Decision-Theoretic Control of Crowd-Sourced Workflows

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

Abstract

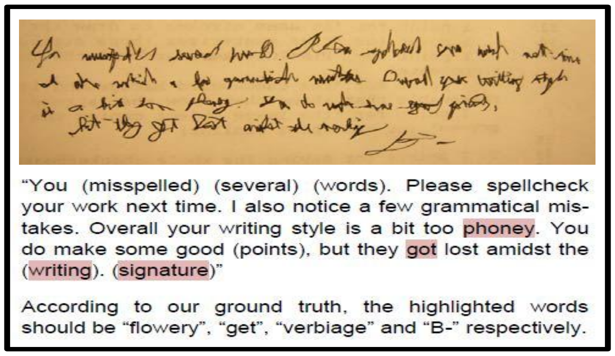
Crowd-sourcing is a recent framework in which human intelligence tasks are outsourced to a crowd of unknown people ("workers") as an open call (e.g., on Amazon's Mechanical Turk). Crowd-sourcing has become immensely popular with hoards of employers ("requesters"), who use it to solve a wide variety of jobs, such as dictation transcription, content screening, etc. To achieve quality results, requesters often subdivide a large task into a chain of bite-sized subtasks that are combined into a complex, iterative workflow in which workers check and improve each other's results. This raises an exciting question for AI — could an autonomous agent control these workflows without human intervention, yielding better results than today's state of the art, a fixed control program?

We plan to study this AI problem, and hope to build an autonomous agent to control crowd-sourcing workflows. This paper shows some initial results by presenting a planner TURKONTROL, that formulates workflow control as a decision-theoretic optimization problem, trading off the implicit quality of a solution artifact against the cost for workers to achieve it. We lay the mathematical framework to govern the various decisions at each point in a popular class of workflows. Based on our analysis we implement the workflow control algorithm and present experiments demonstrating that TURKONTROL obtains much higher utilities than popular fixed policies. We also propose directions to pursue in the future.

Introduction

In today's rapidly accelerating economy an efficient workflow for achieving one's complex business task is often the key to business competitiveness. *Crowd-sourcing*, "the act of taking tasks traditionally performed by an employee or contractor, and outsourcing them to a group (crowd) of people or community in the form of an open call" [8], has the potential to revolutionize information-processing services by quickly coupling human workers with software automation in productive workflows [2].

While the phrase 'crowd-sourcing' was only recently termed, the area has grown rapidly in economic significance with the growth of general-purpose platforms such a Amazon's *Mechanical Turk* [6] and task-specific sites for call centers [5], programming jobs [7] and more. Recent research has shown surprising success in solving difficult tasks


"You (misspelled) (several) (words). Please spellcheck your work next time. I also notice a few grammatical mistakes. Overall your writing style is a bit too phoney. You do make some good (points), but they got lost amidst the (writing). (signature)"

According to our ground truth, the highlighted words should be "flowery", "get", "verbiage" and "B-" respectively.

Figure 1: A handwriting recognition task (almost) successfully solved at Mechanical Turk using an iterative workflow. Workers were shown the text written by a human and in a few iterations they deduced the message (with errors highlighted). Figure adapted from [4].

using the strategy of incremental improvement in an iterative workflow [4]; similar workflows are used commercially to automate dictation transcription and screening of posted content. See Figure 1 for a successful example of a complex task solved using Mechanical Turk — this challenging handwriting was deciphered step by step, with output of one worker feeding as the input to the next. Additional ballot jobs were used to assess whether a worker actually improved the transcription compared to the prior effort.

From an AI perspective, crowdsourced workflows offer a new, exciting and impactful application area for intelligent control. While the handwriting example shows the power of collaborative workflows, we still do not know answers to many questions: (1) what is the optimal number of iterations for such a task? (2) how many ballots should be used for voting? (3) how do these answers change if the workers are skilled (or very error prone)?

Decision Theoretic Optimization

We motivate our work from the iterative workflow example introduced by Little *et al.*. Little's chosen task is iterative text improvement. There is an initial job, which presents the worker with an image and requests an English description of the picture's contents. A subsequent iterative process consists of an *improvement job* and *ballot jobs*. In the improvement job, a (different) worker is shown this same image as well as the current description and is requested to generate an improved English description. Next $n \geq 1$ ballot jobs are posted ("Which text best describes the picture?"). Based on a majority opinion the best description is selected and the loop continues.

The agent's control problem for a workflow like iterative text improvement is defined as follows. As input the agent is given an initial artifact, and the agent is asked to return an artifact which maximizes some payoff based on the quality of the submission.

