Artificial Intelligence 202 (2013) 52–85

Contents lists available at SciVerse ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

POMDP-based control of workflows for crowdsourcing

Peng Dai *, Christopher H. Lin, Mausam, Daniel S. Weld

Department of Computer Science and Engineering, University of Washington, Seattle, WA 98195, United States

A R T I C L E   I N F O

Article history:
Received 20 December 2011
Received in revised form 8 June 2013
Accepted 9 June 2013
Available online 20 June 2013

Keywords:
Partially-Observable Markov Decision Process
POMDP
Planning under uncertainty
Crowdsourcing

A B S T R A C T

Crowdsourcing, outsourcing of tasks to a crowd of unknown people ("workers") in an open call, is rapidly rising in popularity. It is already being heavily used by numerous employers ("requesters") for solving a wide variety of tasks, such as audio transcription, content screening, and labeling training data for machine learning. However, quality control of such tasks continues to be a key challenge because of the high variability in worker quality. In this paper we show the value of decision-theoretic techniques for the problem of optimizing workflows used in crowdsourcing. In particular, we design AI agents that use Bayesian network learning and inference in combination with Partially-Observable Markov Decision Processes (POMDPs) for obtaining excellent cost-quality tradeoffs.

We use these techniques for three distinct crowdsourcing scenarios: (1) control of voting to answer a binary-choice question, (2) control of an iterative improvement workflow, and (3) control of switching between alternate workflows for a task. In each scenario, we design a Bayes net model that relates worker competency, task difficulty and worker response quality. We also design a POMDP for each task, whose solution provides the dynamic control policy. We demonstrate the usefulness of our models and agents in live experiments on Amazon Mechanical Turk. We consistently achieve superior quality results than non-adaptive controllers, while incurring equal or less cost.

1. Introduction

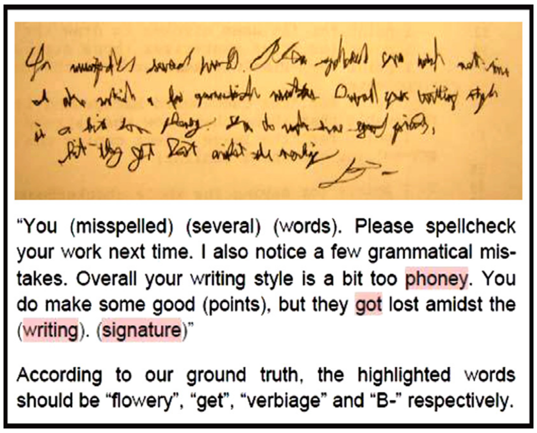
The ready availability of the Internet across the world has had far-reaching consequences. Not only has widespread online connectivity revolutionized communication, politics, entertainment, and other aspects of everyday life, it has enabled the ability to easily unite large groups of people around the world (a crowd) for a common purpose. This ability to *crowdsource* has in turn opened up radical new possibilities and resulted in novel successes, like Wikipedia[1] – a whole encyclopedia written by the crowd, and platforms that are rapidly impacting the world's laborforce by bringing them together in online marketplaces that provide work on-demand.

We believe that *Crowdsourcing*, "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,"[2] has the potential to revolutionize information-processing services by coupling human workers with intelligent machines in productive workflows [21].

* Corresponding author. Current affiliation: Google Inc., 1600 Amphitheater Pkwy, Mountain View, CA 94043.
E-mail addresses: daipeng@cs.washington.edu (P. Dai), chrislin@cs.washington.edu (C.H. Lin), mausam@cs.washington.edu (Mausam), weld@cs.washington.edu (D.S. Weld).

[1] http://en.wikipedia.org.
[2] http://en.wikipedia.org/wiki/Crowdsourcing.

0004-3702/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.artint.2013.06.002



"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.

Fig. 1. A handwriting recognition task (almost) successfully solved on Mechanical Turk using an iterative-improvement workflow. Workers were shown the text written by a human and in a few iterations they deduced the message (with minimal errors highlighted). Figure adapted from [37].

While the word "crowdsourcing" was only coined in 2006, the area has grown rapidly in economic significance with the emergence of general-purpose platforms such as Amazon's *Mechanical Turk*,[3] task-specific sites for call centers,[4] programming jobs[5] and more [37,4,7,33,24].

Crowdsourced workers are motivated by a variety of incentives. Common incentives include entertainment, *e.g.*, in the context of playing games with a purpose [63,10], contribution to science,[6] a sense of community, and monetary rewards. While our work is applicable to several crowdsourcing scenarios, we focus on financially motivated micro-crowdsourcing – crowdsourcing of small jobs in exchange for monetary payments. Labor markets, like oDesk, handle medium and large-sized tasks requiring a diverse range of skills and micro-crowdsourcing has become very popular with requesters, who use a concert of small-sized jobs to handle a wide variety of higher-level tasks, such as audio transcription, language translation, calorie counting [42], and helping blind people navigate unknown surroundings [7]. It has also been immensely popular with workers in both developed and developing countries [49].

On popular micro-crowdsourcing platforms, like Mechanical Turk, requesters often use *workflows*, a series of steps, to complete tasks. For instance, for a simple image classification task (*e.g.* "Is there a lion in this picture?"), a workflow as simple as asking several workers the same binary-choice question will suffice. For a more difficult task like the handwriting recognition task shown in Fig. 1, a requester might create a more complicated workflow like *iterative improvement*, in which workers incrementally refine each others' solutions until no further improvement is necessary [37]. A plethora of research shows that for simple binary classification workflows, micro-crowdsourcing can achieve high-quality results [58,60]. Similarly, iterative improvement, find–fix–verify [3] and other workflows have been shown to yield excellent output even when individual workers err.

But the use of these workflows raises many important questions. For example, when designing a workflow to handle problems like the handwriting recognition task shown in Fig. 1, we do not know answers to questions like: (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 depending on workers' skill levels?

Our paper offers answers to these questions by constructing AI agents for workflow optimization and control. We study three distinct crowdsourcing scenarios. We start by considering the problem of dynamically controlling the aggregation of the simplest possible workflow: binary classification. Next, we extend our model to control the significantly more complex iterative improvement workflow. Finally, we show how our model can be used to dynamically switch between alternative workflows for a given task.

We use a shared toolbox of techniques on each of these crowdsourcing scenarios; there are two key components. First, we propose a probabilistic model for worker responses. This model relates worker ability, task difficulty and worker response quality by means of a Bayesian network. Second, we view the problem of workflow control as a problem of decision-theoretic planning and execution, and cast it as a Partially-Observable Markov Decision Process (POMDP) [6]. The Bayes net provides the model for the POMDP, and the POMDP policy controls the workflow to obtain a high-quality output in a cost-efficient manner.

We make several important contributions. First, we provide a principled solution to dynamically decide the number of votes for a task based on the exact history of workers who worked on a task instance, their answers and a notion of

[3] http://mturk.com.
[4] http://liveops.com.
[5] http://topcoder.com.
[6] http://galaxyzoo.org, Audubon Christmas Bird Count, etc.

varying problem difficulty. We also provide a framework for answering more complex questions in larger workflows (*e.g.* the number of iterations in an iterative improvement workflow). The commonality of approaches in these diverse tasks suggests an underlying abstraction, which may be exploited for optimizing and controlling many different kinds of workflows. Our experiments consistently outperform best known baselines by large margins, *e.g.*, obtaining 50% error reduction in the scenario of multiple workflows, or 30% cost savings for iterative improvement.

Moreover, our work reveals some surprising discoveries. First, for the iterative improvement scenario, our AI agent proposes a rather unexpected voting policy, which was not predicted by any human expert (see Section 4). Second, we demonstrate that judiciously using alternative workflows for a task is much better than using a single "best" workflow. While the base idea is not novel, the fact that we can do this dynamic switching between workflows automatically to obtain high-quality results is a surprising discovery.

The rest of the paper is structured as follows. Section 2 formally defines Markov Decision Processes and Partially-Observable Markov Decision Processes (POMDP). Then we review and propose planning algorithms for solving POMDPs. Section 3 details how we model and control the aggregation of a simple binary classification workflow using our first agent, TurKontrol$_0$. Section 4 extends our model and agent to iterative improvement workflows to create our second agent, TurKontrol. We present some simulation-based investigation of the performance of TurKontrol, illustrate how to learn the model parameters, and finally demonstrate the usefulness of TurKontrol on Mechanical Turk, with real tasks. Section 5 details how we model the availability of multiple workflows and the design of our third agent, AgentHunt, which dynamically selects the next best workflow to use. We illustrate how to learn model parameters and then demonstrate the usefulness of AgentHunt in simulation and on Mechanical Turk. Finally, we present related work, propose future work, and make conclusions. We note software packages of our implementations are available for general use at http://cs.washington.edu/node/7714.

2. Background

2.1. Markov decision processes

AI researchers typically use Markov Decision Processes (MDPs) to formulate fully-observable decision making under uncertainty problems [40].

Definition 1. An MDP is a four-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} \rangle$, where

- $\mathcal{S}$ is a finite set of discrete states.
- $\mathcal{A}$ is a finite set of all actions.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ is the transition function describing the probability that taking an action in a given state will result in another state.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward for taking an action in a state.

An agent executes its actions in discrete time steps starting from some initial state. At each step, the system is at one distinct state $\mathbf{s} \in \mathcal{S}$. The agent can execute any action $\mathbf{a}$ from a set of *actions* $\mathcal{A}$, and receives a reward $R(\mathbf{s}, \mathbf{a})$. The action takes the system to a new state $\mathbf{s}'$ stochastically. The transition process exhibits the *Markov property*, *i.e.*, the new state $\mathbf{s}'$ is independent of all previous states given the current state $\mathbf{s}$. The transition probability is defined by $T_{\mathbf{a}}(\mathbf{s}'|\mathbf{s})$. The model assumes *full observability*, *i.e.*, after executing an action and transitioning stochastically to a next state as governed by $\mathcal{T}$, the agent has full knowledge of the state.

Any solution to an MDP problem is in the form of a *policy*.

Definition 2. A policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, of an MDP is a mapping from the state space to the action space.

A policy is *static* if the action taken at each state is the same at every time step. $\pi(\mathbf{s})$ indicates which action to execute when the system is at state $\mathbf{s}$. To solve an MDP we need to find an *optimal policy* ($\pi^* : \mathcal{S} \rightarrow \mathcal{A}$), a probabilistic execution plan that achieves the maximum expected utility, or sum of rewards. We evaluate any policy π by its *value function*, the set of values that satisfy the following equation:

$$V^{\pi}(\mathbf{s}) = R\big(\mathbf{s}, \pi(\mathbf{s})\big) + \beta \sum_{\mathbf{s}' \in \mathcal{S}} T_{\pi(\mathbf{s})}\big(\mathbf{s}'\big|\mathbf{s}\big) V^{\pi}\big(\mathbf{s}'\big). \tag{1}$$

$\beta \in (0, 1]$ is the *discount factor*, which controls the value of future rewards. Any optimal policy's value function must satisfy the following system of *Bellman equations*:

$$V^*(\mathbf{s}) = \max_{\mathbf{a} \in A} \left[R(\mathbf{s}, \mathbf{a}) + \beta \sum_{\mathbf{s}' \in \mathcal{S}} T_{\mathbf{a}}\big(\mathbf{s}'\big|\mathbf{s}\big) V^*\big(\mathbf{s}'\big) \right]. \tag{2}$$

The corresponding optimal policy can be extracted from the value function:

$$\pi^*(\mathbf{s}) = \operatorname*{argmax}_{\mathbf{a}\in A}\left[R(\mathbf{s},\mathbf{a}) + \beta \sum_{\mathbf{s}'\in\mathcal{S}} T_{\mathbf{a}}\left(\mathbf{s}'\middle|\mathbf{s}\right)V^*\left(\mathbf{s}'\right)\right]. \tag{3}$$

Given an implicit optimal policy π^* in the form of its optimal value function $V^*(\cdot)$, we measure the superiority of an action by a *Q-function*: $\mathcal{S}\times\mathcal{A}\rightarrow\mathbb{R}$.

Definition 3. The Q^**-value* of a state–action pair $(\mathbf{s},\mathbf{a})$ is the value of state $\mathbf{s}$, if an immediate action $\mathbf{a}$ is performed, followed by π^* afterwards. More concretely,

$$Q^*(\mathbf{s},\mathbf{a}) = R(\mathbf{s},\mathbf{a}) + \beta \sum_{\mathbf{s}'\in\mathcal{S}} T_{\mathbf{a}}\left(\mathbf{s}'\middle|\mathbf{s}\right)V^*\left(\mathbf{s}'\right). \tag{4}$$

Therefore, the optimal value function can be expressed by:

$$V^*(\mathbf{s}) = \max_{\mathbf{a}\in A} Q^*(\mathbf{s},\mathbf{a}). \tag{5}$$

2.2. Solving an MDP

Many optimal MDP algorithms are based on dynamic programming. A simple, yet powerful such algorithm, *value iteration* [2], is also the basis of many heuristic search and approximate algorithms. It first initializes the value function arbitrarily. Then, the values are updated iteratively using an operator called the *Bellman backup* to create successively better approximations for the value of each state per iteration. The *Bellman residual* of a state is the absolute difference of a state value before and after a Bellman backup. Value iteration stops when the value function converges. In implementation, it is typically signaled by when the *Bellman error*, the largest Bellman residual of all states, becomes less than some pre-defined threshold.

A simple way to approximate solutions to large MDPs is to use Monte Carlo (MC) simulation algorithms. Such algorithms repeatedly perform simulation trials originating from the initial state. During each simulation, actions are chosen based on some heuristic function h, and the resulting states are chosen stochastically based on the transition probabilities of the action. Different heuristics can be used, depending on the desired tradeoff between exploration and exploitation. For example, a heuristic that purely exploits knowledge of the value function would always pick a greedy action that maximizes the value function. On the other hand, a heuristic that solely explores would always choose a random action. One simulation trial terminates when a terminal state[7] is encountered. At termination, the Q-values of all visited state–action pairs are updated in reverse order.

Upper Confidence Bounds Applied on Trees (UCT) [29] is an MC algorithm, whose power has been demonstrated by its application to several challenging problems like Go [18] and Real-time Strategy Games [1]. UCT uses a heuristic that considers both actions that have already been proven valuable and under-explored branches, in case better policies can be found. It remembers the total number of times a state $\mathbf{s}$ has been visited, $n_{\mathbf{s}}$, and the number of times action $\mathbf{a}$ is picked when $\mathbf{s}$ is visited, $n_{(\mathbf{s},\mathbf{a})}$. Its heuristic function, $h_{UCT}(\mathbf{s},\mathbf{a})$, is defined as follows:

$$h_{UCT}(\mathbf{s},\mathbf{a}) = Q(\mathbf{s},\mathbf{a}) + \kappa\sqrt{\frac{2\ln n_{\mathbf{s}}}{n_{(\mathbf{s},\mathbf{a})}}}. \tag{6}$$

The second term increases the value of actions that are visited less frequently. The *exploration parameter* κ is used to balance between exploration and exploitation. Theoretical error bounds given a sufficient number of trials have been proven [29].

2.3. Partially-Observable Markov Decision Processes

Partially-Observable MDPs (POMDPs) [25] provide a more flexible framework with which to model decision-making under uncertainty by relaxing the assumption that an agent has complete knowledge of the world and can only make noisy observations about its current state. This generality can model many single-agent, real-world problems, since perfect information about the world is rarely available to an agent.

Definition 4. A POMDP is a six-tuple $\langle\mathcal{S},\mathcal{A},\mathcal{O},\mathcal{T},\mathcal{P},\mathcal{R}\rangle$, where

- $\mathcal{S}$ is a finite set of discrete states.
- $\mathcal{A}$ is a finite set of all actions.

[7] A terminal state is a state whose only positive-probability transition is to itself.

- $\mathcal{O}$ is a finite set of observations.
- $\mathcal{T}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1]$ is the transition function describing the probability that taking an action in a given state will result in another state.
- $\mathcal{P}: \mathcal{S} \times \mathcal{O} \to [0,1]$ is the observation function describing the probability that taking an action in a given state will result in an observation.
- $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward for taking an action in a state.

We see that a POMDP extends an MDP by adding a finite set of observations $\mathcal{O}$ and a corresponding observation model $\mathcal{P}$. Since the agent is unable to directly observe the world's current state, it maintains a probability distribution over states, called a *belief state* $\mathbf{b}$, reflecting its estimate of their likelihood:

$$\mathbf{b}(\mathbf{s}) \in [0,1], \quad \sum_{\mathbf{s} \in \mathcal{S}} \mathbf{b}(\mathbf{s}) = 1, \tag{7}$$

where $\mathbf{b}(\mathbf{s})$ is the probability that the current state is $\mathbf{s}$, inferred from the previous belief state, the most recent action, and the resulting observation.

A POMDP can be cast as an MDP by letting the state space of the MDP be the space of all possible belief states. Since the transition between two belief states is completely determined by the action and resulting observation, the resulting model is a fully-observable MDP – but over a vastly larger space. Even when a POMDP is defined over only two world states, $\mathbf{s}_1$ and $\mathbf{s}_2$, the space of belief states is infinite, since a belief state can be an arbitrary combination of $\mathbf{b}(\mathbf{s}_1) = p$ and $\mathbf{b}(\mathbf{s}_2) = 1 - p$ where p can be any real number in $[0,1]$.

Solving a POMDP is a hard problem. For the simplest case where the set of $\mathcal{A}$, $\mathcal{S}$ and $\mathcal{O}$ are finite, Sondik [59] proves that the optimal value function for a finite-horizon problem is piecewise linear and convex (PWLC). Therefore, value iteration-type algorithms converge on these problems. However, the number of reachable belief states grows exponentially in the number of observations $|\mathcal{O}|$. The complexity of one update iteration is exponential in the total number of observations [25]. For an infinite-horizon POMDP one needs to perform an infinite number of iterations in the worst case, as the total number of reachable belief states can be infinite. Therefore, the problem is undecidable [38] in the worst case. Porta et al. [45] later proved that finite-horizon continuous state POMDPs are PWLC.

Researchers seek various approximation methods to solve POMDPs efficiently. Point-based value iteration [61,44,54] is a very successful approach. The basic idea is to approximate the value function of the belief space, $\mathcal{B}$, by only estimating the values of a fixed, sampled subset, $\bar{\mathcal{B}} \subseteq \mathcal{B}$. For all the other belief states, the values are approximately estimated according to the values of the sampled space, *e.g.*, inferred from integration [45] or a mixture of Gaussians representation [8]. For a tractable, finite-horizon problem, the algorithm iteratively computes an approximately optimal t-horizon value function. The PWLC properties of a discrete POMDP guarantee that the error of the value function computed by the point-based algorithms is bounded [45].

2.4. Planning algorithms

We now present three algorithms that we use in this paper for solving POMDPs.

2.4.1. Limited lookahead

Our first decision-making algorithm is an l-step lookahead. The goal is to evaluate all sequences of up to l decisions and find the best sequence. To determine the value of a given sequence of l decisions, we calculate the expected utility from stopping at the lth decision based on the agent's current belief. Based on these expected utility estimations, the agent picks the sequence with the best utility, executes the first action of that sequence, and then repeats the process until a terminal state is reached.

2.4.2. A discretized POMDP algorithm

We also try a discretization-based POMDP algorithm, which we call ADBS, short for approximation and discretization of belief space. The method approximates the belief distribution by the values of its mean and standard deviation. Similar techniques have been used in the robot navigation domain [50]. ADBS discretizes the range of each variable into small, equal-sized intervals. With discretization and known bounds (we show later that we can bound these variables for our application), the space of belief states is finite. ADBS performs a reachability search from the initial state to build an MDP model of the approximated belief space. It then optimally solves the discretized MDP by value iteration.

2.4.3. A UCT variant

For our third algorithm, we use a variant of UCT applied to a space whose states are a simplified history of actions and observations, which we call *condensed histories* and define in Section 4.4. Although using condensed histories has the unfortunate property that different histories may represent the same belief state, this approximation technique excels in several ways. It saves computation, since we do not need to maintain posterior beliefs. Maintaining beliefs is tricky for

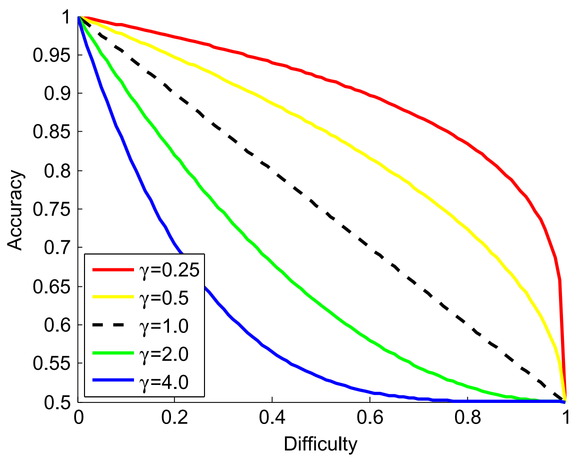


Fig. 2. Worker accuracies given the intrinsic difficulty under various error parameters γ. With a fixed difficulty, the greater a worker's γ value, the more likely that worker makes a mistake.

our problems, since the state space is continuous, and a belief over continuous state space is non-trivial to represent and update. For the same reason, implementing history-based POMDPs is much easier. Finally, it makes execution faster too, since one can directly map the history to the action. Monte Carlo simulation-based POMDP algorithms have been proved to be effective on very large problems [56].

3. Decision-theoretic optimization of a binary classification workflow

We now begin the study of decision-theoretic control of crowdsourcing. We first consider one of the simplest of workflows, the binary classification, or the binary ballot job. These workflows present workers with a question and ask them to pick the best answer from two choices. For instance, a workflow might show workers a picture and ask them if there is a human face in the picture. Then some method of aggregation is used, like a majority vote, to account for worker variability. Such workflows might be used to collect training data for a computer vision algorithm.

The agent's control problem is defined as follows. As input the agent is given the task, and the agent is asked to return the correct answer. In pursuit of this goal, the agent is allowed to create jobs on a crowdsourcing platform. We model the agent's decision problem as a POMDP.

To do this, we must first define a generative model for worker responses.

3.1. Probabilistic model for a binary classification response

We assume workers are diligent, so they answer all ballots to the best of their abilities. In other words, we assume workers are not adversarial. Additionally, we also assume that workers do not collaborate.

We define a worker's accuracy to be

$$a(d,\gamma)=\frac{1}{2}\bigl(1+(1-d)^{\gamma}\bigr) \tag{8}$$

where $d\in[0,1]$ is the *intrinsic difficulty* of the task, which we will motivate shortly, and $\gamma_x\in[0,\infty]$ is the *error parameter* of the worker.

As a worker's error parameter and/or the workflow's difficulty increases, a approaches 1/2, suggesting that worker is randomly guessing. On the other hand, as the parameters decrease, a approaches 1, when the worker always produces the correct answer. As γ decreases, the accuracy curve becomes more concave, and thus the expected accuracy increases for a fixed d. Fig. 2 is an illustration of accuracy.

The answer b_w that worker w with error parameter γ_w provides is governed by the following equations:

$$P(b_w=v|d)=a(d,\gamma_w), \tag{9}$$

$$P(b_w\neq v|d)=1-a(d,\gamma_w), \tag{10}$$

Fig. 7 illustrates the plate notation for our generative model, which encodes a Bayes Net for responses made by W workers on T tasks. The correct answer, v, the difficulty parameter, d, and the error parameter, γ, influence the final answer, b, that a worker provides, which is the only observed variable.

We note that given our assumption that workers do not collaborate, one might believe that the workers' responses b_w are independent of each other given the true answer v. However, they are not because of the following subtlety. Even though the different workers are not collaborating, a mistake by one worker changes the error probability of others, because a mistake gives evidence that the question may be intrinsically hard and hence, difficult for others to get it right as well. Thus, as shown in the graphical model, we make the assumption that the worker responses are independent of each other only after we are additionally given the difficulty d.

With this generative model, we can define our POMDP.

3.2. The POMDP

Definition 5. The POMDP for our simple binary classification workflow is a six-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \mathcal{O}, P \rangle$, where

- $\mathcal{S} = \{(d, v) \mid d \in [0,1], v \in \{0,1\}\}$ d is the difficulty of the task and v is the true answer.
- $\mathcal{A} = \{\textit{create another job}, \textit{submit true}, \textit{submit false}\}$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is specified below.
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0,1] = ((d, v), a, (d, v)) \mapsto 1$. All other probabilities are 0.
- $\mathcal{O} = \{\textit{true}, \textit{false}\}$ is a Boolean response by a worker.
- $P : \mathcal{S} \times \mathcal{O} \to [0,1]$ is defined by our generative model.

The reward function maintains the value of submitting a correct answer and the penalty for submitting an incorrect answer. Additionally, it maintains a cost that TurKontrol$_0$ incurs when it creates a job. We can modify the reward function to match our desired budgets and accuracies.

We note that this POMDP is a purely sensing-POMDP. None of the actions changes the state of the POMDP.

In many crowdsourcing platforms, such as Mechanical Turk, we cannot preselect the workers for a job. However, in order to specify our observation probabilities, which are defined by our generative model, we need access to future workers' parameters. To simplify the computation, our POMDP calculates observation probabilities using an average γ. In other words, it assumes that every future worker is an average worker. Formally, every future worker has an error parameter equal to $\overline{\gamma} = \frac{1}{W} \sum_w \gamma_w$ where W is the number of known workers.

However, our agent can keep around knowledge about worker accuracy in order to use accurate estimates of γ when updating its belief state. In particular, if someone answers a question correctly (according to the agent's belief), then she is a good worker (and her γ_w should decrease) and if someone made an error in a question her γ_w 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 answers a question of difficulty d correctly then $\gamma_w \leftarrow \gamma_x - d\delta$.
2. If a worker makes an error when answering a question then $\gamma_w \leftarrow \gamma_w + (1-d)\delta$.

We use δ to represent the learning rate, which we can slowly reduce over time so that the accuracy of a worker approaches an asymptote.

More sophisticately, one may update the parameters using Bayesian updates, in the same flavor as discussed in Section 3.4.

3.3. Simulated experiments

We evaluate our model in simulation against an agent that uses a majority-vote strategy to assess the viability of using a POMDP to control such a simple workflow for crowdsourcing.

The POMDP must manage a belief state over the cross product of the Boolean answer and one continuous variable – the task difficulty. Since solving a POMDP with a continuous state space is challenging, we discretize difficulty into eleven possible values, leading to a (world) state space of size $2 \times 11 = 22$. To solve POMDPs, we run the ZMDP package[8] for 300 s using the default Focused Real-Time Dynamic Programming search strategy [57].

On each run, the simulator draws difficulty uniformly. We fix the reward of returning the correct answer to 0, and vary the reward (penalty) of returning an incorrect answer between the following values: -10, -100, and -1000. We set the cost of creating a job for a (simulated) worker to -1. We use a discount factor of 0.9999 in the POMDP so that the POMDP solver converges quickly. We ensure that majority vote uses a number of ballots that is at least as many and within two of the average number of ballots that TurKontrol$_0$ uses. Difficulties are drawn uniformly randomly and workers' γ are drawn from Normal $(1.0, 0.2)$. For each setting of reward values we run 1000 simulations and report mean net utilities (Fig. 3) and the percent reduction in error that TurKontrol$_0$ achieves when comparing its accuracy to that of majority voting (Fig. 4).

We see that our model works well when the average difficulty of the task is not too difficult and not too easy and the variance is wide. To gain further insight, we investigate the effectiveness of our model at varying settings of task difficulty.

[8] http://www.cs.cmu.edu/~trey/zmdp/.

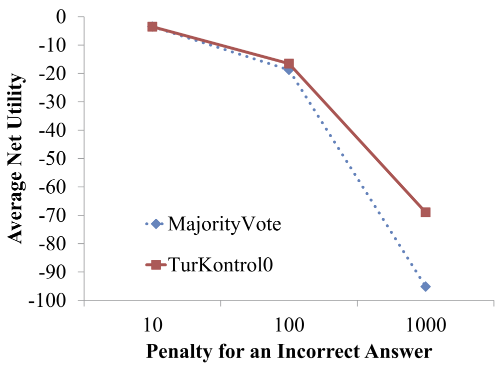


Fig. 3. In simulation, as the importance of answer correctness increases, TurKontrol$_0$ outperforms an agent using a majority-vote strategy by an ever-increasing margin.

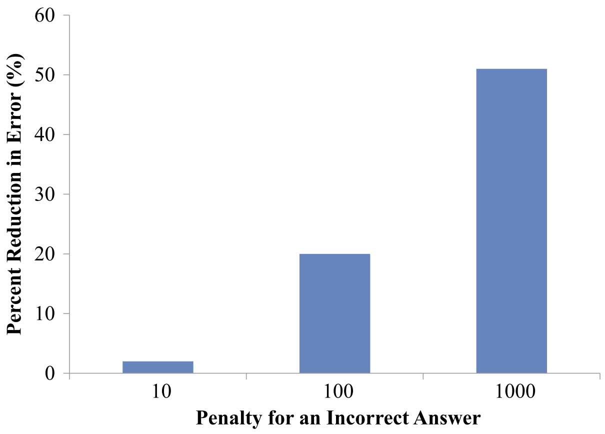


Fig. 4. In simulation, as the importance of answer correctness increases, TurKontrol$_0$ increases percent reduction in error that it achieves when comparing its accuracy to that of majority voting.

We run 1000 simulations at nine equally spaced difficulty settings, starting from $d = 0.1$. Workers' γ are drawn from Normal $(1.0, 0.2)$, and we set the reward for an incorrect answer to be -1000. However, we assume that TurKontrol$_0$ has no knowledge of worker γ, always receives observations from new workers, and only has access to the average γ. To further place our agent in the most unfavorable testing conditions as possible, we compare TurKontrol$_0$ against an agent (MV) that uses the following adaptive majority-vote strategy. For every difficulty setting, we determine, on average, how many ballots TurKontrol$_0$ uses. Then we set the majority-vote strategy to use at least as many ballots, and at most 1 more.

Fig. 5 shows the percent reduction in error that TurKontrol$_0$ achieves when comparing its accuracy to that of MV. Since TurKontrol$_0$ always achieves equal or better accuracy than MV, and MV always uses equal or more ballots, TurKontrol$_0$ also always achieves equal or better net utility than MV. Interestingly, the figure shows an illuminating trend. TurKontrol$_0$ only achieves slightly better results than MV at the extremes of the difficulty spectrum, but shows larger gains in the middle difficulty settings. Such a result intuitively makes sense because of the following reasons. When the problems are easy, there is no need to do anything dynamic. Similarly, when the problems are extremely difficult, there is nothing TurKontrol$_0$ can do without more knowledge about the workers. However, when the problem difficulty is in the middle range, sometimes the workers will agree quickly and sometimes they will not, and the ability of our agent to make dynamic decisions contributes to both higher accuracy and higher net utility.

3.4. Learning the model

Extensive simulation results in the previous section show the usefulness of decision-theoretic techniques for a simple binary classification workflow. This section addresses learning the model from real Mechanical Turk data [13]. We will use the following task. Given an image and a pair of descriptions, we ask workers to select the better description. This learning problem is challenging due to the large number of parameters and the sparse, noisy training data.

We begin by defining the supervised learning problem. Fig. 6 presents our generative model of ballot jobs; we can observe the ballots, the true answer, and the difficulty of the task. We seek to learn the error parameters $\boldsymbol{\gamma}$ where γ_w is the parameter for worker w. To generate training data for our task we select T images and corresponding pairs of descriptions

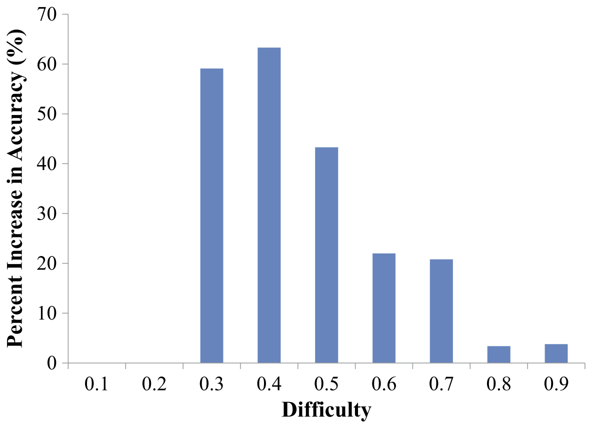


Fig. 5. The percent reduction in error that TurKontrol$_0$ achieves when comparing its accuracy to that of MV.

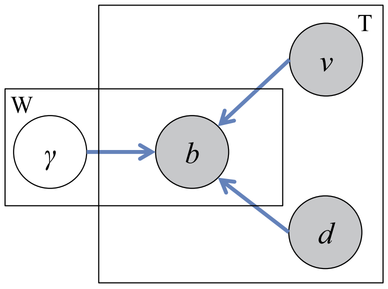


Fig. 6. A plate model of ballot jobs under a supervised learning setting. b represents the ballot outcome, γ is a worker's individual error parameter, d is the difficulty of the job and v is the truth value of the job. Shaded nodes represent observed variables.

and post W copies of a ballot job for each task. We use $b_{i,w}$ to denote worker w's ballot on the ith question. Let v_i denote whether the first artifact of the ith pair is better than the second, and d_i denote the difficulty of answering such a question. The ballot answer of each worker depends on her error parameter, as well as the difficulty of the job, d, and its real truth value, v.

For our learning problem, we collect values of v and d for the T ballot questions from the consensus of three human experts and treat these values as observed. In our experiments we assume γ are drawn from a uniform prior, though our model can incorporate more informed priors.[9] We use the standard maximum a posteriori approach to estimate the γ parameters:

$$P(\boldsymbol{\gamma}|\mathbf{b},\mathbf{w},\mathbf{d}) \propto P(\boldsymbol{\gamma})P(\mathbf{b}|\boldsymbol{\gamma},\mathbf{w},\mathbf{d}). \tag{11}$$

Under the uniform prior and conditional independence of different workers given difficulty and truth value of the task, Eq. (11) can be simplified to:

$$\begin{aligned} P(\boldsymbol{\gamma}|\mathbf{b},\mathbf{w},\mathbf{d}) &\propto P(\mathbf{b}|\boldsymbol{\gamma},\mathbf{w},\mathbf{d}) \\ &= \prod_{i=1}^{T}\prod_{w=1}^{W} P(b_{i,w}|\gamma_w,d_i,v_i). \end{aligned} \tag{12}$$

Taking the log, the maximum a posteriori problem is transformed to the following optimization problem:

$$\begin{aligned} &\text{Constants:} && d_1,\ldots,d_T,v_1,\ldots,v_T,b_{11},\ldots,b_{T,W} \\ &\text{Variables:} && \gamma_1,\ldots,\gamma_W \\ &\text{Maximize:} && \sum_{i=1}^{T}\sum_{w=1}^{W}\log\bigl[P(b_{i,w}|\gamma_w,d_i,v_i)\bigr] \\ &\text{Subject to:} && \emptyset \end{aligned}$$

[9] We also tried priors that penalized extreme values but that did not help in our experiments.

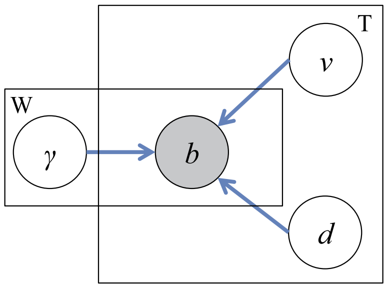


Fig. 7. A plate model of ballot jobs under an unsupervised learning setting. b represents the ballot outcome, γ is the worker's individual error parameter, d is the difficulty of the job, and w is the truth value of the job. Shaded nodes represent observed variables.

We also try an unsupervised learning algorithm. The plate notation is shown in Fig. 7. We don't assume knowledge of correct answers or the difficulties. Instead, we adopt an EM-style algorithm based on Whitehill et al.'s learning mechanism [68]. We initialize the difficulty values and error parameters with the corresponding average values learned from the supervised learning algorithm. We find applying prior distributions on the parameters does not help, so we do not use a prior. During the expectation step, we compute the probability of the true answers given the workers' answers, and the current values of the difficulty and error parameters. During the maximization step, we update the difficulty and error parameters based on the likelihood function (Eq. (12)).

3.4.1. Experiments on ballot model

We evaluate the effectiveness of our learning procedures against each other and against a majority-vote baseline on the image description task. We select 20 pairs of descriptions ($T = 20$) and collect sets of ballots from 50 workers. 5 spammers were detected manually and dropped from learning ($W = 45$). We spend \$4.50 on the supervised learning process. We solve the optimization problem using the NLopt package[10] and implement the unsupervised learning algorithm based on Whitehill's framework [68].

We run a five-fold cross-validation experiment. We take 4/5th of the image pairs and learn error parameters over them, and then use these parameters to estimate the true ballot answer for the images in the fifth fold. Our supervised learning algorithm obtains an accuracy of 80.01%, our unsupervised learning algorithm obtains an accuracy of 80.0%, and a majority-voting baseline obtains an accuracy of 80%. We investigate these similar accuracies and see that the four ballots frequently missed by the models are those in which the mass opinion differs from our expert labels.

We also compare the confidence, or the degree of belief in the correctness of an answer, for two approaches. For the majority-vote baseline, we calculate confidence by dividing the number of votes for the inferred correct answer by the total number of votes. For our supervised learning approach, we use the average posterior probability of the inferred answer. The average confidence values derived from the supervised learning approach is much higher than the majority vote (82.2% against 63.6%). Thus, even though the two approaches achieve the same accuracy on all 45 votes, our ballot model has superior belief in its answer.

Although the confidence values are different, the ballot models (learned from both supervised and unsupervised learning) seem to offer no distinct advantage over the simple majority-voting baseline given a large number of votes. In hindsight, this result is not surprising, since we are using a large number of workers. In other work, researchers have shown that a simple average of a large number of non-experts often beats even the expert opinion [58].

However, one will rarely have the resources to use 45 voters per question, so we consider the effect of varying the number of available voters. For each image pair, we randomly sample, without replacement, 50 000 sets of 3 to 11 ballots and compute the average accuracies of the three approaches. Fig. 8 shows that using our model (learned from both the supervised and the unsupervised learning algorithms) consistently outperforms the majority-vote baseline. Furthermore, applying the supervised learning algorithm consistently achieves higher accuracy than applying the unsupervised learning algorithm, which shows the usefulness of using expert labeling. The unsupervised learning algorithm gradually catches up as the number of votes increases, which distinguishes itself from majority voting. In contrast, the model learned from supervised learning always outperforms majority voting by a wide margin. With just 11 votes, it is able to achieve an accuracy of 79.3%, which is very close to the accuracy achieved using all 45 votes. Also, the supervised ballot model with only 5 votes achieves an accuracy similar to the one achieved by a majority vote with 11 votes. Our ballot model significantly reduces the number of votes and thus the amount of money needed for a given desired accuracy. Since the unsupervised learning algorithm does not require any labeled data, it can be useful when data labeling is expensive.

[10] http://ab-initio.mit.edu/wiki/index.php/NLopt.

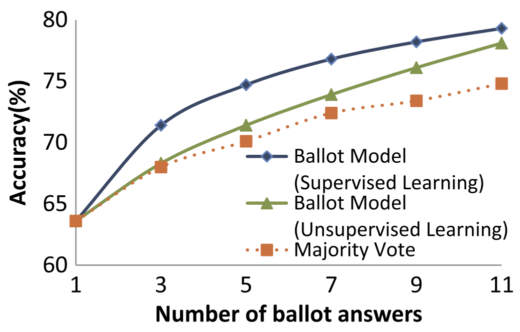


Fig. 8. Accuracies of using our ballot model (by applying both supervised and unsupervised learning) and majority vote on random voting sets of different sizes, averaged over 50000 random sets of each size. The models generated by the two learning algorithms both achieve higher accuracy than the majority vote. Using supervised learning, our ballot model achieves significantly higher accuracy than the majority vote ($p < 0.01$).

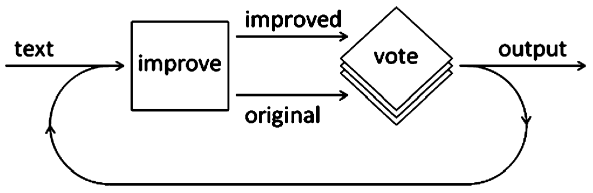


Fig. 9. Flow-chart for the iterative text improvement task, reprinted from [37].

4. Decision-theoretic optimization of an iterative improvement workflow

4.1. Motivation example

We now move to the decision-theoretic control of the iterative improvement workflow [12], introduced by Little et al. [37]. This workflow is depicted in Fig. 9. While the ideas in our paper are applicable to many complex workflows, we choose to apply them to this one because it is representative of a number of flows in commercial use today; at the same time, it is moderately complex making it ideal for a first investigation.

Iterative text improvement works as follows. There is an initial job, which presents the worker with an image and requests an English description of the image's contents. A subsequent iterative process consists of an *improvement job* and multiple *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 (see Fig. 22). Next, $n \geqslant 1$ ballot jobs are posted ("Which text best describes the picture?"). See Figs. 23, 24, 25, and 26 for our user interface design. 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 does not have built-in methods for monitoring the accuracy of workers; nor does it automatically determine the ideal number of voters or estimate the appropriate number of iterations before returns diminish.

4.2. Problem overview

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. In our initial model, we assume that requesters will express their utility as a function U from quality to dollars. To accomplish the task, the agent is allowed to post an improvement job or a ballot job.

To fully specify the problem, we must define what "quality" means. 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 an artifact is high quality if it is difficult to improve. Therefore, we define the quality of an artifact as follows. Let the *quality* be $q \in [0, 1]$. An artifact with quality q means an average dedicated worker has probability $1 - q$ of improving the artifact.

The agent never exactly knows the quality of an artifact. At best, it can estimate q based on domain dynamics and observations (like ballot results). Therefore, we can model the agent's control problem as a POMDP. Since quality is a real number, the state space of the POMDP is continuous [8].

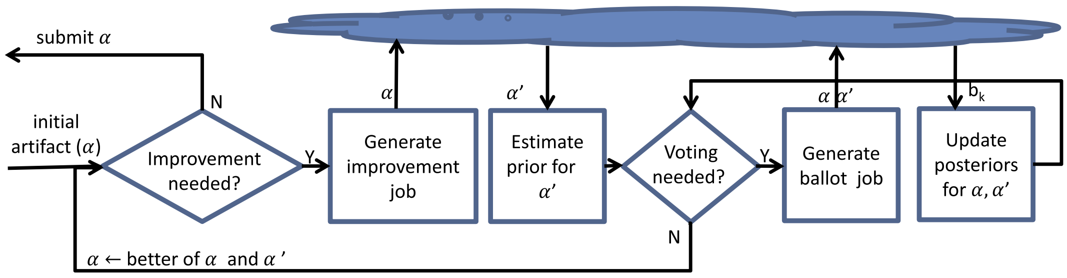

submit α
initial artifact (α)
Improvement needed?
N
Y
α
Generate improvement job
α'
Estimate prior for α'
Voting needed?
Y
N
α α'
Generate ballot job
b_k
Update posteriors for α, α'
$\alpha \leftarrow$ better of α and α'

Fig. 10. Computations needed by TurKontrol$_0$ for control of an iterative-improvement workflow.

4.3. The POMDP

Definition 6. The POMDP for the iterative improvement workflow is a six-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, P \rangle$, where

- $\mathcal{S} = \{(q, q') | q, q' \in [0, 1]\}$.
- $\mathcal{A} = \{$*create a ballot job*, *create an improvement job*, *submit best artifact*$\}$.
- $\mathcal{R}((q, q'), submit)$ is the reward received from submitting an artifact with quality $\max\{q, q'\}$.
- $\mathcal{R}((q, q'), create)$ is the cost of creating a job.
- $\mathcal{T}$: defined below.
- $\mathcal{O} = \{true, false\}$ is a Boolean answer received for a ballot question.
- P: defined below.

4.3.1. The transition function

We first note that the underlying state of the POMDP only changes when the agent requests an improvement. The qualities of the artifacts do not change when the agent requests votes. Votes only change the belief state of the agent.

Suppose we currently have artifacts α, α'', with unknown qualities q and q''. Therefore, the current state is (q, q''). Suppose without loss of generality that the agent posts an improvement job for α, and a worker x 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 q' also depends on x. In order to determine the next state, we must compute $P(q'|q, x)$, the conditional distribution of q' when worker x improves an artifact of quality q. We describe how to learn this conditional distribution later. The current state (q, q'') transitions to the new state (q, q') with probability $P(q'|q, x)$. However, since the agent does not know a priori the worker who will submit the improvement, it must assume, for the purpose of lookahead, that the worker has an average ability.

4.3.2. The observation function

Our observation function is only relevant when the agent requests ballot jobs. If the current (hidden) state is (q, q') with corresponding artifacts α and α', the agent can either observe a vote for α or α'. The observation function is defined by $P(o|q, q')$. Since a ballot job is simply an instantiation of a binary classification task, we can use our already defined generative model by defining the difficulty of the task.

To make this definition, we observe that the difficulty of the ballot job depends on the relative closeness of the two qualities of the underlying state. If the artifacts are of similar quality, it is difficult to judge whether one is better or not. Thus, we define the relationship between the difficulty and qualities as

$$d(q, q') = 1 - |q - q'|^{\mathcal{M}}, \tag{13}$$

where $\mathcal{M}$ is the *difficulty constant*.

4.3.3. Discussion

Fig. 10 is a high-level flow describing 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 improvement attempt or vote gives us new information, 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.

Our POMDP 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.

We note that with this general POMDP model, the belief in the quality of the previous artifact (posterior of α) can change based on ballots comparing it with the new artifact. Why should this be the case? We make the following 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 not easily improvable. Under such evidence we should increase our quality estimation of α. Similarly, if all voting workers unanimously thought that α' is much better than α, then α' likely incorporates significant improvements over α and the qualities should reflect such knowledge.

4.3.4. Updating difficulty and worker accuracy

To update its knowledge about the quality of each worker after each voting phase, the agent first estimates the expected difficulty of the voting using its estimates of quality as follows:

$$d^* = \int_0^1 \int_0^1 d(q, q') P(q) P(q')\, dq\, dq' = \int_0^1 \int_0^1 \left(1 - |q - q'|^{\mathcal{M}}\right) P(q) P(q')\, dq\, dq'. \tag{14}$$

Then, it uses d^* and what it believes to be the correct answer to update worker quality records as we discussed in Section 3.2.

4.3.5. Implementation

For efficient storage and computation TurKontrol$_0$ could employ the piecewise constant/piecewise linear value function representations or use particle filters. Although approximate, both these techniques are very popular in the literature for efficiently maintaining continuous distributions [39,17] and can provide arbitrarily close approximations. Because some of our equations require double integrals and can be time-consuming (*e.g.*, Eq. (14)) these compact representations will help in overall efficiency of the implementation.

Our piecewise constant function over a distribution density function, $P(q)$, divides the domain into fixed number of intervals and uses the average value $\frac{1}{u-l}\int_l^u f(q)\,dq$ as the value for each interval $[l, u]$. In implementation, a continuous function is represented by an array of values, where each value equals the constant function value of its designated interval.

4.4. Simulations

This section aims to empirically answer the following questions about the POMDP-solving algorithms we introduced in Section 2.4: (1) For a simple lookahead approach, how deep should an agent's lookahead be to best tradeoff between computation time and utility? (2) How well does the ABDS algorithm perform and scale? (3) What is the error of the UCT algorithm using a history approximation? (4) How well does the UCT algorithm perform? (5) What is the best planner for TurKontrol$_0$? (6) Does TurKontrol$_0$ make better decisions compared to a non-adaptive workflow? (7) Can our planner outperform an agent following a well-informed, fixed policy? (8) How well does our planner handle workers that are less effective at completing ballot jobs than improvement jobs?

4.4.1. Experimental setup

We define reward for submitting an artifact of quality q to be the convex function $R(q) = 1000\frac{e^q-1}{e-1}$ with $R(0) = 0$ and $R(1) = 1000$. 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.

We model results of an improvement task in a manner akin to ballot tasks. We know that the higher the quality of an artifact, the less likely that artifact can be improved. Suppose the quality of the current artifact is q, we define the conditional distribution $P(q'|q, x)$ to be $Beta(10\mu_{Q'|q,x}, 10(1 - \mu_{Q'|q,x}))$, with mean $\mu_{Q'|q,x}$, where

$$\mu_{Q'|q,x} = q + 0.5\left[(1 - q) \times \left(a_x(q) - 0.5\right) + q \times \left(a_x(q) - 1\right)\right]. \tag{15}$$

Thus, the resulting distribution of qualities is influenced by the worker's accuracy and the hardness of an improvement, indicated by the quality of the original artifact, q.

We fix the ratio of the costs of improvements and ballots to be $c_{imp}/c_b = 3$, 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_w, follow a bell shaped distribution with a fixed mean $\bar{\gamma}$. We also distinguish the accuracies of performing an improvement and answering a ballot by using one half of γ_w when worker w is answering a ballot, since answering a ballot is an easier task, and therefore a worker should have higher accuracy.[11]

4.4.2. The l-step lookahead algorithm

We first try to find the optimal depth for the l-step lookahead algorithm we presented earlier. When $l = 2$, such an algorithm considers the following set of action sequences $\{\langle stop\rangle, \langle ballot, stop\rangle, \langle improvement, stop\rangle, \langle ballot, ballot\rangle, \langle ballot, improvement\rangle, \langle improvement, ballot\rangle, \langle improvement, improvement\rangle\}$.

We run 10 000 simulation trials with average error coefficient $\bar{\gamma} = 1$ 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 the l-step lookahead algorithm. Fig. 11 shows

[11] For simplicity reasons, in our simulations only, we assume a worker has only one γ_w. We relax this assumption when we run real experiments on Mechanical Turk.

Table 1
Average utility of TurKontrol(2) and performance of the ADBS algorithms with different resolution (discrete interval lengths) on three sets of (improvement, ballot) costs: (30, 10), (3, 1), and (0.3, 0.1). TurKontrol(2) outperforms the ADBS algorithms in all settings.

Costs	TurKontrol(2)	ADBS			
		Interval length	0.02	0.01	0.005
		$\|\mathcal{S}\|$	3577	41 072	809 420
(30, 10)	264.906	$V^*(\mathbf{s}_0)$	253.951	253.951	253.951
(3, 1)	445.721	$V^*(\mathbf{s}_0)$	365.866	366.798	367.674
(0.3, 0.1)	466.820	$V^*(\mathbf{s}_0)$	382.066	385.698	387.366

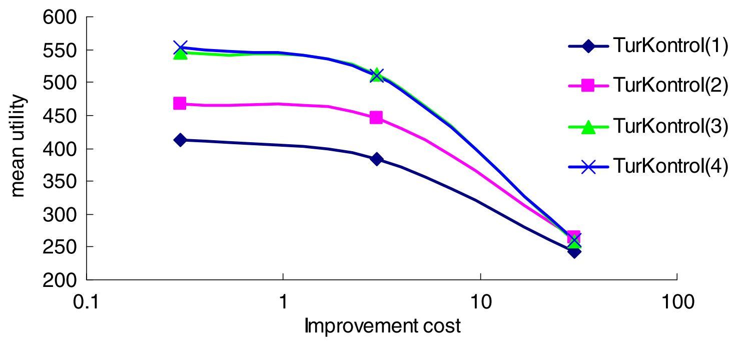


Fig. 11. Average utility of TurKontrol with various lookahead depths calculated using 10 000 simulation trials on three sets of (*improvement*, *ballot*) costs: (30, 10), (3, 1), and (0.3, 0.1). Longer lookahead produces better results, but 2-step lookahead is good enough when costs are relatively high: (30, 10).

the average utility, achieved by 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, 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's lookahead to depth 2 in subsequent experiments.

4.4.3. The ADBS algorithm

Next, we try our ADBS algorithm. The states of the MDP are four-tuples $\langle\mu, \sigma, \mu', \sigma'\rangle$, an average and a standard deviation for each hidden quality. Since quality is a real number in [0, 1], we have that $\mu \in [0, 1]$ and $\sigma \in [0, 1]$, and we can finitely discretize these intervals.

We try different resolutions of discretization, *i.e.* various constant interval lengths, and run with average error coefficient $\overline{\gamma} = 1$ on three pairs of improvement and ballot costs – (30, 10), (3, 1), and (0.3, 0.1). Table 1 lists the average utility of TurKontrol(2) on 10 000 simulations (results incorporated from Fig. 11). For the ADBS algorithm, we report the size of the reachable belief states under each resolution, $|\mathcal{S}|$, and the value of the initial state, $V^*(\mathbf{s}_0)$, calculated by value iteration.

We find TurKontrol(2) outperforms ADBS in all settings. Moreover, the smaller the costs, the bigger the performance gap. The poor performance of ADBS is probably due to the errors generated during approximation and discretization. Also notice that with more refined discretization, the reachable state space grows very quickly (at approximately a rate of an order of magnitude per doubly refined interval), yet the optimal value of the initial state increases very slowly. This indicates that the error most likely comes from the approximation as opposed to the discretization. We also try an interval length of 0.0025, but the algorithm terminates prematurely when the reachability search runs out of memory – indicating the limited scalability of the ADBS algorithm. This experiment shows that a simple POMDP method does not work as well as the 2-step lookahead algorithm, yet we cannot yet draw the conclusion that POMDP algorithms do not work on this planning problem.

4.4.4. The UCT algorithm

Finally, we try the UCT variant. We first evaluate the amount of suboptimality introduced by the our condensed history approximation. An example of a condensed history is: (Step 1) An initial artifact α_1 is provided. (Step 2) The agent creates an improvement HIT and obtains α_2. (Step 3) The agent creates a ballot job, and receives a vote that α_2 is better than α_1. (Step 4) The agent submits α_2.

Since the worker identities are ignored, the belief state space size is drastically reduced.

In a new TurKontrol(2) version, we do everything the same as TurKontrol(2) except we regard every worker as an anonymous worker of the same, average accuracy, $\overline{\gamma}$. We denote this new variant TurKontrol(2, anonymous). Note that in

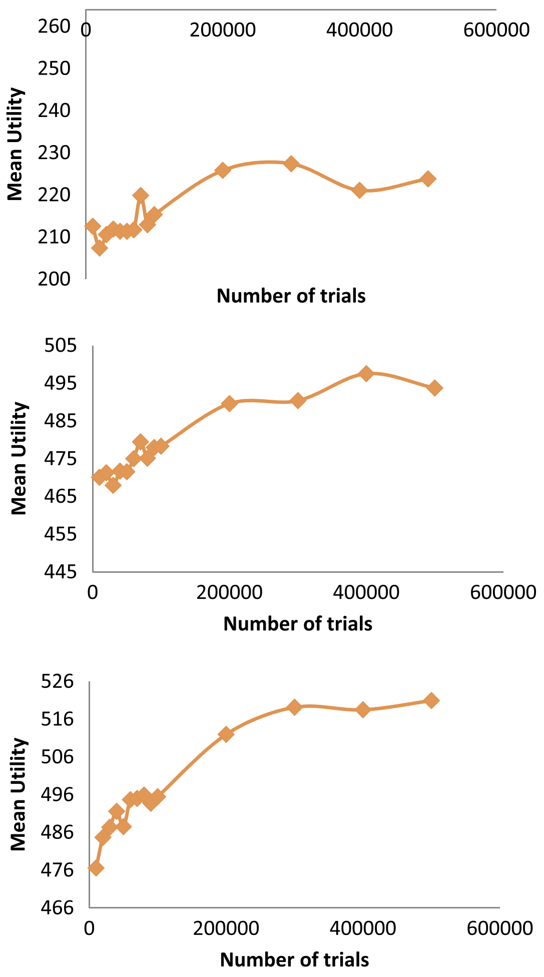


Fig. 12. Average utility of the UCT algorithm while varying the number of trials compared to average utility of 10 000 simulation trials of TurKontrol(2) (the horizontal lines, which do not represent the x-axes) with improvement and ballot costs $(30, 10)$ [top], $(3, 1)$ [middle], and $(0.3, 0.1)$ [bottom]. UCT achieves higher average utility when the costs are low.

this case, the policy is non-adaptive, as there is no uncertainty in workers' accuracies. We perform 10 000 simulations on the following two configurations[12]:

1. $\overline{\gamma} = 2$ and ballot accuracies are higher than improvement accuracies: TurKontrol(2) gets 4.5% more utility than TurKontrol(2, anonymous).
2. $\overline{\gamma} = 1$ and ballot accuracies are the same as improvement accuracies: TurKontrol(2) gets 3.7% more utility than TurKontrol(2, anonymous).

This shows using the average $\overline{\gamma}$ is not a bad approximation, because it only reduces the expected utility of TurKontrol(2) by less than 5% in both cases. We expect it to produce even better results if the approximation is more refined (*e.g.*, if we group the workers by accuracy).

In investigating the UCT algorithm, we use a fixed $\kappa = 50$ and a dynamically decreasing learning parameter $\theta = \frac{0.2 \times \log_2 10}{n_{(s,a)}}$. As an action has been visited more frequently, the effect of randomness decreases, so less weight should be given to the new observation. Fig. 12 plots the average utility of the UCT algorithm as a function of the number of trials completed (from 10 000 to 500 000). The horizontal lines are not x-axes, but represent the average utility of TurKontrol(2) (data as in Fig. 11). First notice that the performance of UCT improves with the increase in the number of completed trials. We also observe that UCT outperforms TurKontrol(2) after very few (10 000) trials on the two problems where the costs are small, but underperforms TurKontrol(2) on the problem where the costs are high. This behavior is probably because TurKontrol(2) already performs close to optimal on the high cost problem. We also find UCT consistently underperforms

[12] We intentionally choose these two cases because they are the most favorable cases for TurKontrol(2), as we will find in the following subsections.

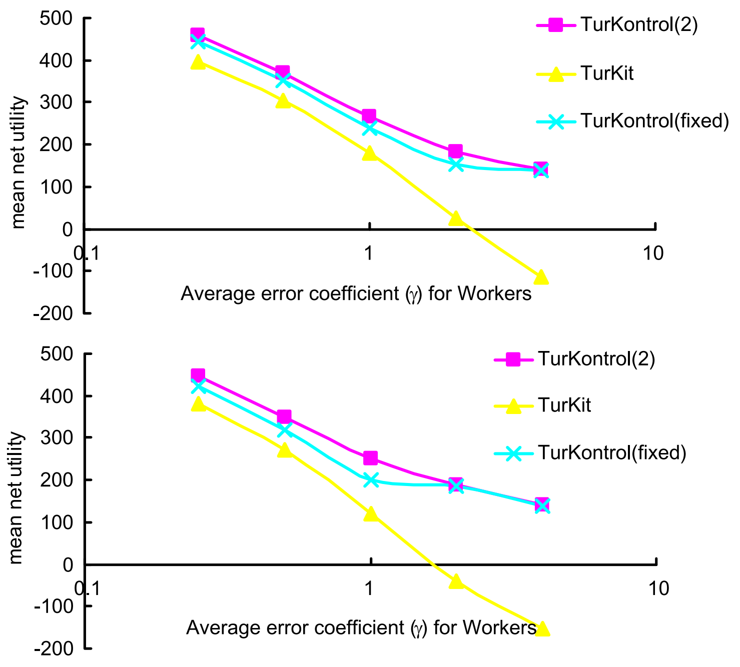


Fig. 13. Average utility of three control policies averaged over 10 000 simulation trials, varying mean error coefficient, $\overline{\gamma}$. Top – We set workers to have better accuracies in ballot jobs than improvement jobs. Bottom – We set workers to have equally accurate in both ballot and improvement jobs. TurKontrol(2) produces the best policy in every case.

TurKontrol(3) (from comparing against Fig. 11). This is because TurKontrol(3) computes close-to-optimal values, while UCT's performance is influenced by random noises during search trials. This experiment shows that UCT is useful for quickly finding a good suboptimal policy but has limited power in finding a close-to-optimal policy.

4.4.5. *Choosing the best planning algorithm*

From comparing the results of the three algorithms: the l-step lookahead, ADBS and UCT, we conclude that the l-step lookahead is the most efficient planning algorithm for the domain.

4.4.6. *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 non-adaptive policy from the literature [37]; it performs as many iterations as possible until its fixed allowance (400 in our experiment) is depleted and on each iteration it requests 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). Thus this represents non-adaptive policy whose parameters are tuned to costs and worker accuracies.

Fig. 13 (top) 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 with $p < 0.01$. We also note that the performance of TurKontrol(fixed) is very similar to that of TurKontrol(2), when workers are very inaccurate, $\overline{\gamma} = 4$. 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 confirms the differences are all statistically significant for $\overline{\gamma} < 4$. 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 $\overline{\gamma} = 2$, where TurKontrol(2) generates 19.7% more utility than TurKontrol(fixed).

4.4.7. *Robustness in the face of bad voters*

As a final study, we consider the sensitivity of the previous three policies to increasingly noisy voters. Specifically, we repeat the previous experiment using the same error coefficient, γ_w, for each worker's improvement *and ballot* behavior (Fig. 13 (bottom)). (We previously set the error coefficient for ballots to one half γ_w to model the fact that voting is easier.) Fig. 13 (bottom) has the same shape as that of Fig. 13 (top) but with lower overall utility. Once again, TurKontrol(2) continues to achieve the highest average utility across all settings. Interestingly, the utility gap between the two TurKontrol variants and TurKit is consistently bigger for all $\overline{\gamma}$ than in the previous experiment. In addition, when $\overline{\gamma} = 1$, TurKontrol(2)

Table 2
Qualities of 10 artifacts, collected through three methods: definition simulation, direct expert estimation and averaged worker estimation. The averaged worker estimation is the cheapest method that provides comparable results to the other two methods.

	α_1	α_2	α_3	α_4	α_5	α_6	α_7	α_8	α_9	α_{10}
Def. simulation	0.05	0.53	0.48	0.62	0.33	0.41	0.10	0.4	0.74	0.32
Avg worker est.	0.12	0.51	0.43	0.45	0.44	0.36	0.27	0.56	0.63	0.44
Expert est.	0.1	0.5	0.4	0.6	0.4	0.2	0.3	0.5	0.9	0.4

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 $\bar{\gamma} < 2$ and the differences between both TurKontrol variants and TurKit are significant at all settings.

4.5. Learning the improvement model

Extensive simulation results in the previous section show the usefulness of decision-theoretic techniques for iterative improvement workflows. This section addresses learning the model from real Mechanical Turk data [13]. We already explored the learning of our ballot model in Section 3.4, so now we learn our improvement model. To learn the effect of a worker trying to improve an artifact, we first need a method for determining the ground truth for the quality of an arbitrary artifact.

4.5.1. Estimating artifact quality gold standards

Since quality is a partially-observable statistical measure, we consider three ways to approximate it: simulating the definition, direct expert estimation, and averaged worker estimation.

Our first technique simply *simulates the definition*. We ask W workers to improve an artifact α and as before use multiple ballots, say l, to judge each improvement. We define the quality of α to be 1 minus the fraction of workers that are able to improve it. Unfortunately, this method requires $W + Wl$ jobs in order to estimate the quality of a single artifact; thus, it is both slow and expensive in practice. As an alternative, *direct expert estimation* is less complex. We teach a statistically-sophisticated computer scientist the definition of quality and ask her to estimate the quality to the nearest decile.[13] Our final method, *averaged worker estimation*, is similar, but averages the judgments from several Mechanical Turk workers via *scoring jobs*. These scoring jobs provide a definition of quality along with a few examples, mapped to a 0–10 integer scale; the workers are then asked to score several more artifacts. See Figs. 27, 28, and 29 for our user interface design.

We collect data on 10 images from the Web and use Mechanical Turk to generate multiple descriptions for each. We then select one description for each image, carefully ensuring that the chosen descriptions span a wide range of detail and language fluency. We also modified a description to obtain one that, we felt, was very hard to improve, thereby accounting for the high-quality region. When simulating the definition, we average over $W = 22$ workers.[14] We use a single expert for direct expert estimation and an average of 10 worker scores for averaged worker estimation. See Table 2.

Our hope, following the work of [58], is that averaged worker estimation, definitely the cheapest method, proves comparable to expert estimates and especially to the simulated definition. Indeed, we find that all three methods produce similar results. They agree on the two best and worst artifacts, and on average both expert and worker estimates are within 0.1 of the score produced by simulating the definition. We conclude that averaged worker estimation is equally effective and additionally easier and more economical (1 cent per scoring job), so we adopt this method to assess qualities in subsequent experiments.

4.5.2. Actually learning the model

Finally, we describe our approach for learning a model for the improvement phase. Our objective is to estimate $P(q'|q, w)$, the probability distribution that describes the quality q' of a new artifact, α', when worker w improves artifact α of quality q. Moreover, we also learn a prior, $P(q'|q)$, in order to model work by a previously unseen worker.

There are two main challenges in learning this model: first, these functions are over a two-dimensional continuous space, and second, the training data is scant and noisy. To alleviate these difficulties, we break the task into two learning steps: (1) learn a mean value for quality using regression, and (2) fit a conditional density function given the mean. We make the second learning task tractable by choosing parametric representations for these functions. Our full solution follows the following steps:

1. Generate an improvement job that contains T original artifacts $\alpha_1, \ldots, \alpha_T$.
2. Crowdsource W workers to improve each artifact to generate WT new artifacts.
3. Estimate the qualities q_i and $q'_{i,w}$ for all artifacts in the set (see previous section). q_i is the quality of α_i and $q'_{i,w}$ denotes the quality of the new artifact produced by worker w. This data is our training data.

[13] The consistency of this type of subjective rating has been carefully evaluated in the literature; see *e.g.* [11].

[14] We collected 24 sets of improvements, but two workers improved less than 3 artifacts, and their results were dropped from analysis.

4. Learn a worker-dependent distribution $P(q'|q, w)$ for every participating worker w.
5. Learn a worker-independent distribution $P(q'|q)$ to act as a prior for unseen workers.

We now describe the last two steps in detail: the learning algorithms. We first estimate the mean of worker w's improvement distribution, denoted by $\mu_{q'_w}(q)$.

We assume that $\mu_{q'_w}$ is a linear function of the quality of the original artifact.[15] By introducing $\mu_{Q'_w}$, we separate the variance in a worker's ability in improving all artifacts of the same quality from the variance in our training data, which is due to her starting out from artifacts of different qualities. To learn this we perform linear regression on the training data $(q_i, q'_{i,w})$. This yields $q'_w = a_w q + b_w$ as the line of regression with standard error e_w, which we truncate for values outside $[0, 1]$.

To model a worker's variance when improving artifacts with the same quality, we consider three parametric representations for $P(q'|q, w)$: Triangular, Beta, and Truncated Normal. While clearly making an approximation, restricting attention to these distributions significantly reduces the parameter space and makes our learning problem tractable. Note that we assume the mean of each of these distributions is given by the line of regression. We consider each distribution in turn.

Triangular. The triangular-shaped probability density function has two fixed vertices $(0, 0)$ and $(1, 0)$. We set the x-coordinate of the third vertex to $\mu_{q'_w}(q)$, yielding the following probability density function $f_{q'_w|q}(q'_w)$:

$$f_{Q'_w|q}(q'_w) = \begin{cases} \frac{2q'_w}{\mu_{q'_w}(q)} & \text{if } q'_w < \mu_{q'_w}(q), \\ \frac{2(1-q'_w)}{1-\mu_{q'_w}(q)} & \text{if } q'_w \geqslant \mu_{q'_w}(q). \end{cases} \tag{16}$$

Beta. We wish the Beta distribution's mean to be $\mu_{q'_w}$ and its standard deviation to be proportional to e_w. Therefore, we train a constant, c_1, using gradient descent that maximizes the log-likelihood of observing the training data for all workers.[16] resulting in the distribution $Beta(\frac{c_1}{e_w} \times \mu_{q'_w}(q), \frac{c_1}{e_w} \times (1 - \mu_{q'_w}(q)))$. The error e_w appears in the denominator because the two parameters for the Beta distribution are approximately inversely related to its standard deviation.

Truncated Normal. As before we set the mean to $\mu_{q'_w}$ and the standard deviation to be $c_2 \times e_w$ where c_2 is a constant, trained to maximize the log-likelihood of the training data. This yields the distribution $= \mathit{Truncated\ Normal}(\mu_{q'_w}(q), c_2^2 e_w^2)$ where the truncated interval is $[0, 1]$.

We use similar approaches to learn the worker-independent model $P(q'|q)$, except that training data is of the form $(q_i, \bar{q}'_i)$ where $\bar{q}'_i$ is the *average* improved quality for ith artifact, *i.e.*, the mean of $q'_{i,w}$ over all workers. Denote the standard deviation of this set as $\sigma_{q'_i|q_i}$. As before, we start with linear regression, $q' = aq + b$. The Triangular distribution is defined exactly as before. For the other two distributions, their standard deviations depend on $\sigma_{q'_i|q_i}$. We assume that the conditional standard deviation $\sigma_{q'|q}$ is quadratic in q, and then infer an unknown conditional standard deviation using the existing ones, by running a quadratic regression. As before, we use gradient descent to train constants for the Beta and Truncated Normal distributions.

We seek to determine which of the three distributions best models the data, and we employ leave-one-out cross validation. We set the number of original artifacts and number of workers to be ten each ($T = W = 10$), and spend \$16.50 for this data collection. The algorithm iteratively trains on nine training examples, *e.g.* $\{(q_i, \bar{q}'_i)\}$ for the worker-independent case, and measures the probability density of observing the tenth. We score a model by summing the ten log probability densities.

Our results show that Beta distribution with $c_1 = 3.76$ is the best conditional distribution for worker-dependent models. For the worker-independent model, with an intercept of 0.35 and slope of 0.34 from the linear regression, Truncated Normal with $c_2 = 1.00$ performs the best. We suspect this is the case because most workers have average performance and Truncated Normal has a thinner tail than the Beta. In all cases, the Triangular distribution performs worst. This is probably because Triangular assumes a linear probability density, whereas, in reality, workers tend to provide reasonably consistent results, which translates to higher probabilities around the conditional mean. We use these best performing distributions in all subsequent experiments. In case of a returning worker, we use the corresponding worker-dependent model. Otherwise, we assume the worker performs averagely, and instead, use the worker-independent model.

4.6. TurKontrol on Mechanical Turk

Now that we have learned the POMDP model, our final evaluation assesses the benefits of the dynamic workflow controlled by TurKontrol versus a non-adaptive workflow (as originally used in TurKit [37]) under similar monetary con-

15 While this is obviously an approximation, we find it is surprisingly close; $R^2 = 0.82$ for the worker-independent model.

16 We use Newton's method with 1000 random restarts. Initial values are chosen uniformly from the real interval $(0, 100.0)$.

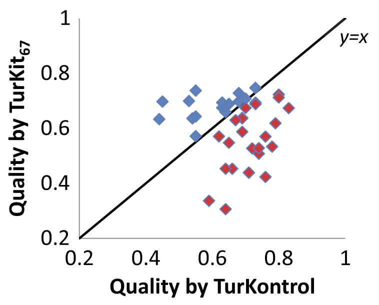


Fig. 14. Average qualities of 40 descriptions generated by TurKontrol and by TurKit under the same monetary consumption. TurKontrol generates statistically-significant higher-quality descriptions than TurKit.

sumption settings. We aim to answer the following questions: (1) Is there a significant quality difference between artifacts produced using TurKontrol and TurKit? (2) What are the qualitative differences between the two workflows?

As before, we run our experiments using the image description task. We use 40 fresh pictures from the Web and employ iterative improvement to generate descriptions for these. For each picture, we restrict a worker to take part in at most one task in each setting (*i.e.*, non-adaptive or dynamic). We set the user interfaces to be identical for both settings and randomize the order in which the two conditions are presented to workers in order to eliminate human learning effects. Altogether there are 655 participating workers, of which 57 take part in both settings.

We devise automated rules to detect spammers. We reject an improvement job if the new artifact is identical to the original. We reject ballot and scoring jobs if they are returned so quickly that the worker could not have made a reasonable judgment.

Note that our system does not need to learn a model for a new worker before assigning them jobs; instead, it uses the worker-independent parameters $\bar{\gamma}$ and $P(q'|q)$ as a prior. These parameters are incrementally updated as TurKontrol obtains more information.

Recall that TurKontrol performs decision-theoretic control based on a user-defined reward function. We define the reward for submitting an artifact of quality q to be $R(q) = \$25q$ for our experiments. We set the cost of an improvement job to be 5 cents and a ballot job to be 1 cent. We use the 3-step lookahead algorithm for the controller. Under these parameters, TurKontrol-workflows run an average of 6.25 iterations with an average of 2.32 ballots per iteration, costing about 46 cents per image description on average.

We use TurKit's original non-adaptive policy for ballots, which requests a third ballot if the first two voters disagree. We compute the number of iterations for TurKit so that the total money spent matches TurKontrol's. Since this number comes to be 6.47 we compare against three cases: TurKit$_6$ with 6 iterations, TurKit$_7$ with 7 iterations and TurKit$_{67}$ a weighted average of the two that equalizes monetary consumption.

For each final description we create a scoring job in which multiple workers score the descriptions. Fig. 14 compares the artifact qualities generated by TurKontrol and by TurKit$_{67}$ for the 40 images. We note that most points are below the $y = x$ line, indicating that the dynamic workflow produces superior descriptions. Furthermore, the quality produced by TurKontrol is greater on average than TurKit's, and the difference is statistically significant: $p < 0.01$ for TurKit$_6$, $p < 0.01$ for TurKit$_{67}$ and $p < 0.05$ for TurKit$_7$, using the Student's t-test.

Using our parameters, TurKontrol generates some of the highest-quality descriptions with an average quality of 0.67. TurKit$_{67}$'s average quality is 0.60; furthermore, it generates the two worst descriptions with qualities below 0.3. Finally, the standard deviation for TurKontrol is lower (0.09) than TurKit's (0.12). These results demonstrate overall superior performance of decision-theoretic control on live workflows.

While the 11% average quality increase produced by TurKontrol is statistically significant, some wonder if it is *material*. To better illustrate the importance of quality, we include another experiment. We run the nonadaptive, TurKit policy for additional improvement iterations, until it produces artifacts with an average quality equal to that produced by TurKontrol. Fixing the quality threshold, the TurKit policy has to run an average of 8.76 improvements, compared to the 6.25 improvement iterations used by TurKontrol. As a result the nonadaptive policy spends 28.7% more money than TurKontrol to achieve the same quality results. Note that final artifact quality is neither linear in the number of iterations nor total cost. Intuitively, it is much easier to improve an artifact when its quality is low than when it is high.

We also qualitatively study TurKontrol's behavior compared to TurKit's and find an interesting difference in the use of ballots. Fig. 15 plots the average number of ballots per iteration number. Since TurKit's ballot policy is fixed, it consistently uses about 2.45 ballots per iteration. TurKontrol, on the other hand, uses ballots much more intelligently. In the first two improvement iterations TurKontrol does not bother with ballots because it expects that most workers will improve the artifact. As iterations increase, TurKontrol increases its use of ballots, because the artifacts are harder to improve in later iterations, and hence TurKontrol needs more information before deciding which artifact to promote to the next iteration. The eighth iteration is an interesting exception; at this point improvements have become so rare that if even the first voter rates the new artifact as a loser, then TurKontrol often believes the verdict.

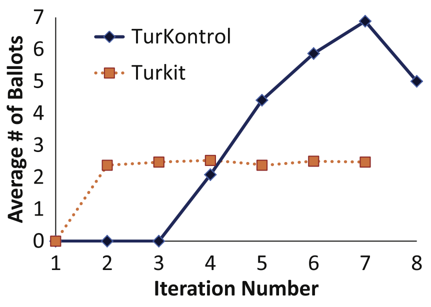


Fig. 15. Average number of ballots for the non-adaptive and dynamic workflows. TurKontrol makes intelligent use of ballots.

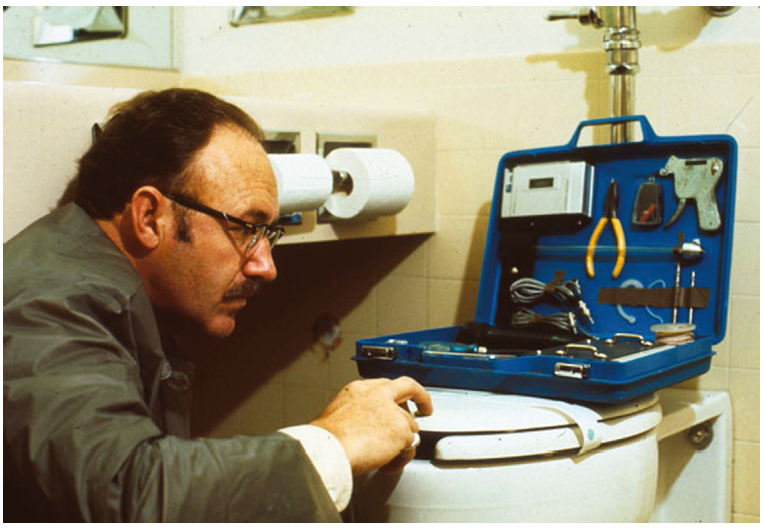


Fig. 16. An image description example. It took TurKontrol 6 improvement jobs and 14 ballot jobs to reach the final version: "This is Gene Hackman, in a scene from his film "The Conversation," in which he plays a man paid to secretly record people's private conversations. He is squatting in a bathroom gazing at tape recorder which he has concealed in a blue toolbox that is now placed on a hotel or motel commode (see paper strip on toilet seat). He is on the left side of the image in a gray jacket while the commode is on the right side of the picture. His fingertips rest on the lid of the commode. He is wearing a black court and a white shirt. He has put on glasses also." It took the nonadaptive workflow 6 improvement jobs and 13 ballot jobs to reach a version: "A thought about repairing: Image shows a person named Gene Hackman is thinking about how to repair the toilet of a hotel room. He has opened his tool box which contains plier, screw diver, wires, etc. He looks seriously in his tool box and thinking which tool he willuse. Wearing a grey coat he sits in front of the toilet seat resting gently on the toilet seat".

Besides using ballots intelligently we believe that TurKontrol adds two other kinds of reasoning. First, six of the seven pictures that TurKontrol finished in 5 iterations have higher qualities than TurKit's. This suggests that its quality tracking is working well. Perhaps due to the agreement among various voters, TurKontrol is able to infer that a description already has quality high enough to warrant termination. Secondly, TurKontrol has the ability to track individual workers, and this also affects its posterior calculations. For example, in one instance TurKontrol decided to trust the first vote because that worker had superior accuracy as reflected in a low error parameter. We expect that for repetitive tasks this will be an enormously valuable ability, since TurKontrol will be able to construct more informed worker models and make more superior decisions.

We present an image description example in Fig. 16. It is interesting to note that both processes managed to find the origin of the image. However, the TurKontrol version exemplifies better use of language, factuality and level of detail. In retrospect, we find the nonadaptive workflow probably made a wrong ballot decision in the sixth iteration, where a decision was critical yet only three voters were consulted. TurKontrol on the other hand, reached a decision after 6 unanimous votes at the same stage.

5. Decision-theoretic optimization of multiple workflows

Now that we have explored the decision-theoretic control of a single workflow, we consider the scenario when the requester has more than one [36]. In our exploration, for simplicity, we will only consider binary classification workflows. However, we note that the ideas we present are equally applicable for more complex workflows, like the iterative improvement workflow we considered earlier.

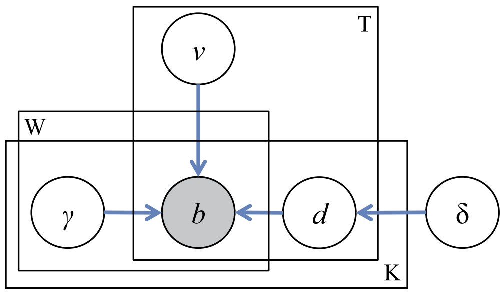


Fig. 17. Worker's answer b depends on the difficulty of the question d (generated by δ), worker's error parameter γ and the question's true answer v. There are W workers, who complete T tasks, all of which can be solved using a set of K different workflows. b is the only observed variable.

To see why it is useful to switch between workflows, consider the process task designers use to design their workflows. Typically, they will quantitatively experiment with several alternative workflows to accomplish the task, but choose a single one for the production runs (*e.g.* the workflow that achieves the best performance during early testing). In the simplest case, alternative workflows may differ only in their user interfaces or instructions; in other cases, workers may be asked to solve a problem with a very different set of steps.

Unfortunately, this seemingly natural design paradigm does not achieve the full potential of crowdsourcing. Selecting a single best workflow is suboptimal, because alternative workflows can compose synergistically to attain higher-quality results. While a given workflow may have the best performance on average, it is not necessarily best on every problem.

Suppose after gathering some answers for a task, one wishes to further increase one's confidence in the results; which workflow should be invoked? Due to the very fact that it is different, an alternative workflow may offer independent evidence, and this can significantly bolster one's confidence in the answer. If the "best" workflow is giving mixed results for a task, then an alternative workflow is often the best way to disambiguate. Instead of selecting one a priori best workflow, a better solution should reason about this potential synergy. Our POMDP model automatically tracks the expected accuracy of alternative workflows, switching away from the "on average best" workflow if intermediate results lead it to conclude another might be superior for the particular instance of the task at hand.

5.1. Probabilistic model for multiple workflows

We extend our generative model of worker responses to incorporate the existence of multiple workflows to complete the same task. It includes multiple, workflow-specific error parameters for each worker and workflow-specific difficulties.

Now, there are K alternative workflows that a worker could use to arrive at an answer. Let $d^k \in [0,1]$ denote the inherent difficulty of completing a task using workflow k, and let $\gamma_w^k \in [0,\infty)$ be worker w's error parameter for workflow k. Notice that every worker has K error parameters. Having several parameters per worker incorporates the insight that some workers may perform well when asked the question in one way (*e.g.*, visually) but not so well when asked in a different way (*e.g.*, when asked in English, since their command of that language may not be great).

The accuracy of a worker w, $a(d^k,\gamma_w^k)$, is the probability that she produces the correct answer using workflow k. We rewrite our original definition of worker accuracy accordingly:

$$a\big(d^k,\gamma_w^k\big)=\frac{1}{2}\big(1+\big(1-d^k\big)^{\gamma_w^k}\big). \tag{17}$$

Fig. 17 illustrates the plate notation for our generative model, which encodes a Bayes Net for responses made by W workers on T tasks, all of which can be solved using a set of K alternative workflows. The correct answer, v, the difficulty parameter, d, and the error parameter, γ, influence the final answer, b, that a worker provides, which is the only observed variable. d is generated by δ, a K-dimensional random variable describing a joint distribution on workflow difficulties. The answer b_w^k that worker w with error parameter γ_w^k provides for a task using workflow k is governed by the following equations:

$$P\big(b_w^k=v\,\big|\,d^k\big)=a\big(d^k,\gamma_w^k\big), \tag{18}$$

$$P\big(b_w^k\neq v\,\big|\,d^k\big)=1-a\big(d^k,\gamma_w^k\big). \tag{19}$$

As before, an underlying assumption is that given the workflow difficulty, d^k, and the true answer v, the b_w^k's are independent of each other. δ encodes the assumption that workflows may not be independent of each other. The fact that one workflow is easy might imply that a related workflow is easy. Finally, we still assume that the workers do not collaborate with each other and that they are not adversarial, *i.e.*, they do not purposely submit incorrect answers.

Now we discuss how to dynamically switch between workflows to obtain the highest accuracy for a given task.

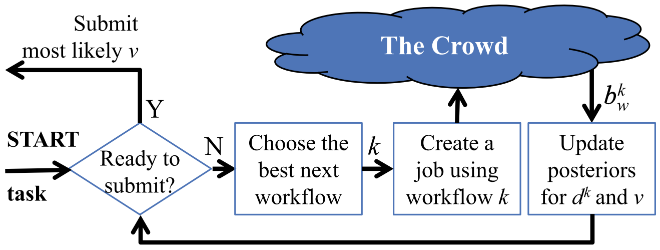

Submit
most likely v
The Crowd
Y
N
START
task
Ready to submit?
Choose the best next workflow
k
Create a job using workflow k
Update posteriors for d^k and v
b^k_w

Fig. 18. AgentHunt's decisions when executing a task.

5.2. A decision-theoretic agent

In this section, we answer the following question: Given a specific task that can be accomplished using alternative workflows, how do we design an agent that can leverage the availability of these alternatives by dynamically switching between them, in order to achieve a high-quality solution? We design an automated agent, named AgentHunt, that uses the following POMDP.

Definition 7. The POMDP is a six-tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{O}, P \rangle$, where

- $\mathcal{S} = \{(d^1, d^2, \ldots, d^K, v) | d^k \in [0,1], v \in \{0,1\}\}$ d^k is the difficulty of the kth workflow. v is the true answer.
- $\mathcal{A} = \{ballot^1, ballot^2, \ldots, ballot^K, submit\ true, submit\ false\}$.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is described below.
- $\mathcal{T}$: Identity map.
- $\mathcal{O} = \{true, false\}$ is a Boolean answer received for a ballot question.
- $P : \mathcal{S} \times \mathcal{O} \to [0,1]$ is defined by our generative model.

As in the POMDP describing our simple binary classification workflow, the reward function maintains the value of submitting a correct answer and the penalty for submitting an incorrect answer. Additionally, it maintains a cost that AgentHunt incurs when it creates a job. We can modify the reward function to match our desired budgets and accuracies.

Fig. 18 is a flow-chart of decisions that AgentHunt has to take.

As before, we assume that every future worker is an average worker. In other words, for a given workflow k, every future worker has an error parameter equal to $\bar{\gamma}^k = \frac{1}{W} \sum_w \gamma^k_w$ where W is the number of workers.

Also as before, after submitting an answer, AgentHunt updates its records about all the workers who participated in the task using what it believes to be the correct answer, using the following modified update rules. For a worker w who submitted an answer using workflow k, $\gamma^k_w \leftarrow \gamma^k_w - d^k \alpha$, should the worker answer correctly, and $\gamma^k_w \leftarrow \gamma^k_w + (1 - d^k)\alpha$, should the worker answer incorrectly, where α is a learning rate. Any worker that AgentHunt has not seen previously begins with the average $\bar{\gamma}^k$.

5.3. Learning the model

In order to behave optimally, AgentHunt needs to learn all $\boldsymbol{\gamma}$ values, average worker error parameters $\bar{\boldsymbol{\gamma}}$, and the joint workflow difficulty prior δ, which is a part of its initial belief. We consider two unsupervised approaches to learning – offline batch learning and online RL.

5.3.1. Offline learning

In this approach we first collect training data by having a set of workers complete a set of tasks using a set of workflows. This generates a set of worker responses, **b**. Since the true answer values **v** are unknown, an option is supervised learning, where experts label true answers and difficulties. However, this option requires significant expert time upfront, so instead, we use an EM algorithm and learn all parameters jointly.

For EM purposes, we simplify the model by removing the joint prior δ, and treat the variables **d** and $\boldsymbol{\gamma}$ as parameters. In the E-step, we keep parameters fixed to compute the posterior probabilities of the hidden true answers: $p(v_t | \mathbf{b}, \mathbf{d}, \boldsymbol{\gamma})$ for each task t. The M-step uses these probabilities to maximize the standard expected complete log-likelihood Q over **d** and $\boldsymbol{\gamma}$:

$$Q(\mathbf{d}, \boldsymbol{\gamma}) = E\left[\ln p(\mathbf{v}, \mathbf{b} | \mathbf{d}, \boldsymbol{\gamma})\right] \tag{20}$$

where the expectation is taken over **v** given the old values of $\boldsymbol{\gamma}$ and **d**.

After estimating all the hidden parameters, AgentHunt can compute $\bar{\gamma}^k$ for each workflow k by taking the average of all the learned $\boldsymbol{\gamma}^k$ parameters. Then, to learn δ, we can fit a Truncated Multivariate Normal distribution to the learned **d**. This difficulty prior determines a part of the initial belief state of AgentHunt. We complete the initial belief state by assuming the correct answer is distributed uniformly among the 2 alternatives.

5.3.2. *Online reinforcement learning*

Offline learning of our model can be very expensive, both temporally and monetarily. Moreover, we cannot be sure how much training data is necessary before the agents are ready to act in the real-world. An ideal AI agent will learn while acting in the real-world, tune its parameters as it acquires more knowledge, while still producing meaningful results. We modify AgentHunt to build its RL twin, AgentHunt$_{RL}$, which is able to accomplish tasks right out of the box.

AgentHunt$_{RL}$ starts with uniform priors on difficulties of all workflows. When it begins a new task, it uses the existing parameters to recompute the best policy and uses that policy to guide the next set of decisions. After completing the task, AgentHunt$_{RL}$ recalculates the maximum-likelihood estimates of the parameters $\boldsymbol{\gamma}$ and $\mathbf{d}$ using EM as above. The updated parameters define a new POMDP for which our agent computes a new policy for the future tasks. This relearning and POMDP-solving can be time-consuming, but we do not have to relearn and resolve after completing every task. We can easily speed the process by solving a few tasks before launching a relearning phase.

As in all of RL, AgentHunt$_{RL}$ must also make a tradeoff between taking possibly suboptimal actions in order to learn more about its model of the world (exploration), or taking actions that it believes to be optimal (exploitation). AgentHunt$_{RL}$ uses a modification of the standard ϵ-greedy approach [62]. With probability ϵ, AgentHunt$_{RL}$ will uniformly choose between suboptimal actions. The exception is that it will never submit an answer that it believes to be incorrect, since doing so would not help it learn anything about the world.

5.4. *Experiments*

This section addresses the following three questions. (1) In practice, how much value can be gained from switching between different workflows for a task? (2) What is the tradeoff between cost and accuracy? and (3) Previous decision-theoretic crowdsourcing systems have required an initial training phase; can reinforcement learning provide similar benefits without such training? We choose an NLP labeling task, for which we create $K = 2$ alternative workflows (described below). To answer the first two questions, we compare two agents: TurKontrol$_0$, which we will now refer to from now on as TurKontrol, a state-of-the-art controller for optimizing the execution of a single (best) workflow, and our AgentHunt, which can switch between the two workflows dynamically. We first compare them in simulation; then we allow the agents to control live workers on Amazon Mechanical Turk.

We answer the third question by comparing AgentHunt with AgentHunt$_{RL}$.

5.4.1. *Implementation*

The POMDP must manage a belief state over the cross product of the Boolean answer and two continuous variables – the difficulties of the two workflows. Since solving a POMDP with a continuous state space is challenging, we discretize difficulty into eleven possible values, leading to a (world) state space of size $2 \times 11 \times 11 = 242$. To solve POMDPs, we run the ZMDP package[17] for 300 s using the default Focused Real-Time Dynamic Programming search strategy [57]. Since we can cache the complete POMDP policy in advance, AgentHunt can control workflows in real time.

Since we have discretized difficulty, we also modify the learning process slightly. After we learn all values of $\mathbf{d}$, we round the values to the nearest discretizations and construct a histogram to count the number of times every state appears in the training data. Then, before we use the implicit joint distribution as the agent's starting belief state, we smooth it by adding 1 to every bin (Laplace smoothing).

5.4.2. *Evaluation task: NER tagging*

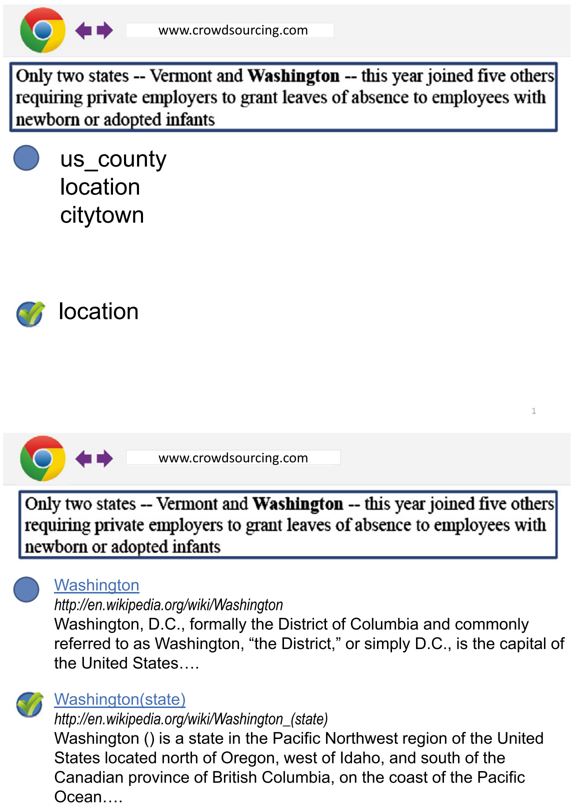
In order to test our agents, we select a task that is needed by several colleagues: labeling training data for named-entity recognition (NER) [52]. NER tagging is a common problem in NLP and information extraction: given a body of text (*e.g.*, "Barack Obama thinks this research is not bad.") and a subsequence of that text (*e.g.*, "Barack Obama") that specifies an entity, output a set of tags that classify the type of the entity (*e.g.*, *person*, *politician*). Since machine learning techniques are used to create production NER systems, large amounts of labeled data (of the form described above) are needed. Obtaining the most accurate training data at minimal cost, is therefore, an excellent test of our methods.

In consultation with NER domain experts we develop two workflows for the task (Fig. 19). Both workflows begin by providing users with a body of text and an entity, like "**Nixon** concluded five days of private talks with Chinese leaders in Beijing." The first workflow, called "WikiFlow," first uses *Wikification* [41,46] to find a set of possible Wikipedia articles describing the entity, such as "Nixon (film)" and "Richard Nixon." It displays these articles (including the first sentence of each article) and asks workers to choose the one that best describes the entity. Finally, it returns the Freebase[18] tags associated with the Wikipedia article selected by the worker.

The second workflow, "TagFlow," asks users to choose the best set of Freebase tags directly. For example, the Freebase tags associated with "Nixon (film)" is $\{\texttt{/film/film}\}$, while the tags associated with "Richard Nixon" includes $\{\texttt{/people/person}, \texttt{/government/us} - \texttt{congressperson}\}$. TagFlow displays the tag sets corresponding to the different options and asks the worker to choose the tag set that best describes the entity mentioned in the sentence.

[17] http://www.cs.cmu.edu/~trey/zmdp/.

[18] www.freebase.com.



www.crowdsourcing.com

Only two states -- Vermont and **Washington** -- this year joined five others requiring private employers to grant leaves of absence to employees with newborn or adopted infants

● us_county
location
citytown

✓ location

www.crowdsourcing.com

Only two states -- Vermont and **Washington** -- this year joined five others requiring private employers to grant leaves of absence to employees with newborn or adopted infants

● Washington
http://en.wikipedia.org/wiki/Washington
Washington, D.C., formally the District of Columbia and commonly referred to as Washington, “the District,” or simply D.C., is the capital of the United States….

✓ Washington(state)
http://en.wikipedia.org/wiki/Washington_(state)
Washington () is a state in the Pacific Northwest region of the United States located north of Oregon, west of Idaho, and south of the Canadian province of British Columbia, on the coast of the Pacific Ocean….

Fig. 19. In this NER task, the TagFlow (top) is considerably harder than the WikiFlow (bottom) since the tags are very similar. The correct tag set is {location} since Washington state is neither a county nor a citytown.

5.4.3. *Experimental setup*

First, we gather training data using Mechanical Turk. We generate 50 NER tasks. For each task, we submit 40 identical WikiFlow jobs and 40 identical TagFlow jobs to Mechanical Turk. At \$0.01 per job, the total cost is \$60.00 including Amazon commission. Using our EM technique, we then calculate average worker accuracies, $\bar{\gamma}^{WF}$ and $\bar{\gamma}^{TF}$, corresponding to WikiFlow and TagFlow respectively. Somewhat to our surprise, we find that $\bar{\gamma}^{TF} = 0.538 < \bar{\gamma}^{WF} = 0.547$ – on average, workers found TagFlow to be very slightly easier than WikiFlow. Note that this result implies that AgentHunt will always create a TagFlow job to begin a task. We also note that the difference between $\bar{\gamma}^{TF}$ and $\bar{\gamma}^{WF}$ influences the switching behavior of AgentHunt. Intuitively, if AgentHunt were given two workflows whose average difficulties were further apart, AgentHunt would become more reluctant to switch to a harder workflow. Because we find TagFlow jobs to be slightly easier, for all experiments, we set TurKontrol so it creates TagFlow jobs. We also use this training data to construct both agents' initial beliefs.

5.4.4. *Experiments using simulation*

We first run our agents in a simulated environment. On each run, the simulator draws states from the agents' initial belief distributions. We fix the reward of returning the correct answer to 0, and vary the reward (penalty) of returning an incorrect answer between the following values: -10, -100, -1000, and $-10\,000$. We set the cost of creating a job for a (simulated) worker to -1. We use a discount factor of 0.9999 in the POMDP. For each setting of reward values we run 1000 simulations and report mean net utilities (Fig. 21).

We find that when the stakes are low, the two agents behave almost identically. However, as the penalty for an incorrect answer increases, AgentHunt's ability to switch between workflows allows it to capture much more utility than TurKontrol. As expected, both agents submit an increasing number of jobs as the importance of answer correctness rises and in the process, both of their accuracies rise too (Fig. 20). However, while both agents become more accurate, TurKontrol does not increase its accuracy enough to compensate for the exponentially growing penalties. Instead, as AgentHunt experiences an almost sublinear decline in net utility, TurKontrol sees an exponential drop (Fig. 21).

A Student's t-test shows that for all settings of penalty except -10, the differences between the two systems' average net utilities are statistically significant. When the penalty is -10, $p < 0.4$, and at all other reward settings, $p < 0.0001$. Thus we find that at least in simulation, AgentHunt outperforms TurKontrol on all our metrics.

We also analyze the systems' behaviors qualitatively. As expected, AgentHunt always starts by creating a TagFlow job, since $\bar{\gamma}^{TF} < \bar{\gamma}^{WF}$ implies that TagFlows lead to higher worker accuracy on average. Interestingly, although AgentHunt has

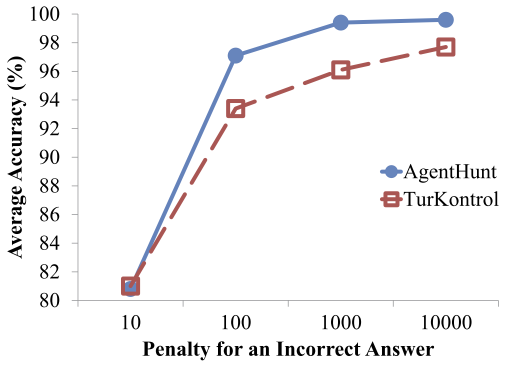


Fig. 20. In simulation, as the importance of answer correctness increases, both agents converge to 100 percent accuracy, but AgentHunt does so more quickly.

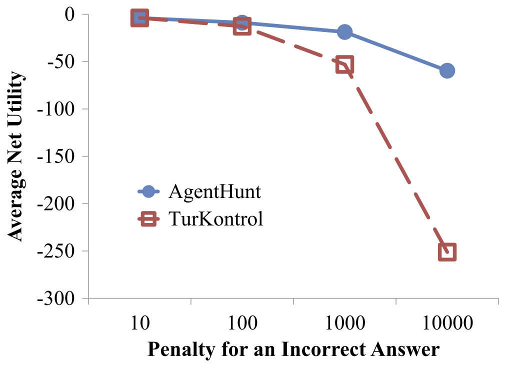


Fig. 21. In simulation, as the importance of answer correctness increases, AgentHunt outperforms TurKontrol by an ever-increasing margin.

Table 3
Comparisons of accuracies, costs, and net utilities of various agents when run on Mechanical Turk.

	AgentHunt	TurKontrol	TurKontrol$_{300}$
Avg accuracy (%)	92.45	85.85	84.91
Avg cost	5.81	4.21	6.26
Avg net utility	−13.36	−18.35	−21.35

more available workflows, it creates fewer actual jobs than TurKontrol, even as correct answers become increasingly important. We also split our problems into three categories to better understand the agents' behaviors: 1) TagFlow is easy, 2) TagFlow is hard, but WikiFlow is easy, and 3) Both workflows are difficult.

In the first case, both agents terminate quickly, though AgentHunt spends a little more money since it also requests WikiFlow jobs to double-check what it learns from TagFlow jobs. In the second case, TurKontrol creates an enormous number of jobs before it decides to submit an answer, while AgentHunt terminates much faster, since it quickly deduces that TagFlow is hard and switches to creating easy WikiFlow jobs. In the third case, AgentHunt expectedly creates more jobs than TurKontrol before terminating, but AgentHunt does not do too much worse than TurKontrol, since it correctly deduces that gathering more information is unlikely to help.

5.4.5. Experiments using Mechanical Turk

We next run the agents on real data gathered from Mechanical Turk. We generate 106 new NER tasks for this experiment, and use gold labels supplied by a single expert. Since in our simulations we found that the agents spend on average about the same amount of money when the reward for an incorrect answer is −100, we use this reward value in our real-world experiments.

As Table 3 shows, AgentHunt fares remarkably better in the real-world than TurKontrol. A Student's t-test shows that the difference between the average net utilities of the two agents is statistically significant with $p < 0.03$. However, we see that TurKontrol spends less, leading one to naturally wonder whether the difference in utility can be accounted for by the cost discrepancy. Thus, we modify the reward for an incorrect answer (to −300) for TurKontrol to create TurKontrol$_{300}$, which spends about the same amount of money as AgentHunt.

Table 4
Comparisons of accuracies, costs, and net utilities of various agents when run on Mechanical Turk.

	AgentHunt	AgentHunt$_{RL}$
Avg accuracy (%)	92.45	93.40
Avg cost	5.81	7.25
Avg net utility	−13.36	−13.85

But even after the modification, the accuracy of AgentHunt is still much higher. A Student's t-test shows that the difference between the average net utilities of AgentHunt and TurKontrol$_{300}$ is statistically significant at $p < 0.01$ showing that in the real-world, given similar budgets, AgentHunt produces significantly better results than TurKontrol. Indeed, AgentHunt reduces the error of TurKontrol by 45% and the error of TurKontrol$_{300}$ by 50%. Surprisingly, the accuracy of TurKontrol$_{300}$ is lower than that of TurKontrol despite the additional jobs; we attribute this to statistical variance.

5.4.6. *Adding reinforcement learning*

Finally, we compare AgentHunt to AgentHunt$_{RL}$. AgentHunt$_{RL}$'s starting belief state is a uniform distribution over all world states and it assumes that $\bar{\gamma}^k = 1$ for all workflows. To encourage exploration, we set $\epsilon = 0.1$. We test it using the same 106 tasks described above. Table 4 reproduces the results of AgentHunt from Table 3 alongside those of AgentHunt$_{RL}$. We see that while AgentHunt$_{RL}$ achieves a slightly higher accuracy than AgentHunt, the difference between their net utilities is not statistically significant ($p = 0.4$), which means AgentHunt$_{RL}$ is comparable to AgentHunt, suggesting that AgentHunt can perform in an "out of the box" mode, without needing a training phase.

6. Related work

Modeling repeated labeling in the face of noisy workers has received significant attention, but we are the first to use decision theory to *dynamically* poll for more labels as necessary. Dawid et al. [14] and Romney et al. [48] are one of the firsts to incorporate a worker accuracy model to improve label quality. Raykar et al. [47] propose a model in which the parameters for worker accuracy depend on the true answer. Whitehill et al. [67] address the concern that worker labels should not be modeled as independent of each other unless given problem difficulty. Welinder et al. [66] design a multidimensional model for workers that takes into account competence, expertise, and annotator bias. Kamar et al. [27] extract features from the task at hand and use Bayesian Structure Learning to learn the worker response model. Parameswaran et al. [43] conduct a policy search to find an optimal dynamic control policy with respect to constraints like cost or accuracy. Karger et al. [28] develop an algorithm to generate a graph that represents an assignment of tasks to workers and infers correct answers based on low-rank matrix approximation. Given the assumptions that all tasks share an equal level of difficulty and that workers are either always correct or randomly guess, they analytically prove the optimality of their algorithm at minimizing a budget for a target error rate with respect to any other possible algorithm using their assumptions. Our methods use more general models which fall outside their assumptions, but provide the utility maximization guarantees that any POMDP model inherits. Wauthier et al. [64] treat the entire crowdsourcing pipeline from data collection to learning within a Bayesian framework. Kajino et al. [26] formulate the repeated labeling problem as a convex optimization problem. Lin et al. [35] address the case when either requesters cannot enumerate all possible answers for the worker or when the solution space is infinitely large.

Other innovative workflows have been designed for complex tasks, for example, find–fix–verify for an intelligent editor [5], iterative dual pathways for speech-to-text transcription [34] and others for counting calories on a food plate [42]. Lasecki et al. [32] design a system that allows multiple users to control the same interface in real-time. Control can be switched between users depending on who is doing better. Kulkarni et al. [30] show the crowd itself can help with the design and execution of complex workflows.

Ipeirotis et al. [23] observe that workers tend to have bias on multiple-choice, annotation tasks. They learn a *confusion matrix* to model the error distribution of individual workers. However, their model assumes workers' errors are completely independent, whereas, our model handles situations where workers make correlated errors due to the intrinsic difficulty of the task.

Kulkarni et al. [31] design an alternative crowdsourcing platform that is not a marketplace. Instead, it intelligently routes work to workers and controls for accuracy and quality. Ho et al. [20] consider how to assign tasks to workers on arrival.

Huang et al. [22] look at the problem of designing a task under budget and time constraints. They illustrate their approach on an image-tagging task. By wisely setting variables, such as reward per task and the number of labels requested per image, they increase the number of useful tags acquired. Donmez et al. [16] observe that workers' accuracies often change over time (*e.g.*, due to fatigue, mood, task similarity, etc). Rzeszotarski et al. [51] propose to use micro-breaks to overcome fatigue effects. As these approaches are orthogonal to ours, we would like to integrate the methods in the future.

Shahaf and Horvitz [53] develop an HTN-planner style decomposition algorithm to find a coalition of workers, each with different skill sets, to solve a task. Some members of the coalition may be machines and others humans; different skills may command different prices. In contrast to our work, Shahaf and Horvitz do not consider methods for learning models of their workers.

The benefits from combining disparate workflows have been previously observed. Babbage's Law of Errors suggests that the accuracy of numerical calculations can be increased by comparing the outputs of two or more methods [19]. However, in previous work these workflows have been combined manually; AgentHunt embodies the first method for *automatically* evaluating potential synergy and dynamically switching between workflows.

7. Discussion and future work

We believe that AI has the potential to impact the growing thousands of requesters who use crowdsourcing to make their processes more efficient. Our work can be extended in several important directions towards realizing this vision. We list a few below.

Task economic uncertainty. We have so far assumed that the price for each task is a constant and an input. However, figuring out the actual value of a task is an important problem. This problem is difficult due to the highly dynamic nature of crowdsourcing markets. At the supply end, the value of a job can be influenced by workers' expertise, motivation of work, and personal interests. First, we plan to explore approaches that estimate value based on passively observing workflow activity. As a first step we will develop models that allow for the estimation of an effective hourly wage for a job. This problem is non-trivial given the many sources of variance including network delay, a worker's technology adequacy, whether a worker is multi-tasking, etc. Second, we will explore active algorithms for estimating these models. These approaches will actively experiment with payment choices in order to more efficiently learn a model. Further, for both the active and passive cases, we plan to extend this model to handle tasks with time constraints. This will require modeling the relationship between the amount of payment and the task completion speed.

Incentives. Paying an optional bonus for exceptional work can be a viable way of motivating high-quality results and forming long-term collaborations with competent workers. Questions that need to be answered are; (1) When to pay a bonus? (2) Who should receive a bonus? and (3) How much should the bonus be? An intuitive way is to pay a portion of the monetary savings of a good result. Another form of incentive involves recognition, such as leader boards. Investigations need to be carried out to determine which (combination of) bonus forms (and magnitude) can be the most effective. Here again we plan to study both passive and active approaches for modeling incentive structure in workflows.

A universal worker model. A worker's productivity varies over time. This is due to her fatigue level, variance of mood and work efficiency during the day, or the skillfulness acquired during her experiences with similar tasks. We plan to generalize our worker model by integrating several temporal parameters: (1) the time of the day and the day of the week or year, (2) a worker's consecutive working hours, and (3) her cumulative hours on the same type of tasks. Further, the performance of any crowdsourcing system can be lowered by noisy answers provided by spammers, or even malicious workers. An important aspect of our model will be to support the inference of spamming and malicious intent.

Example selection for reinforcement learning. Our work does not consider the problem of carefully curating the reinforcement learning process whether it be online or offline. For example, given a set of tasks, we do not know how large a subset of tasks we should solve before we update parameters, nor do we know which subset of tasks to use, and in which order. All these factors can affect the ability of our agent to learn parameters accurately and quickly. We wish to investigate these questions in future research.

Additive utility function. Our utility function is additive, in that each task instance is treated independently of each other. The total utility that we achieve for a set of tasks is equal to the sum of the utilities we achieve for each task. Such an assumption may not hold in other scenarios, *e.g.*, if we are trying to train a machine learning classifier using crowdsourced labels. Here, an active learning formalism may be more appropriate [15,55,69].

Abstractness of utility. To make our work usable for the vast majority, we must be able to elicit a utility function from our users. This is not an issue just with our approach, but with all decision-theoretic formalizations. The concept of utility is abstract, and as such, cannot be easily constructed even with domain knowledge. An extension of this work could consider ways to derive utilities from budgetary or accuracy constraints, which are far more concrete and comprehensible. Alternatively, we may cast this as a utility elicitation problem [9].

General-purpose crowdsourcing control. Our long-term research goal is to automatically control workflows consisting of a broader range of tasks [65]. To achieve this goal, we need to specify a general language for workflows, and develop methods for translating workflow specifications into decision problems. We must also build a worker model for each task type. However, we believe that many crowdsourcing tasks can be captured with a relatively small number of task classes, such as tasks with discrete alternatives, content generation, etc. By having a task library we can share parameters across similar tasks. Given a new task, we can transfer knowledge from related tasks in the library, thereby simplifying and expediting the model training process. We are currently developing a general-purpose workflow controller called CLOWDER, which, given any new workflow, will automatically convert it into a POMDP and control it for the requester. Such a software will be extremely important in realizing our vision, since then our techniques will be easily accessible to requesters who do not have advanced AI knowledge.

8. Conclusions

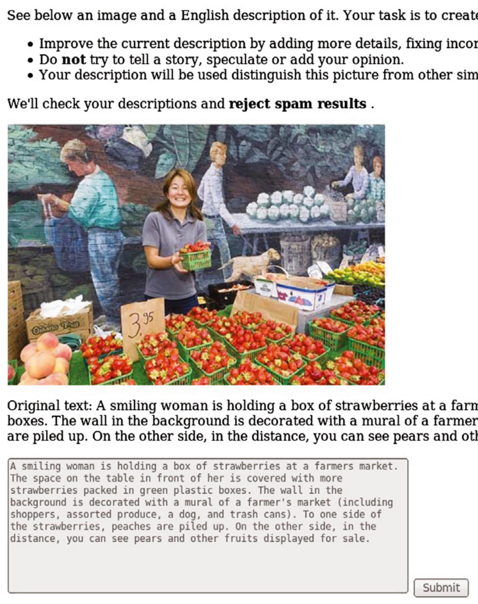
This work applies modern planning and machine learning techniques to the quality control problem of crowdsourcing, a new and rapidly-growing method for accomplishing various tasks, and demonstrates the usefulness of decision theory in a real-world setting, by controlling dynamic workflows that successfully achieve statistically-significant, higher-quality results than traditional, non-adaptive workflows. In particular, we make the following contributions.[19]

1. As complex workflows have become more commonplace in crowdsourcing and are regularly employed for high-quality output, we introduce an exciting new application for artificial intelligence, the control of these workflows.
2. We use POMDPs to model single workflows as well as multiple workflows, and define generative models that govern various observation models of the processes. Our agents implement our mathematical framework and use it to optimize and control selection and execution of workflows. Extensive simulations and real-world experiments show that our agents are robust in a variety of scenarios and parameter settings, and achieve higher utilities than previous, non-adaptive policies.
3. We present efficient and cheap mechanisms for learning model parameters from limited and noisy training data. We validate the parameters independently and show that our learned models significantly outperform the popular majority-vote baseline when resources are constrained.

Acknowledgements

Thanks to the anonymous reviewers and Jonathan Bragg for helpful comments on this manuscript. Conversations with Lydian Chilton and Greg Little informed our thoughts on crowdsourcing. This work was supported by the WRF/TJ Cable Professorship, Office of Naval Research grants N00014-12-1-0211 and N00014-06-1-0147, and National Science Foundation grants IIS 1016713 and IIS 1016465.

Appendix A. Snapshots of tasks



See below an image and a English description of it. Your task is to create a better description of it.

- Improve the current description by adding more details, fixing incorrect grammar and tightening the language.
- Do **not** try to tell a story, speculate or add your opinion.
- Your description will be used distinguish this picture from other similar photos; so accurate details and conciseness are crucial.

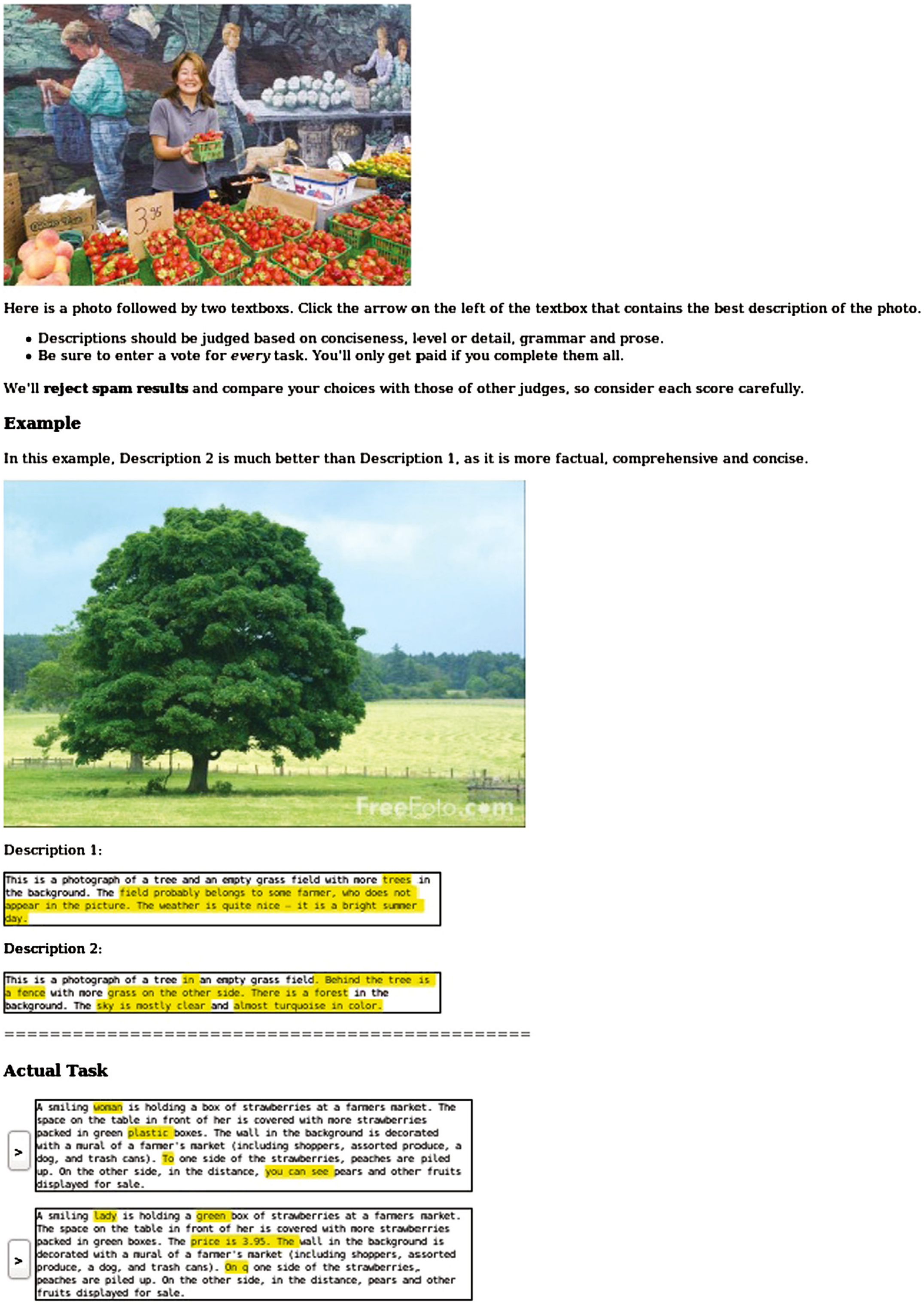
We'll check your descriptions and **reject spam results** .

Original text: A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

Fig. 22. Snapshot of an improvement HIT.

[19] Software packages of our implementations are available for general use at http://cs.washington.edu/node/7714.



Here is a photo followed by two textboxs. Click the arrow on the left of the textbox that contains the best description of the photo.

- Descriptions should be judged based on conciseness, level or detail, grammar and prose.
- Be sure to enter a vote for *every* task. You'll only get paid if you complete them all.

We'll **reject spam results** and compare your choices with those of other judges, so consider each score carefully.

Example

In this example, Description 2 is much better than Description 1, as it is more factual, comprehensive and concise.

Description 1:

This is a photograph of a tree and an empty grass field with more trees in the background. The field probably belongs to some farmer, who does not appear in the picture. The weather is quite nice – it is a bright summer day.

Description 2:

This is a photograph of a tree in an empty grass field. Behind the tree is a fence with more grass on the other side. There is a forest in the background. The sky is mostly clear and almost turquoise in color.

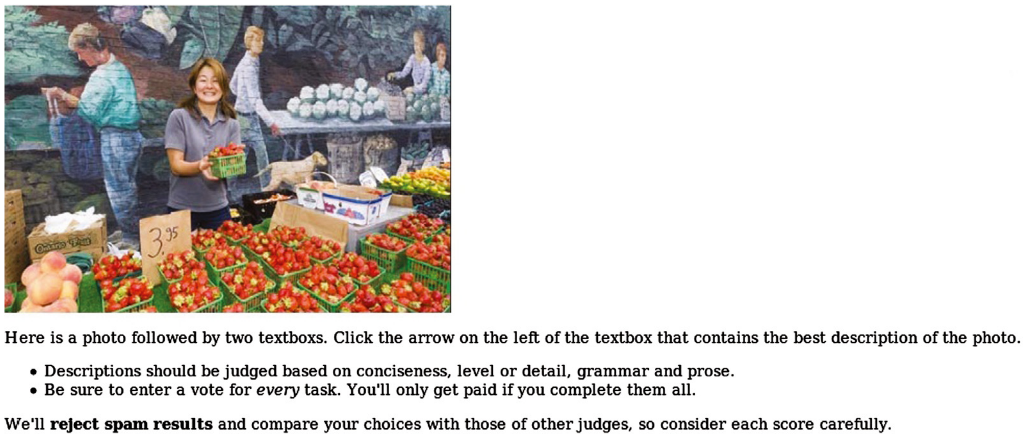
==

Actual Task

> A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

> A smiling lady is holding a green box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green boxes. The price is 3.95. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). On q one side of the strawberries, peaches are piled up. On the other side, in the distance, pears and other fruits displayed for sale.

Fig. 23. Snapshot of a complete ballot HIT.

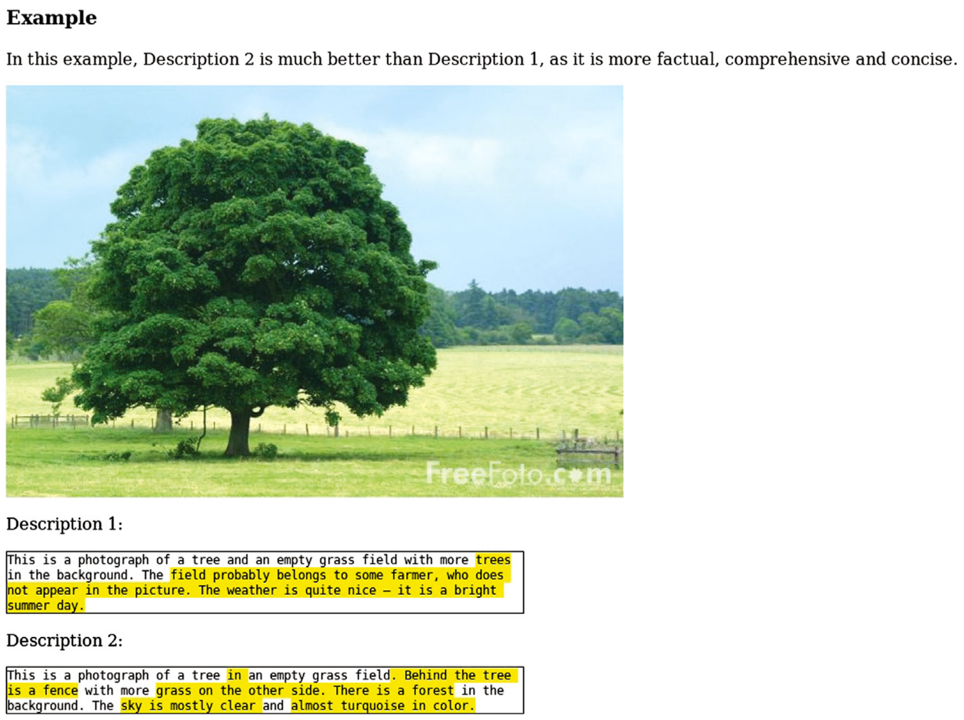


Here is a photo followed by two textboxs. Click the arrow on the left of the textbox that contains the best description of the photo.

- Descriptions should be judged based on conciseness, level or detail, grammar and prose.
- Be sure to enter a vote for *every* task. You'll only get paid if you complete them all.

We'll **reject spam results** and compare your choices with those of other judges, so consider each score carefully.

Fig. 24. Snapshot of the instruction portion of a ballot HIT.



Example

In this example, Description 2 is much better than Description 1, as it is more factual, comprehensive and concise.

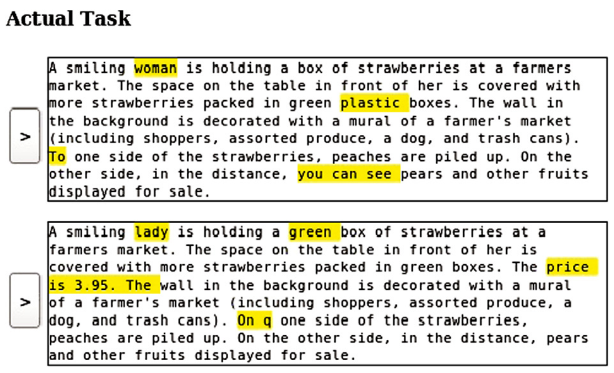
Description 1:

This is a photograph of a tree and an empty grass field with more trees in the background. The field probably belongs to some farmer, who does not appear in the picture. The weather is quite nice – it is a bright summer day.

Description 2:

This is a photograph of a tree in an empty grass field. Behind the tree is a fence with more grass on the other side. There is a forest in the background. The sky is mostly clear and almost turquoise in color.

Fig. 25. Snapshot of the example portion of a ballot HIT.

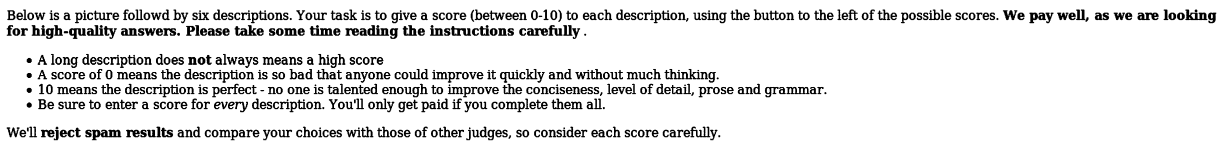


Actual Task

A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

A smiling lady is holding a green box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green boxes. The price is 3.95. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). On q one side of the strawberries, peaches are piled up. On the other side, in the distance, pears and other fruits displayed for sale.

Fig. 26. Snapshot of the actual task portion of a ballot HIT.

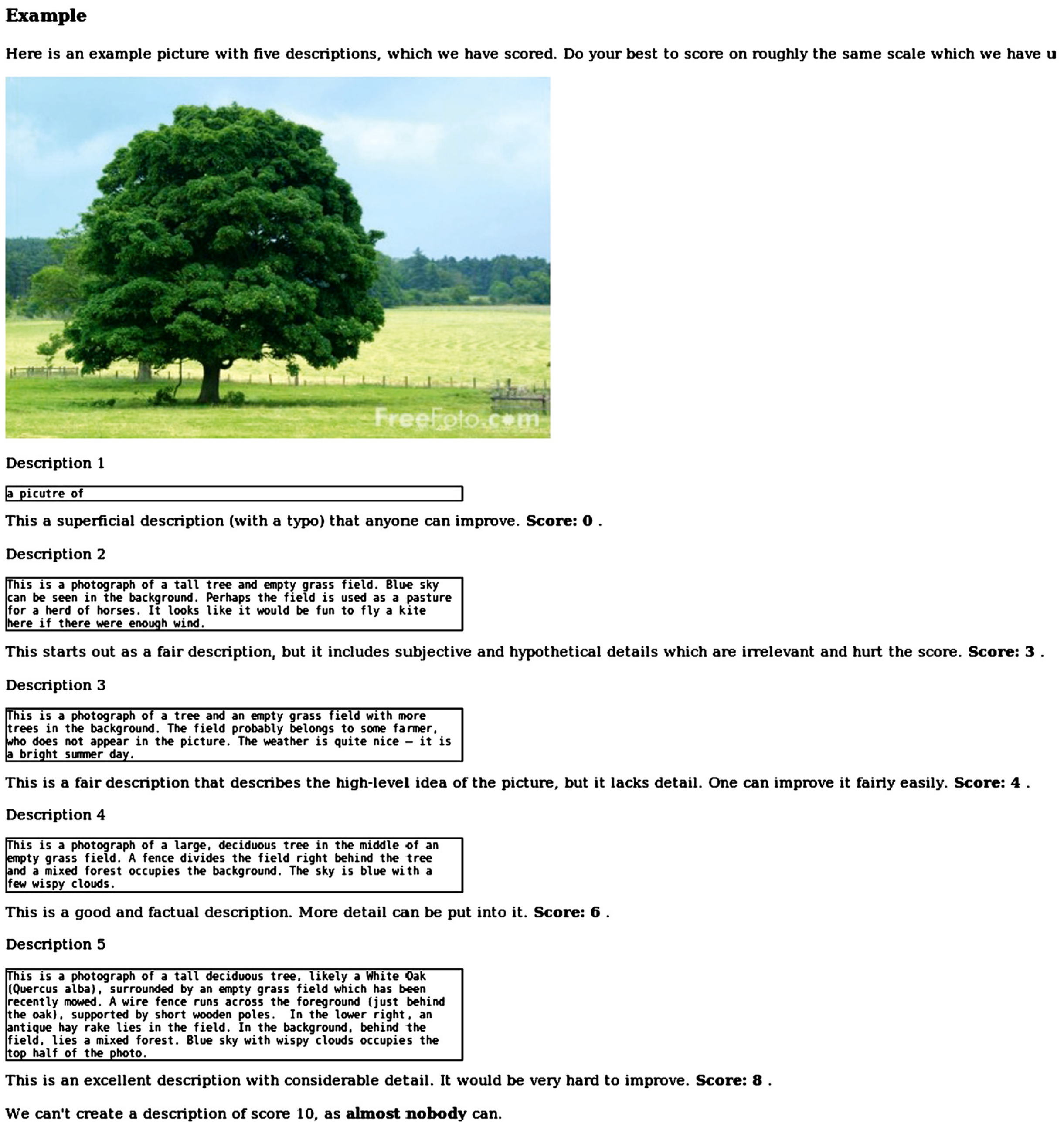


Below is a picture followd by six descriptions. Your task is to give a score (between 0-10) to each description, using the button to the left of the possible scores. **We pay well, as we are looking for high-quality answers. Please take some time reading the instructions carefully** .

- A long description does **not** always means a high score
- A score of 0 means the description is so bad that anyone could improve it quickly and without much thinking.
- 10 means the description is perfect - no one is talented enough to improve the conciseness, level of detail, prose and grammar.
- Be sure to enter a score for *every* description. You'll only get paid if you complete them all.

We'll **reject spam results** and compare your choices with those of other judges, so consider each score carefully.

Fig. 27. Snapshot of the instruction portion of a scoring HIT.



Example

Here is an example picture with five descriptions, which we have scored. Do your best to score on roughly the same scale which we have u

Description 1

```
a picutre of
```

This a superficial description (with a typo) that anyone can improve. Score: 0 .

Description 2

```
This is a photograph of a tall tree and empty grass field. Blue sky
can be seen in the background. Perhaps the field is used as a pasture
for a herd of horses. It looks like it would be fun to fly a kite
here if there were enough wind.
```

This starts out as a fair description, but it includes subjective and hypothetical details which are irrelevant and hurt the score. Score: 3 .

Description 3

```
This is a photograph of a tree and an empty grass field with more
trees in the background. The field probably belongs to some farmer,
who does not appear in the picture. The weather is quite nice — it is
a bright summer day.
```

This is a fair description that describes the high-level idea of the picture, but it lacks detail. One can improve it fairly easily. Score: 4 .

Description 4

```
This is a photograph of a large, deciduous tree in the middle of an
empty grass field. A fence divides the field right behind the tree
and a mixed forest occupies the background. The sky is blue with a
few wispy clouds.
```

This is a good and factual description. More detail can be put into it. Score: 6 .

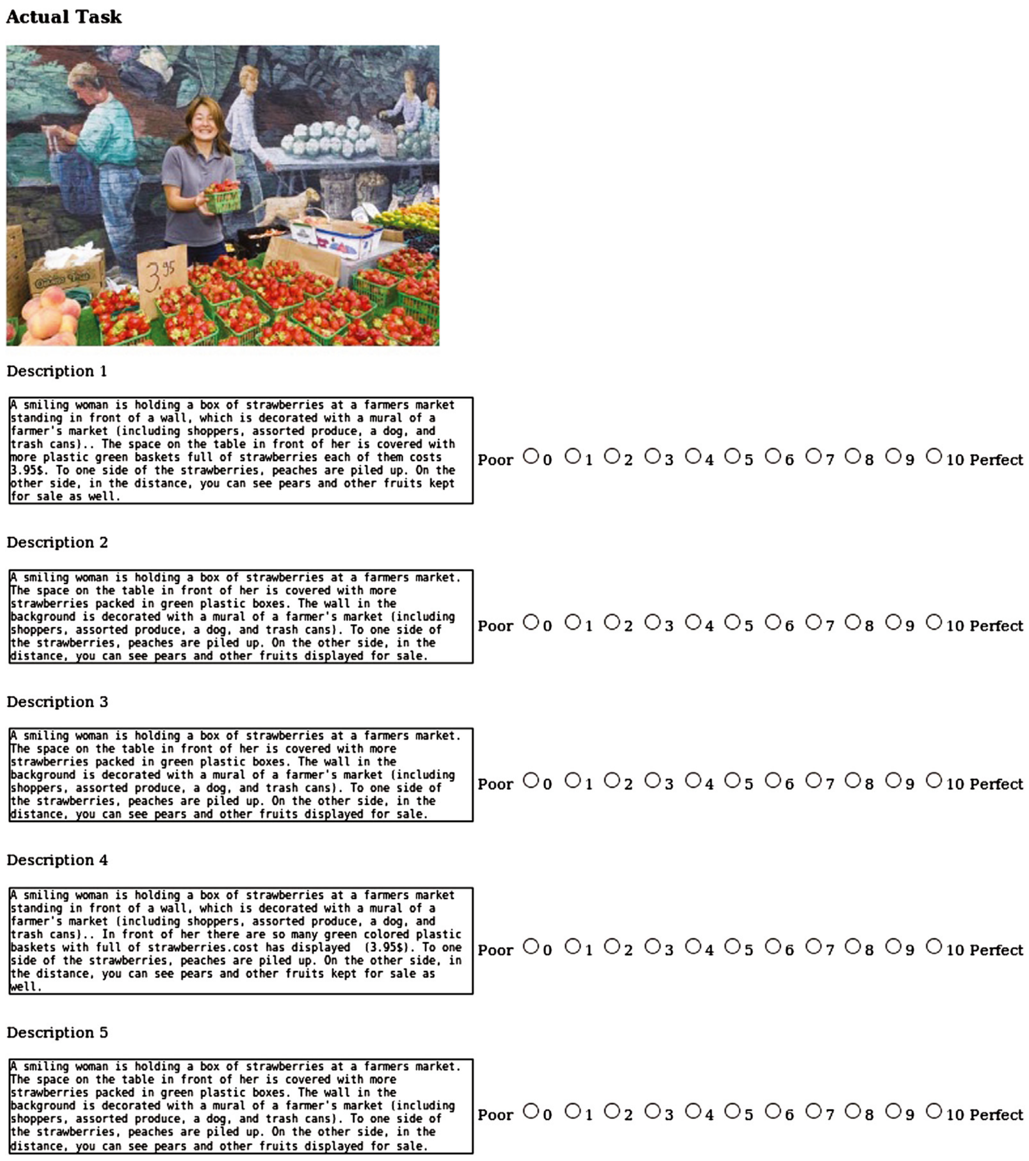
Description 5

```
This is a photograph of a tall deciduous tree, likely a White Oak
(Quercus alba), surrounded by an empty grass field which has been
recently mowed. A wire fence runs across the foreground (just behind
the oak), supported by short wooden poles.  In the lower right, an
antique hay rake lies in the field. In the background, behind the
field, lies a mixed forest. Blue sky with wispy clouds occupies the
top half of the photo.
```

This is an excellent description with considerable detail. It would be very hard to improve. Score: 8 .

We can't create a description of score 10, as almost nobody can.

Fig. 28. Snapshot of the example portion of a scoring HIT.



Actual Task

Description 1

A smiling woman is holding a box of strawberries at a farmers market standing in front of a wall, which is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans).. The space on the table in front of her is covered with more plastic green baskets full of strawberries each of them costs 3.95$. To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits kept for sale as well.

Poor ○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Perfect

Description 2

A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

Poor ○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Perfect

Description 3

A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

Poor ○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Perfect

Description 4

A smiling woman is holding a box of strawberries at a farmers market standing in front of a wall, which is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans).. In front of her there are so many green colored plastic baskets with full of strawberries.cost has displayed (3.95$). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits kept for sale as well.

Poor ○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Perfect

Description 5

A smiling woman is holding a box of strawberries at a farmers market. The space on the table in front of her is covered with more strawberries packed in green plastic boxes. The wall in the background is decorated with a mural of a farmer's market (including shoppers, assorted produce, a dog, and trash cans). To one side of the strawberries, peaches are piled up. On the other side, in the distance, you can see pears and other fruits displayed for sale.

Poor ○0 ○1 ○2 ○3 ○4 ○5 ○6 ○7 ○8 ○9 ○10 Perfect

Fig. 29. Snapshot of the actual task portion of a scoring HIT.

References

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

[2] R. Bellman, Dynamic Programming, Princeton University Press, Princeton, NJ, 1957.

[3] Michael S. Bernstein, Joel Brandt, Robert C. Miller, David R. Karger, Crowds in two seconds: Enabling realtime crowd-powered interfaces, in: UIST, 2011.

[4] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, Katrina Panovich, Soylent: A word processor with a crowd inside, in: UIST, 2010, pp. 313–322.

[5] Michael S. Bernstein, Greg Little, Robert C. Miller, Björn Hartmann, Mark S. Ackerman, David R. Karger, David Crowell, Katrina Panovich, Soylent: A word processor with a crowd inside, in: UIST, 2010.

[6] Dimitri P. Bertsekas, Dynamic Programming and Optimal Control, vol. 1, 2nd ed., Athena Scientific, 2000.

[7] Jeffrey P. Bigham, Chandrika Jayant, Hanjie Ji, Greg Little, Andrew Miller, Robert C. Miller, Robin Miller, Aubrey Tatarowicz, Brandyn White, Samuel White, Tom Yeh, Vizwiz: Nearly real-time answers to visual questions, in: UIST, 2010, pp. 333–342.

[8] Emma Brunskill, Leslie Kaelbling, Tomas Lozano-Perez, Nicholas Roy, Continuous-state POMDPs with hybrid dynamics, in: Symposium on Artificial Intelligence and Mathematics, 2008.

[9] Urszula Chajewska, Daphne Koller, Ronald Parr, Making rational decisions using adaptive utility elicitation, in: AAAI, 2000.

[10] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beene, Andrew Leaver-Fay, David Baker, Zoran Popovic, Foldit players, Predicting protein structures with a multiplayer online game, Nature 446 (2010) 756–760.

[11] Dan Cosley, Shyong K. Lam, Istvan Albert, Joseph A. Konstan, John Riedl, Is seeing believing?: How recommender system interfaces affect users' opinions, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, 2003, pp. 585–592.

[12] Peng Dai Mausam, Daniel S. Weld, Decision-theoretic control of crowd-sourced workflows, in: AAAI, 2010.

[13] Peng Dai Mausam, Daniel S. Weld, Artificial intelligence for artificial intelligence, in: AAAI, 2011.
[14] A.P. Dawid, A.M. Skene, Maximum likelihood estimation of observer error-rates using the em algorithm, Appl. Stat. 28 (1) (1979) 20–28.
[15] Pinar Donmez, Jaime G. Carbonell, Jeff Schneider, Efficiently learning the accuracy of labeling sources for selective sampling, in: KDD, 2009.
[16] Pinar Donmez, Jaime G. Carbonell, Jeff Schneider, A probabilistic framework to learn from multiple annotators with time-varying accuracy, in: SIAM International Conference on Data Mining (SDM), 2010, pp. 826–837.
[17] Arnaud Doucet, Nando de Freitas, Neil Gordon, Sequential Monte Carlo Methods in Practice, Springer, 2001.
[18] Sylvain Gelly, Yizao Wang, Exploration exploitation in go: UCT for Monte Carlo go, in: NIPS On-line Trading of Exploration and Exploitation Workshop, 2006.
[19] David Alan Grier, Error identification and correction in human computation: Lessons from the WPA, in: HCOMP, 2011.
[20] Chien-Ju Ho, Jennifer Wortman Vaughan, Online task assignment in crowdsourcing markets, in: AAAI, 2012.
[21] Leah Hoffmann, Crowd control, Comm. ACM 52 (3) (2009) 16–17.
[22] Eric Huang, Haoqi Zhang, David C. Parkes, Krzysztof Z. Gajos, Yiling Chen, Toward automatic task design: A progress report, in: Proceedings of the ACM SIGKDD Workshop on Human Computation, 2010, pp. 77–85.
[23] Panagiotis G. Ipeirotis, Foster Provost, Jing Wang, Quality management on Amazon Mechanical Turk, in: Proceedings of the ACM SIGKDD Workshop on Human Computation, 2010, pp. 64–67.
[24] Hyun Joon Jung, Matthew Lease, Spam worker filtering and featured-voting based consensus accuracy improvement, in: Proc. of CrowdConf, 2011.
[25] Leslie Pack Kaelbling, Michael L. Littman, Anthony R. Cassandra, Planning and acting in partially observable stochastic domains, Artificial Intelligence 101 (1–2) (1998) 99–134.
[26] Hiroshi Kajino, Yuta Tsuboi, Hisashi Kashima, A convex formulation for learning from crowds, in: AAAI, 2012.
[27] Ece Kamar, Severin Hacker, Eric Horvitz, Combining human and machine intelligence in large-scale crowdsourcing, in: AAMAS, 2012.
[28] David R. Karger, Sewoong Oh, Devavrat Shah, Budget-optimal crowdsourcing using low-rank matrix approximations, in: Allerton, 2011.
[29] Levente Kocsis, Csaba Szepesvári, Bandit based Monte Carlo planning, in: ECML, 2006, pp. 282–293.
[30] Anand Kulkarni, Matthew Can, Bjorn Hartmann, Collaboratively crowdsourcing workflows with Turkomatic, in: Proceedings of CSCW, 2012.
[31] Anand Kulkarni, Philipp Gutheim, Prayag Narula, David Rolnitzky, Tapan Parikh, Bjoern Hartmann Mobileworks, Designing for quality in a managed crowdsourcing architecture, in: HCOMP, 2012.
[32] Walter, S. Lasecki, Kyle I. Murray, Samuel White, Robert C. Miller, Jeffrey P. Bigham, Real-time crowd control of existing interfaces, in: Proceedings of UIST, 2011.
[33] Edith Law, Luis von Ahn, Human Computation. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers, 2011.
[34] Beatrice Liem, Haoqi Zhang, Yiling Chen, An iterative dual pathway structure for speech-to-text transcription, in: HCOMP, 2011.
[35] Christopher H. Lin Mausam, Mausam, Daniel S. Weld, Crowdsourcing control: Moving beyond multiple choice, in: UAI, 2012.
[36] Christopher H. Lin, Mausam, Daniel S. Weld, Dynamically switching between synergistic workflows for crowdsourcing, in: AAAI, 2012.
[37] Greg Little, Lydia B. Chilton, Max Goldman, Robert C. Miller, TurKit: Tools for iterative tasks on Mechanical Turk, in: KDD Workshop on Human Computation, 2009, pp. 29–30.
[38] Omid Madani, Steve Hanks, Anne Condon, On the undecidability of probabilistic planning and related stochastic optimization problems, Artificial Intelligence 147 (1–2) (2003) 5–34.
[39] Mausam, Emmanuel Benazera, Ronen I. Brafman, Nicolas Meuleau, Eric A. Hansen, Planning with continuous resources in stochastic domains, in: IJCAI, 2005, pp. 1244–1251.
[40] Mausam, Andrey Kolobov, Planning with Markov Decision Processes: An AI Perspective, Morgan and Claypool, 2012.
[41] David Milne, Ian H. Witten, Learning to link with Wikipedia, in: Proceedings of the ACM Conference on Information and Knowledge Management, 2008.
[42] John Noronha, Eric Hysen, Haoqi Zhang, Krzysztof Z. Gajos, Platemate: Crowdsourcing nutrition analysis from food photographs, in: UIST, 2011.
[43] Aditya Parameswaran, Hector Garcia-Molina, Hyunjung Park, Neoklis Polyzotis, Aditya Ramesh, Jennifer Widom, Crowdscreen: Algorithms for filtering data with humans, in: VLDB, 2010.
[44] Joelle Pineau, Geoffrey Gordon, Sebastian Thrun, Anytime point-based approximations for large POMDPs, J. Artificial Intelligence Res. 27 (1) (2006) 335–380.
[45] Josep M. Porta, Nikos Vlassis, Matthijs T.J. Spaan, Pascal Poupart, Point-based value iteration for continuous POMDPs, J. Mach. Learn. Res. 7 (2006) 2329–2367.
[46] Lev Ratinov, Dan Roth, Doug Downey, Mike Anderson, Local and global algorithms for disambiguation to Wikipedia, in: Proceedings of the Annual Meeting of the Association of Computational Linguistics, 2011.
[47] Vikas C. Raykar, Shipeng Yu, Linda H. Zhao, Gerardo Valadez, Learning from crowds, J. Mach. Learn. Res. 11 (2010) 1297–1322.
[48] A. Kimball Romney, Susan C. Weller, William H. Batchelder, Culture as consensus: A theory of culture and informant accuracy, Am. Anthropol. 88 (2) (1986) 313–338.
[49] Joel Ross, Lilly Irani, M. Six Silberman, Andrew Zaldivar, Bill Tomlinson, Who are the crowdworkers? Shifting demographics in Mechanical Turk, in: CHI, 2010.
[50] Nicholas Roy, Sebastian Thrun, Coastal navigation with mobile robots, in: Advances in Neural Processing Systems 12, 1999, pp. 1043–1049.
[51] Jeffery Rzeszotarski, Ed Chi, Praveen Paratosh, Peng Dai, And now for something completely different: Introducing micro-breaks into crowdsourcing workflows, in: Submission, 2013.
[52] E. Tjong, Kim Sang, F. De Meulder, Introduction to the CoNNL-2003 shared task: Language-independent named entity recognition, in: Proceedings of CoNNL, 2003.
[53] Dafna Shahaf, Eric Horvitz, Generalized markets for human and machine computation, in: AAAI, 2010.
[54] Guy Shani, Ronen I. Brafman, Solomon Eyal Shimony, Prioritizing point-based POMDP solvers, IEEE Trans. Syst. Man Cybern., Part B, Cybern. 38 (6) (2008) 1592–1605.
[55] Victor, S. Sheng, Foster Provost, Panagiotis G. Ipeirotis, Get another label? Improving data quality and data mining using multiple, noisy labelers, in: Proceedings of the Fourteenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2008.
[56] David Silver, Joel Veness, Monte Carlo planning in large POMDPs, in: NIPS, 2010, pp. 2164–2172.
[57] Trey Smith, Reid G. Simmons, Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic, in: AAAI, 2006.
[58] Rion Snow, Brendan O'Connor, Daniel Jurafsky, Andrew Y. Ng, Cheap and fast – but is it good? Evaluating non-expert annotations for natural language tasks, in: EMNLP, 2008, pp. 254–263.
[59] Edward J. Sondik, The optimal control of partially observable Markov processes, PhD thesis, Stanford, 1971.
[60] