would reward the absence of errors and the quality of the repairing prose. In a retainer-bonus workflow designed for real-time response, the utility function might follow a step function such that answers delivered more than 2-3 seconds after requested had low utility.

CLOWDER extends the decision-theoretic control methodology used in TURKONTROL [4]. Each controller runs a partially-observable Markov decision process (POMDP). The agent seeks to execute actions that maximize the utility based on the current *belief* – a probability distribution over possible world states – since the true world state is hidden.

For example, in an iterative improvement workflow, the (unseen) world state comprises the *quality* of the current artifact (*e.g.*, an English description of a picture), the quality of a modified artifact (*e.g.*, a potentially improved description just returned by a worker) and the accuracy levels of the workers involved. By executing ballot actions (where a potentially-fallible worker reports which artifact is better) the system updates its belief estimates.

For a find-fix-verify workflow, a world state includes the number of flaws and quality estimates of proposed repairs as

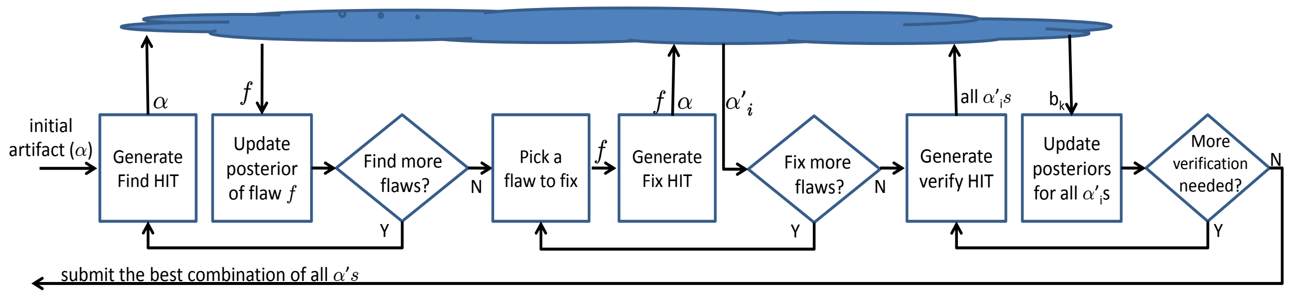

Figure 2: Decision-theoretic computations needed to control the find-fix-verify workflow.

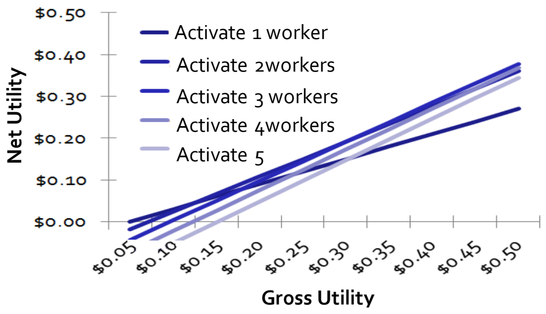

Figure 3: Expected utility of activating retained workers

well as an accuracy model for workers. CLOWDER will control this workflow based on its evolving, probabilistic belief about the state. Figure 2 shows the control flow addressed by a POMDP-based controller for a Soylent-style word processor that uses a Find-Fix-Verify workflow to shorten and rewrite text written by the user [2], 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. By using EM-style learning to track the effectiveness of workers (including both the accuracy of voters and the improvement distribution for those rewriting text), CLOWDER will dynamically calculate how many votes are necessary to verify the edits of different workers. Initial experiments on iterative improvement workflows, which have structure extremely close to fix-verify, 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 hand-coded policies [5].

Because of its 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 [6].

OPTIMIZING REALTIME CROWD RESPONSE

A very different model is required in order to model realtime crowd invocation, such as that popularized in WizViz [3] and Adrenaline [1]. In constrast to our previous models, in which at most one job was sourced to the crowd at any time, the retainer model requires modeling concurrency explicitly. Thus, while our previous POMDP controller chose the next action only upon completion of a HIT, we now need a model which is polled every k time units in order to choose an action.

The world state includes the set of workers currently on retainer with the elapsed time, accuracy and response factors for each. Note that the agent has complete information about the amount of time a retainer worker has been waiting (which affects response time) but only a probability distribution over the worker's accuracy and response characteristics. In addition the agent must model expected arrival time of the next user task (a distribution over delta times until the next task).

The POMDP affords only two actions: recruit another retained worker and submit a user task to a retained worker. We adopt the bonus model described in Bernstein *et al.* [1], since the 3 cent bonus for rapid (2 second) response is cost effective compared to putting additional workers on retainer. But note that the agent may choose not to notify all retained workers when a user task actually comes in. Figure 3 shows the net utility as a function of the user's gross utility for a timely (2 second) response, depending on the number of workers activated and assuming that 60% will respond within 2 seconds. The actual utility calculation must additionally incorporate the individual worker's waiting times, response characteristics, the probability of additional user tasks and the time required to put additional workers on retainer.

REFERENCES

1. M. Bernstein, J. Brandt, R. Miller, and D. Karger. Crowds in two seconds: Enabling realtime crowd-powered interfaces. In *UIST*, 2011.
2. 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.
3. J. Bigham, C. Jayant, H. Ji, G. Little, , A. Miller, R. Miller, A. Tatarowicz, B. White, S. White, and T. Yeh. VizWiz: nearly real-time answers to visual questions. In *UIST*, 2010.
4. P. Dai, Mausam, and D. S. Weld. Decision-theoretic control of crowd-sourced workflows. In *AAAI10*, 2010.
5. P. Dai, Mausam, and D. S. Weld. Artificial intelligence for artificial, artificial intelligence. In *AAAI*, 2011.
6. L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *ECML*, pages 282–293, 2006.
7. G. Little, L. B. Chilton, M. Goldman, and R. C. Miller. TurKit: Tools for Iterative Tasks on Mechanical Turk. In *Human Computation Workshop (HComp2009)*, 2009.
8. D. Weld, Mausam, and P. Dai. Human Intelligence *Needs* Artificial Intelligence. In *Human Computation Workshop (HComp2011)*, 2011.