We measure quality of an artifact in terms of units, which we denote by $q \in [0,1]$. An artifact with quality q means an average dedicated worker has probability $1-q$ of improving the artifact. We assume that requesters will express their utility as a function U from quality to dollars. The quality of an artifact is never exactly known – it is at best estimated based on domain dynamics and observations.

The agent control problem is a POMDP problem [3], as the current state, $(q,\ q')$, of the problem is only partially observable, and can be best approximated by a belief state (Q,Q'). Moreover, since quality is a real number, it is a POMDP in continuous state space. These kind of POMDPs are especially hard to solve for realistic problems. We overcome the computational bottleneck by performing limited lookahead search to make planning more tractable.

Figure 2 summarizes a high level flow for our planner,TURKONTROL [1]. At each step we track our belief in qualities (q and q') of the previous (α) and the current artifact (α'). Each decision or observation gives us new information, reflected in the quality posteriors. These distributions also depend on the accuracy of workers, which we also incrementally estimate based on their previous work.

Quality Tracking Suppose we have an artifact α, with an unknown quality q and a prior[1] density function $f_Q(q)$. Suppose a worker x takes an improvement job and submits another artifact α', with quality q'. We define $f_{Q'|q,x}$ as the conditional quality distribution of q' when worker x improved an artifact of quality q. With a known $f_{Q'|q,x}$ we compute the prior on q' from the law of total probability:

$$f_{Q'}(q') = \int_0^1 f_{Q'|q,x}(q') f_Q(q) dq. \tag{1}$$

While we do have priors on the qualities of both the new and the old artifacts, whether the new artifact is an improvement over the old is not known for sure. Our workflow at this point tries to gather evidence to answer these questions by generating ballot jobs. Say n workers give their votes $\overrightarrow{\mathbf{b}^n} = b_1, \ldots, b_n$, where $b_i \in \{1,0\}$. Based on these votes we compute the posteriors in quality, $f_{Q|\overrightarrow{\mathbf{b}^n}}$ and $f_{Q'|\overrightarrow{\mathbf{b}^n}}$. To accomplish this we make some assumptions. First, we assume each worker x is diligent, so she answers all ballots to the best of her ability. Still she may make mistakes, and we have full knowledge of her accuracy. Second, we assume that several workers will not collaborate adversarially to defeat the system.

We are close to conclude the worker responses $(P(b_i))$ are independent of each other. Notice however, a mistake gives evidence that the question may be intrinsically hard and hence, difficult for others to get it right also. To get around this we introduce *intrinsic difficulty* (d) of our question ($d \in [0,1]$). It depends on whether the two qualities are very close or not. Closer the two artifacts the more difficult it is to judge whether one is better or not:

$$d(q,q') = 1 - |q-q'|^{\mathcal{M}} \tag{2}$$

We can safely assume that given d the probability distributions will be independent of each other.

Moreover, each worker's accuracy will vary with the problem's difficulty. We define $a_x(d)$ as the accuracy of the worker x on a question of difficulty d. We will expect everyone's accuracy to be monotonically decreasing in d. It will approach random behavior as questions get really hard, *i.e.*, $a_x(d) \to 0.5$ as $d \to 1$. Similarly, as $d \to 0$, $a_x(d) \to 1$. We use a group of polynomial functions $\frac{1}{2}[1 + (1-d)^{\gamma_x}]$ for $\gamma_x > 0$ to model $a_x(d)$ under these constraints.

Note that given knowledge of d one can compute the likelihood of a worker answering "Yes". If the i^{th} worker x_i has accuracy $a_{x_i}(d)$, we calculate $P(b_i = 1 \mid q, q')$ as:

$$\begin{aligned} \text{If } q' > q \ P(b_i = 1|q,q') &= a_{x_i}(d(q,q')), \\ \text{If } q' \le q \ P(b_i = 1|q,q') &= 1 - a_{x_i}(d(q,q')). \end{aligned} \tag{3}$$

We first derive the posterior distribution given one more ballot b_{n+1}, $f_{Q|\overrightarrow{\mathbf{b}^n+1}}(q)$ based on existing distributions $f_{Q|\overrightarrow{\mathbf{b}^n}}(q)$ and $f_{Q'|\overrightarrow{\mathbf{b}^n}}(q)$. We abuse notation slightly, using $\overrightarrow{\mathbf{b}^n+1}$ to symbolically denote that n ballots are known and we will receive another ballot (value currently unknown) in the future. By applying the Bayes rule we get

$$\begin{aligned} f_{Q|\overrightarrow{\mathbf{b}^n+1}}(q) &\propto P(b_{n+1} \mid q, \overrightarrow{\mathbf{b}^n}) f_{Q|\overrightarrow{\mathbf{b}^n}}(q) & (4) \\ &= P(b_{n+1} \mid q) f_{Q|\overrightarrow{\mathbf{b}^n}}(q) & (5) \end{aligned}$$

Equation 5 is based on the independence of workers. Now we apply the law of total probability on $P(b_{n+1} \mid q)$:

$$P(b_{n+1} \mid q) = \int_0^1 P(b_{n+1} \mid q, q') f_{Q'|\overrightarrow{\mathbf{b}^n}}(q') dq' \tag{6}$$

The same sequence of steps can be used to compute the posterior of α'. This computation helps us determine the prior quality for the artifact in the the next iteration. It will be either $f_{Q|\overrightarrow{\mathbf{b}}}$ or $f_{Q'|\overrightarrow{\mathbf{b}}}$, depending on whether we decide to keep α or α'.

Utility Estimations We now discuss the computation for the utility of an additional ballot. We use $U_{\overrightarrow{\mathbf{b}^n}}$ to denote the expected utility of stopping now, *i.e.*, without another ballot and $U_{\overrightarrow{\mathbf{b}^n+1}}$ to denote the utility after another ballot. $U_{\overrightarrow{\mathbf{b}^n}}$ can be easily computed as the maximum expected utility from the two artifacts α and α':

$$U_{\overrightarrow{\mathbf{b}^n}} = max\{E[U(Q|\overrightarrow{\mathbf{b}^n})], E[U(Q'|\overrightarrow{\mathbf{b}^n})]\}, \text{ where} \tag{7}$$

$$E[U(Q \mid \overrightarrow{\mathbf{b}^n})] = \int_0^1 \left(\sum_{b_n} U(q) f_{Q|\overrightarrow{\mathbf{b}^n}}(q) P(b_n) \right) dq$$

[1] We will estimate a quality distribution for the very first artifact by a limited training data. Later, posteriors of the previous iteration will become priors of the next.

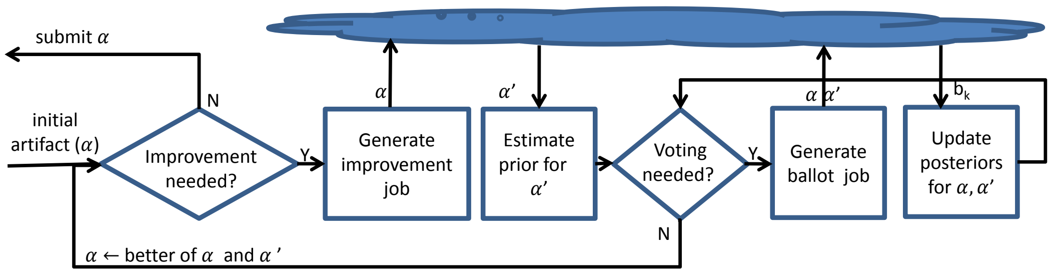

submit α
N
initial artifact (α)
Improvement needed?
Y
α
Generate improvement job
α'
Estimate prior for α'
Voting needed?
Y
N
α α'
Generate ballot job
b_k
Update posteriors for α, α'
$\alpha \leftarrow$ better of α and α'

Figure 2: Computations needed by TURKONTROL for control of an iterative-improvement workflow.

The $n+1^{th}$ ballot, b_{n+1}, could be either "Yes" or "No". The probability distribution $P(b_{n+1} \mid q, q')$ governs this, which also depends on the accuracy of the worker (see Equation 3). Because q and q' are not exactly known, probability of getting the next ballot is computed by applying law of total probability on the joint probability $f_{Q,Q'}(q, q')$

$$P(b_{n+1}) = \int_0^1 \left[\int_0^1 P(b_{n+1}|q, q') f_{Q'|\overrightarrow{\mathbf{b}^n}}(q')dq' \right] f_{Q|\overrightarrow{\mathbf{b}^n}}(q)dq.$$

These allow us to compute $U_{\overrightarrow{\mathbf{b}^n+1}}$ as follows (c_b is the cost of a ballot)

$$U_{\overrightarrow{\mathbf{b}^n+1}} = max\{E[U(Q|\overrightarrow{\mathbf{b}^n+1})], E[U(Q' \mid \overrightarrow{\mathbf{b}^n+1})]\} - c_b$$

Similarly, we can compute the utility of an improvement step. Based on Equation 7 we can choose α or α' to start the improvement with. The belief of the chosen artifact acts as f_Q for Equation 1 and we estimate a new prior $f_{Q'}$ after an improvement step. Expected utility of improvement will be $\max\left(\int_0^1 U(q) f_Q(q) d(q), \int_0^1 U(q') f_{Q'}(q') d(q')\right) - c_{imp}$. Here c_{imp} is the cost an improvement job.

Decision Making At any step we can either choose to do an additional vote, choose the better artifact and attempt another improvement or submit the artifact. We already described computations for utilities for each option. For a greedy 1-step lookahead policy we can simply pick the best of the three options. A greedy policy may be much worse than the optimal. We can compute a better policy by an l-step lookahead algorithm where we evaluate all sequences of l decisions, find the best sequence based on our utilities and then execute the first action of the sequence and repeat.

Experiments

This section aims to empirically answer the following questions: 1) How deep should be an agent's lookahead to best tradeoff between computation time and utility? 2) Does TURKONTROL make better decisions compared to TurKit? 3) Can our planner outperform an agent following a well-informed, fixed policy?

Experimental Setup We set the maximum utility to be 1000 and use a convex utility function $U(q) = 1000\frac{e^q-1}{e-1}$ with $U(0) = 0$ and $U(1) = 1000$. We assume the quality of the initial artifact follows a Beta distribution Beta$(1, 9)$, which implies that the mean quality of the first artifact is 0.1. Suppose the quality of the current artifact is q,

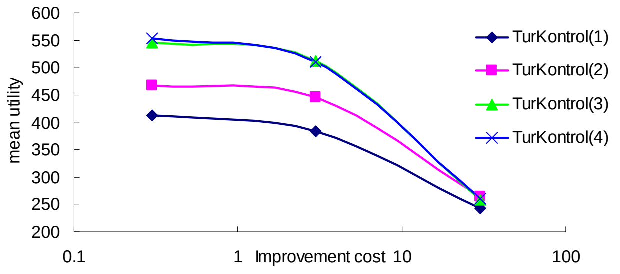


Figure 3: Average net utility of TURKONTROL with various lookahead depths calculated using 10,000 simulation trials on three sets of (improvement, ballot) costs: (30,10), (3,1), and (0.3,0.1). Longer lookahead produces better results, but 2-step lookahead is good enough when costs are relatively high: (30,10).

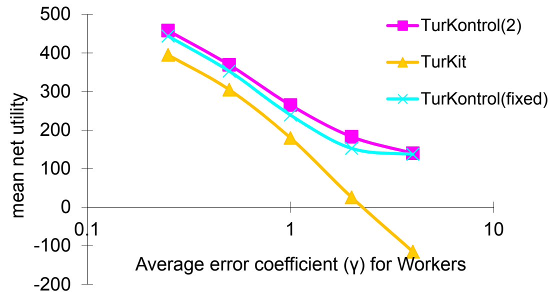


Figure 4: Net utility of three control policies averaged over 10,000 simulation trials, varying mean error coefficient, γ. TurKontrol(2) produces the best policy in every cases.

we assume the conditional distribution $f_{Q'|q,x}$ is Beta distributed, with mean $\mu_{Q'|q,x} = q + 0.5[(1-q) \times (a_x(q) - 0.5) + q \times (a_x(q) - 1)]$. The conditional distribution is Beta$(10\mu_{Q'|q,x}, 10(1-\mu_{Q'|q,x}))$. We fix the ratio of the costs of improvements and ballots, $c_{imp}/c_b = 3$, because ballots take less time. We set the difficulty constant $\mathcal{M} = 0.5$. In each of the simulation runs, we build a pool of 1000 workers, whose error coefficients, γ_x, follow a bell shaped distribution with a fixed mean γ. We also distinguish the accuracies of performing an improvement and answering a ballot by using one half of γ_x when worker x is answering a ballot, since answering a ballot is an easier task, and therefore a worker should have higher accuracy.

Picking the Best Lookahead Depth We first run 10,000 simulation trials with average error coefficient γ=1 on three pairs of improvement and ballot costs — (30,10), (3,1), and (0.3,0.1) — trying to find the best lookahead depth l for TURKONTROL. Figure 3 shows the average *net utility*, the utility of the submitted artifact minus the payment to the workers, of TURKONTROL with different lookahead depths, denoted by TurKontrol(l). Note that there is always a performance gap between TurKontrol(1) and TurKontrol(2), but the curves of TurKontrol(3) and TurKontrol(4) generally overlap. We also observe that when the costs are high, the performance difference between TurKontrol(2) and deeper step lookaheads is negligible. Since each additional step of lookahead increases the computational overhead by an order of magnitude, we limit TURKONTROL' lookahead to depth 2 in subsequent experiments.

The Effect of Poor Workers We now consider the effect of worker accuracy on the effectiveness of agent control policies. Using fixed costs of (30,10), we compare the average net utility of three control policies. The first is TurKontrol(2). The second, TurKit, is a fixed policy from the literature [4]; it performs as many iterations as possible until its fixed allowance (400 in our experiment) is depleted and on each iteration it does at least two ballots, invoking a third only if the first two disagree. Our third policy, TurKontrol(fixed), combines elements from decision theory with a fixed policy. After simulating the behavior of TurKontrol(2), we compute the integer mean number of iterations, μ_{imp} and mean number of ballots, μ_b, and use these values to drive a fixed control policy (μ_{imp} iterations each with μ_b ballots), whose parameters are tuned to worker fees and accuracies.

Figure 4 shows that both decision-theoretic methods work better than the TurKit policy, partly because TurKit runs more iterations than needed. A Student's t-test show all differences are statistically significant with p value 0.01. We also note that the performance of TurKontrol(fixed) is very similar to that of TurKontrol(2), when workers are very inaccurate, γ=4. Indeed, in this case TurKontrol(2) executes a nearly fixed policy itself. In all other cases, however, TurKontrol(fixed) consistently underperforms TurKontrol(2). A Student's t-test results confirm the differences are all statistically significant for $\gamma < 4$. We attribute this difference to the fact that the dynamic policy makes better use of ballots, *e.g.*, it requests more ballots in late iterations, when the (harder) improvement tasks are more error-prone. The biggest performance gap between the two policies manifests when γ=2, where TurKontrol(2) generates 19.7% more utility than TurKontrol(fixed).

Conclusions and Future Work

We introduce an exciting new application for artificial intelligence — control of crowd-sourced workflows. We use decision-theory to model a popular class of iterative workflows and define equations that govern the various steps of the process. We show that our agent, TURKONTROL, which implements our mathematical framework and uses it to optimize and control the workflow is robust in a variety of scenarios and parameter settings, and results in higher utilities than previous, fixed policies. To make our model more general and realistic we plan to perform three important, next steps.

First, we need to develop schemes to quickly and cheaply learn the two sets of parameters required by our decision-theoretic model, the accuracy of an improvement job and of a ballot job per worker. We can divide the learning task into two steps, learning the ballot accuracy models in the first step and the improvement accuracy models in the second. In both steps we use several pictures with one artifact each and let multiple workers improve an artifact. After improvements, we let several other workers vote on the two artifacts. Given the results, we plan to infer the model by solving convex optimization problems.

Second, we hope to generalize our ballot questions to get more informed feedback from the voters. In general a ballot job could ask about the workers confidence, such as, "How sure are you that α is better than α'?" Or one could get an estimate of the quality difference, such as, "Do you think α significantly improves/marginally improves/is no different from/marginally downgrades/messes up α'?" If we could use such questions to our advantage we can save on significant cost and increase the total throughput of the platform.

Third, we want to look at how to set up an intelligent payment structure to get the most qualified results and achieve the fastest throughput. For an automated agent the decision question will be (1) when to pay a bonus, and (2) 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 savings could be used to reward the workers who did well in this task.

As the long term goal, we plan to move beyond simulations, validating our approach on actual MTurk workflows. Finally, we plan to release a user-friendly toolkit that implements our decision-theoretic control regime and which can be used by requesters on MTurk and other crowd-sourcing platforms.

Acknowledgments

This work was supported by Office of Naval Research grant N00014-06-1-0147 and the WRF / TJ Cable Professorship. We thank James Fogarty and Greg Little for helpful discussions and Greg Little for providing the TurKit code. We benefitted from data provided by Smartsheet.com. Comments from Andrey Kolobov and anonymous reviewers significantly improved the paper.

References

[1] Peng Dai, Mausam, and Daniel S. Weld. Decision-theoretic control for crowd-sourced workflows. In *AAAI*, 2010.

[2] L. Hoffmann. Crowd control. *C. ACM*, 52(3):16–17, March 2009.

[3] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101:99–134, 1995.

[4] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. TurKit: Tools for Iterative Tasks on Mechanical Turk. In *Human Computation Workshop (HComp2009)*, 2009.

[5] Contact center in the cloud, December 2009. `http://liveops.com`.

[6] Mechanical turk is a marketplace for work, December 2009. `http://www.mturk.com/mturk/welcome`.

[7] Topcoder, December 2009. `http://topcoder.com`.

[8] Crowdsourcing, December 2009. `http://en.wikipedia.org/wiki/Crowdsourcing`.