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

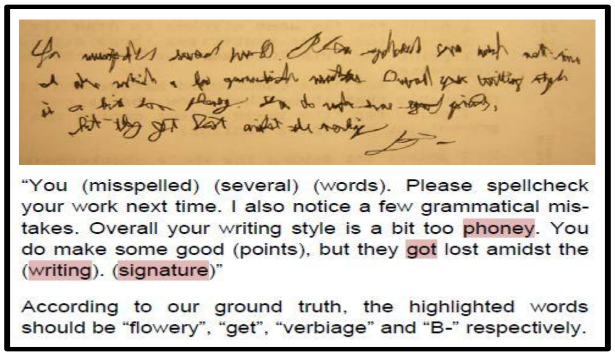
Crowd-sourcing is a recent framework in which human intelligence tasks are outsourced to a crowd of unknown people (”workers”) as an open call (e.g., on Amazon’s Mechanical Turk). Crowd-sourcing has become immensely popular with hoards of employers (”requesters”), who use it to solve a wide variety of jobs, such as dictation transcription, content screening, etc. In order 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 paper 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 describe 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.

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” [21], has the potential to revolutionize information-processing services by quickly coupling human workers with software automation in productive workflows [8].

While the phrase ‘crowd-sourcing’ was only termed in 2006, the area has grown rapidly in economic significance with the growth of general-purpose platforms such as Amazon’s *Mechanical Turk* [15] and task-specific sites for call centers [13], programming jobs [19] and more. Recent research has shown surprising success in solving difficult tasks using the strategy of incremental improvement in an iterative workflow [12]; 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 fed as the input to the next. Additional voting jobs were used to assess whether a worker actually improved the transcription compared to the prior effort.



“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 [12].

From an AI perspective, crowd-sourced workflows offer a new, exciting and impactful application area for intelligent control. Although there is a vast literature on decision-theoretic planning and execution (*e.g.*, [17; 2; 9]), it appears that these techniques have yet to be applied to control a crowd-sourcing platform. 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)?

This paper offers initial answers to these questions by presenting a decision-theoretic planner that dynamically optimizes iterative workflows to achieve the best quality/cost tradeoff. We make the following contributions:

- We introduce the AI problem of optimization and control of iterative workflows over a crowd-sourcing platform.
- We develop a mathematical theory for optimizing the quality/cost tradeoff for a popular class of workflows.
- We implement an agent, TURKONTROL, for taking deci-

sions at each step of the workflow based on the expected utilities of each action.

- We simulate TURKONTROL in a variety of complex scenarios and find that it behaves robustly. We also show that TURKONTROL's decisions result in a significantly higher final utility compared to fixed policies and other baselines.

Background

While the ideas in our paper are applicable to different workflows, for our case study we choose the iterative workflow introduced by Little *et al.* [12] depicted in Figure 2. This particular workflow is representative of a number of flows in commercial use today; at the same time, it is moderately complex, making it ideal for first investigation.

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 one or more *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. Little *et al.* have shown that this iterative process generates better descriptions for a fixed amount than allocating the total reward to a single author.

Little *et al.* support an open-source toolkit, TurKit, that provides a high-level mechanism for defining moderately complex, iterative workflows with voting-controlled conditionals. However, TurKit doesn't have built-in methods for monitoring the accuracy of workers; nor does TurKit automatically determine the ideal number of voters or estimate the appropriate number of iterations before returns diminish. Our mathematical framework in the next section answers these and other questions.

Decision-Theoretic Optimization

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 (or a job description for requesting one), and the agent is asked to return an artifact which maximizes some payoff based on the quality of the submission. Intuitively, something is high-quality if it is better than most things of the same type. For engineered artifacts (including English descriptions) one may say that something is high-quality if it is difficult to improve. Therefore, we define the *quality* of an artifact by $q \in [0, 1]$; an artifact with quality q means an average dedicated worker has probability $1 - q$ of improving it. In our initial model, 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 (*e.g.*, how many workers prefer it to another artifact).

Figure 3 summarizes a high-level flow for our planner's decisions. At each step we track our belief in qualities (q and q') of the previous (α) and the current artifact (α') respectively. Each decision or observation gives us new informa-

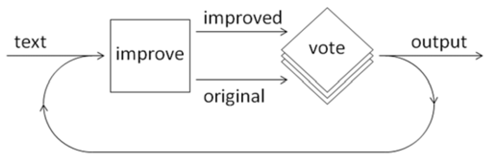


Figure 2: Flowchart for the iterative text improvement task, reprinted from [12].

tion, which is reflected in the quality posteriors. These distributions also depend on the accuracy of workers, which we also incrementally estimate based on their previous work.

Based on these distributions we estimate expected utilities for each action. This lets us answer questions like (1) when to terminate the voting phase (thus switching attention to artifact improvement), (2) which of the two artifacts is the best basis for subsequent improvements, and (3) when to stop the whole iterative process and submit the result to the requester.

One may view this agent-control problem as a partially-observable Markov Decision problem (POMDP) [9]. The actual state of the system, (q, q'), is only partially observable. The agent's knowledge of this state can be represented by a belief state comprising the joint probability distribution over q and q'. There are three actions. Requesting a ballot job is a pure observation action — it changes neither q or q', but when a worker says she prefers one artifact, this observation allows the agent to update the probability distributions for the likely values. Posting an improvement job changes more than the agent's belief, because one artifact, say α', is replaced with a modified version of the other; the agent's belief about q' also changes. Submitting whichever artifact is believed to be best directs the system to a terminal state and produces a reward. Since qualities are real numbers, the state space underlying the beliefs is also continuous [3]. These kind of POMDPs are especially hard to solve. We overcome this computational bottleneck by performing limited lookahead search to make planning more tractable.

Below, we present our mathematical analysis in detail. It is divided into three key stages: quality posteriors after improvement or a new ballot, utility computations for the available actions and finally the decision-making algorithm and implementation details.

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', whose quality is denoted by q'. Since α' is a suggested improvement of α, q' depends on the initial quality q. Moreover, a higher accuracy worker x may improve it much more so it depends on x. We define $f_{Q'|q,x}$ as the conditional quality distribution of q' when worker x improved an artifact of quality q. This function describes the dynamics of the domain. With a known $f_{Q'|q,x}$ we can easily compute the prior

[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.

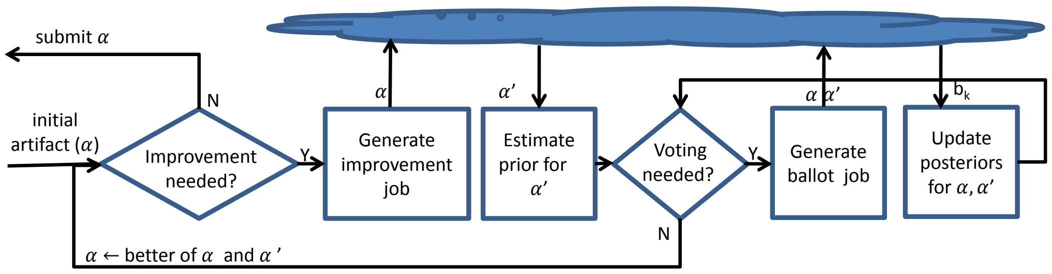

submit α
N
initial artifact (α)
Improvement needed?
Y
α
Generate improvement job
α'
Estimate prior for α'
Voting needed?
Y
α α'
Generate ballot job
b_k
Update posteriors for α, α'
N
$\alpha \leftarrow$ better of α and α'

Figure 3: Computations needed by TURKONTROL for control of an iterative-improvement workflow.

on q' from the law of total probability:

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

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. The worker may have done an excellent or a miserable job. Even if the new artifact *is* an improvement, one needs to assess how much better the new artifact actually is. Our workflow at this point tries to gather evidence to answer these questions by generating ballots and asking new workers a question: "Is α' a better answer than α for the original question?". 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 quality posteriors, $f_{Q|\overrightarrow{\mathbf{b}^n}}$ and $f_{Q'|\overrightarrow{\mathbf{b}^n}}$. These posteriors have three roles to play. First, more accurate beliefs lead to a higher probability of keeping the better artifact for subsequent phases. Second, within the voting phase confident beliefs help decide when to stop voting. Third, a high-quality belief also helps one decide when to quit the iterative process and submit. In order 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. Of course she may still make mistakes, so it is useful for the controller to model her accuracy. Second, we assume that several workers will not collaborate adversarially to defeat the system.

These assumptions might lead one to believe that the probability distributions for worker responses ($P(b_i)$) are independent of each other. Unfortunately, this independence is violated due to a subtlety — even though different workers are not collaborating, a mistake by one worker changes the error probability of others. This happens because one worker's mistake provides evidence that the question may be intrinsically hard — which increases the chance that other workers may err as well. To get around this problem we introduce the *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. We define the relationship between the difficulty and qualities as

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

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. One will expect everyone's accuracy to be monotonically decreasing in d. It will approach random behavior as questions get really hard, *i.e.*, $a_x(d) \rightarrow 0.5$ as $d \rightarrow 1$. Similarly, as $d \rightarrow 0$, $a_x(d) \rightarrow 1$. 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. It is easy to check that this polynomial function satisfies all the conditions when $d \in [0, 1]$. Note that smaller the γ_x the more concave the accuracy curve, and thus greater the expected accuracy for a fixed d.

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:

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

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 $f_{Q'|\overrightarrow{\mathbf{b}^n}}(q)$. Abusing notation slightly, we use $\overrightarrow{\mathbf{b}^n+1}$ to symbolically denote that n ballots are known and another ballot (value currently unknown) will be received in the future. By applying the Bayes rule one gets

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

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

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

The same sequence of steps can be used to compute the posterior of α'.

$$\begin{aligned} f_{Q'|\overrightarrow{\mathbf{b}^n+1}}(q') &\propto P(b_{n+1} \mid q', \overrightarrow{\mathbf{b}^n}) f_{Q'|\overrightarrow{\mathbf{b}^n}}(q') & (7) \\ &= P(b_{n+1} \mid q') f_{Q'|\overrightarrow{\mathbf{b}^n}}(q') & (8) \\ &= \left[\int_0^1 P(b_{n+1}|q, q') f_{Q|\overrightarrow{\mathbf{b}^n}}(q) dq \right] f_{Q'}(q') \end{aligned}$$

Discussion: Why should our belief in the quality of the previous artifact change (posterior of α) based on ballots comparing it with the new artifact? This is a subtle, but important point. If the improvement worker (who has a good accuracy) was unable to create a much better α' in the improvement phase that must be because α already has a high quality and is no longer easily improvable. Under such evidence we should increase quality of α, which is reflected by the posterior of α, $f_{Q|\overrightarrow{\mathbf{b}}}$. Similarly, if all voting workers unanimously thought that α' is much better than α, it means the ballot was very easy, *i.e.*, α' incorporates significant improvements over α and the qualities should reflect that.

This computation helps us determine the prior quality for the artifact in the next iteration. It will be either $f_{Q|\overrightarrow{\mathbf{b}}}$ or $f_{Q'|\overrightarrow{\mathbf{b}}}$ (Equations 5 and 8), depending on whether we decide to keep α or α'.

Utility Estimations

We now discuss how to estimate the utility of an additional ballot. At this point, say, one has already received n ballots ($\overrightarrow{\mathbf{b}^n}$) and the posteriors of the two artifacts $f_{Q|\overrightarrow{\mathbf{b}^n}}$ and $f_{Q'|\overrightarrow{\mathbf{b}^n}}$ are available. 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':

$$U_{\overrightarrow{\mathbf{b}^n}} = max\{E[U(Q|\overrightarrow{\mathbf{b}^n})], E[U(Q'|\overrightarrow{\mathbf{b}^n})]\}, \text{ where} \quad (9)$$

$$E[U(Q|\overrightarrow{\mathbf{b}^n})] = \int_0^1 U(q) f_{Q|\overrightarrow{\mathbf{b}^n}}(q) dq \quad (10)$$

$$E[U(Q'|\overrightarrow{\mathbf{b}^n})] = \int_0^1 U(q') f_{Q'|\overrightarrow{\mathbf{b}^n}}(q') dq' \quad (11)$$

Using $U_{\overrightarrow{\mathbf{b}^n}}$ we need to compute the utility of taking an additional ballot, $U_{\overrightarrow{\mathbf{b}^n+1}}$. 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). However, since we do not know which worker will take our ballot job, we assume anonymity and expect an average worker $\overline{x}$ with the accuracy function $a_{\overline{x}}(d)$. Recall from Equation 2 that difficulty, d, is a function of similarity in qualities. Because q and q' are not exactly known, probability of getting the next ballot is computed by applying the law of total probability to 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):

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

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

We can write a similar equation for $E[U(Q' \mid \overrightarrow{\mathbf{b}^n+1})]$.

Similarly, we can compute the utility of an improvement step. Based on the current beliefs on the qualities of α and α' and Equation 9 we can choose α or α' as the better artifact. The belief of the chosen artifact acts as f_Q for Equation 1 and we can 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 and Other Updates

Decision Making: At any step one 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.

Of course, a greedy policy may be much worse than the optimal. One can compute a better policy by an l-step lookahead algorithm that 1) evaluates all sequences of l decisions, 2) finds the sequencewith the highest expected utility, 3) executes the first action of the sequence, and 4) repeats. As an arbitrarily good policy might have a sub-optimal l-step prefix, one cannot bound the deviation from optimality produced by any finite-step lookahead algorithm. However, the l-step lookahead works well in simulations (as shown in the next section).

Updating Difficulty and Worker Accuracy: The agent updates its estimated d before each decision point based on its estimates of qualities as follows:

$$\begin{aligned} d^* &= \int_0^1 \int_0^1 d(q,q') f_Q(q) f_{Q'}(q') dq dq' \\ &= \int_0^1 \int_0^1 (1 - |q,q'|^{\mathcal{M}}) f_Q(q) f_{Q'}(q') dq dq' \quad (12) \end{aligned}$$

After completing each iteration we have access to estimates for d^* and the believed answer. We can use this information to update our record on the quality of each worker. In particular, if someone answered a question correctly then she is a good worker (and her γ_x should decrease) and if someone made an error in a question her γ_x should increase. Moreover the increase/decrease amounts should depend on the difficulty of the question. The following simple update strategy may work:

1. If a worker answered a question of difficulty d correctly then $\gamma_x \leftarrow \gamma_x - d\delta$.
2. If a worker made an error when answering a question then $\gamma_x \leftarrow \gamma_x + (1-d)\delta$.

We use δ to represent the learning rate, which could be slowly reduced over time so that the accuracy of a worker approaches an asymptotic distribution.

Implementation: In a general model such as ours maintaining a closed form representation for all these continuous functions may not be possible. Uniform discretization is the simplest way to approximate these general functions. However, for efficient storage and computation TurKontrol could employ piecewise constant/piecewise linear value function representations or use particle filters. Even though approximate both these techniques are very popular in the literature for efficiently maintaining continuous distributions [14; 5] and can provide arbitrarily close approximations. Because some of our equations require double integrals and can be time consuming (*e.g.*, Equation 12) these compact representations help in overall efficiency of the implementation.

Experiments

This section aims to empirically answer the following questions: 1) How deep should an agent's lookahead be to best tradeoff between computation time and utility? 2) Does TURKONTROL make better decisions compared to fixed policies such as TurKit's or other better informed ones? 3) How does TURKONTROL compare to other sophisticated planning algorithms such as POMDP solvers?

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 $\text{Beta}(1,9)$, which implies that the mean quality of the first artifact is 0.1. Suppose the quality of the current artifact is q, we assume the conditional distribution $f_{Q'|q,x}$ is Beta distributed, with mean $\mu_{Q'|q,x}$ where:

$$\mu_{Q'|q,x} = q + 0.5[\ (1-q)\times(a_x(q)-0.5) \\ +q\times(a_x(q)-1)\ ]. \quad (13)$$

and the conditional distribution is $\text{Beta}(10\mu_{Q'|q,x}, 10(1-\mu_{Q'|q,x}))$. We know a higher quality means it's less likely the artifact can be improved. We model results of an improvement task, in a manner akin to ballot tasks; the resulting distribution of qualities is influenced by the worker's accuracy and the improvement difficulty, $d = q$.

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 4 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

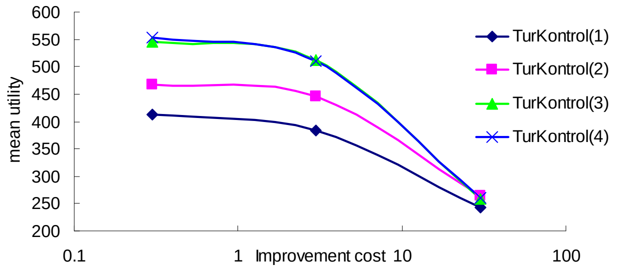


Figure 4: Average net utility of TURKONTROL with various lookahead depths calculated using 10,000 simulation trials on three sets of (improvement, ballot) costs: (30,10), (3,1), and (0.3,0.1). Longer lookahead produces better results, but 2-step lookahead is good enough when costs are relatively high: (30,10).

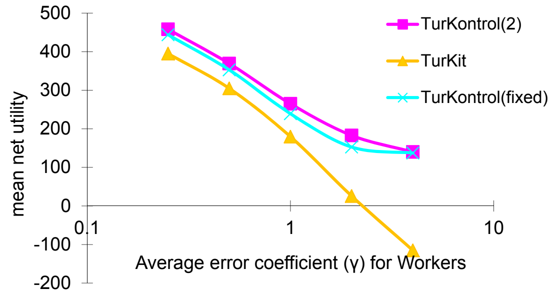


Figure 5: Net utility of three control policies averaged over 10,000 simulation trials, varying mean error coefficient, γ. TurKontrol(2) produces the best policy in every cases.

when the costs are high, such that the process usually finishes in a few iterations, 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.

Comparing to Fixed Policies

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 [12]; 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 5 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 shows all differences are statistically significant ($p < 0.01$). 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).

Robustness in the Face of Bad Voters As a final study, we considered the sensitivity of the previous three policies to increasingly noisy voters. Specifically, we repeated the previous experiment using the same error coefficient, γ_x, for each worker's improvement *and ballot* behavior. (Recall, that we previously set the error coefficient for ballots to one half γ_x to model the fact that voting is easier.) The resulting graph (not shown) has the same shape as that of Figure 5 but with lower overall utility. Once again, TurKontrol(2) continues to achieve the highest average net utility across all settings. Interestingly, the utility gap between the two TurKontrol variants and TurKit is consistently bigger for all γ than in the previous experiment. In addition, when γ=1, TurKontrol(2) generates 25% more utility than TurKontrol(fixed) — a bigger gap seen in the previous experiment. A Student's t-test shows all that the differences between TurKontrol(2) and TurKontrol(fixed) are significant when $\gamma < 2$ and the differences between both TurKontrol variants and TurKit are significant at all settings.

Comparing to Other Algorithms

In addition to our limited lookahead search, we implement two additional algorithms to improve planning quality – an approximate continuous POMDP solver and a UCT-based algorithm. Overall we find that TurKontrol outperforms both of these.

We implement a continuous POMDP solver that approximates and discretizes the belief space into a finite-state MDP and later solves the MDP with value iteration. We call it the ADBS method, short for approximation and discretization of belief space. Our method approximates a general distribution by two values – its mean and standard deviation. This transforms the belief space to a four-tuple $(\mu, \sigma, \mu', \sigma')$, one pair each for quality distributions of current and new artifact. We then discretize the four variables into small, equal-sized intervals. As quality is a real number in [0,1], $\mu \in [0,1]$ and $\sigma \in [0,1]$. With discretization and the known bounds the belief space becomes finite. To generate the MDP, ADBS uses breadth-first search. We try different resolutions of discretization, *i.e.* various interval lengths, and run with average error coefficient γ=1 on three pairs of improvement and ballot costs — (30,10), (3,1), and (0.3,0.1). Results (not displayed) show that TurKontrol(2) outperforms ADBS in all settings. We also find that with more refined discretization, the reachable state space grows very quickly (approximately an order of magnitude per double refined interval), yet the performance of ADBS improves very slowly. In the future

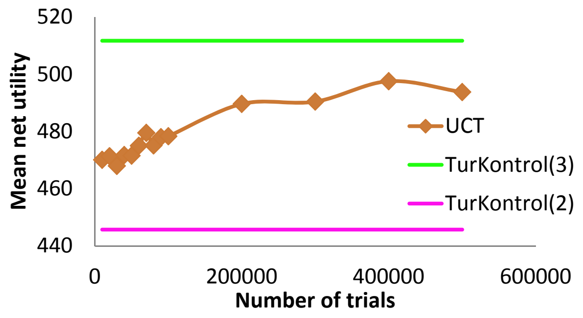


Figure 6: Net utility of the UCT algorithm after different number of trials averaged over 10,000 simulation trials compared to TurKontrol(3) and (2), which do not use trials and hence appear as horizontal lines. Improvement and ballot costs are set to (3,1), but results are similar for other cost settings. Note that TurKontrol(3) returns the best results.

we plan to try other more sophisticated POMDP algorithms such as [3].

Recently a general Monte-Carlo simulation algorithm named UCT (short for "Upper Confidence bounds applied on Trees") [10] was proposed to solve planning problems that are too large to be solved optimally. Its efficacy has been demonstrated on challenging problems such as Go [7] and realtime strategy games [1]. We apply the UCT algorithm on the an approximated belief space MDP where every worker is assumed anonymous with average accuracy $a_{\overline{x}}(d)$.

Figure 6 plots the mean net utility of UCT as a function of the number of completed trails (10,000 to 500,000) when the costs are (3,1). We also tried other costs, and got very similar results. The horizontal lines stand for the net utilities of TurKontrol(3) (upper) and TurKontrol(2) (lower) (data used in Figure 4). We note that the performance of UCT improves with the number of completed trials. We also observe that TurKontrol(3) outperforms UCT, though the performance of UCT is quite good (better than for TurKontrol(2) even after just 10,000 trials). This experiment suggests that UCT is useful for quickly finding a good sub-optimal policy but has limited power in finding a close to optimal policy in the long run.

Related Work

Automatically controlling a crowd-sourcing system may be viewed as an agent-control problem where the crowd-sourcing platform embodies the agent's environment. In this sense, previous work on the control of software agents, such as the Internet Softbot [6] and CALO [16] is relevant. However, in contrast to previous systems, our situation is more circumscribed; hence, a narrower form of decision-theoretic control suffices. In particular, there are a small number of parameters for the agent to control when interacting with the environment.

A related problem is the scheduling of a dynamic processing system given time constraints. Progressive processing [22] is one way to solve it, however, its model uncertainty lies in action durations, and the states are fully observable.

Several researchers are studying crowd-sourcing systems from different perspectives. Ensuring accurate results is one essential challenge in any crowd-sourcing system. Snow *et al.* [18] observe that for five linguistics tasks, the quality of results obtained by voting a small number inexperienced workers can exceed that of a single expert, depending on task, but they provide no general method for determining the number of voters *a priori*.

Several tools are being developed to facilitate parts of this crowd-sourcing process. For example, TurKit [12], `Crowdflower.com`'s *CrowdControl*, and `Smartsheet.com`'s *Smartsourcing* provide simple mechanisms for generating iterative workflows, and integrating crowd-sourced results into an overall workflow. All these tools provide operational conveniences rather than any decision support.

Crowd-sourcing can be understood as a form of human computation, where the primary incentive is economical. Other incentive schemes include fun, altruism, reciprocity, reputation, *etc.* In projects such as Wikipedia and open-source software development, community-related motivations are extremely important [11]. Von Ahn and others have investigated *games with a purpose* (GWAP), designing fun experiences that produce useful results such as image segmentation, optical character recognition [20]. Crowdflower has integrated Mechanical Turk job streams into games [4] and developed a mechanism whereby workers can donate their earnings to double the effective wage of crowd-workers in impoverished countries — thus illustrating the potential to combine multiple incentive structures.

Conclusions

We introduce an exciting new application for artificial intelligence — control of crowd-sourced workflows. Complex workflows have become commonplace in crowd-sourcing and are regularly employed for high-quality output. We use decision theory to model a popular class of iterative workflows and define equations that govern the various steps of the process. Our agent, TURKONTROL, implements our mathematical framework and uses it to optimize and control the workflow. Our simulations show that TURKONTROL is robust in a variety of scenarios and parameter settings, and results in higher utilities than previous, fixed policies.

We believe that AI has the potential to impact the growing thousands of requesters who use crowd-sourcing by making their processes more efficient. To realize our mission we plan to perform three important, next steps. First, we need to develop schemes to quickly and cheaply learn the many parameters required by our decision-theoretic model. Secondly, we need 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, Jieling Han, Yingyi Bu and anonymous reviewers significantly improved the paper.

References

[1] Radha-Krishna Balla and Alan Fern. UCT for tactical assault planning in real-time strategy games. In *IJCAI*, pages 40–45, 2009.

[2] D. Bertsekas. *Dynamic Programming and Optimal Control, Vol 1, 2nd ed.* Athena Scientific, 2000.

[3] E. Brunskill, L.Kaelbling, T.Lozano-Perez, and N. Roy. Continuous-state POMDPs with hybrid dynamics. In *ISAIM'08*, 2008.

[4] Getting the gold farmers to do useful work, October 2009. `http://blog.doloreslabs.com/`.

[5] A. Doucet, N. De Freitas, and N.J. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[6] O. Etzioni and D. Weld. A softbot-based interface to the Internet. *C. ACM*, 37(7):72–6, 1994.

[7] Sylvain Gelly and Yizao Wang. Exploration exploitation in go: UCT for monte-carlo go. In *NIPS On-line trading of Exploration and Exploitation Workshop*, 2006.

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

[9] L. Kaebling, M. Littman, and T. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2):99–134, 1998.

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

[11] S. Kuznetsov. Motivations of contributors to Wikipedia. *ACM SIGCAS Computers and Society*, 36(2), June 2006.

[12] 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.

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

[14] Mausam, Emmanuel Benazera, Ronen Brafman, Nicolas Meuleau, and Eric Hansen. Planning with continuous resources in stochastic domains. In *IJCAI'05*, 2005.

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

[16] B. Peintner, J. Dinger, A. Rodriguez, and K. Myers. Task assistant: Personalized task management for military environments. In AAAI Press, editor, *IAAI-09*, 2009.

[17] S. Russell and E. Wefald. *Do the Right Thing*. MIT Press, Cambridge, MA, 1991.

[18] 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.

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

[20] Luis von Ahn. Games with a purpose. *Computer*, 39(6):92–94, 2006.

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

[22] Shlomo Zilberstein and Abdel-Illah Mouaddib. Optimal scheduling of progressive processing tasks. *International Journal of Approximate Reasoning*, 25(3):169–186, 2000.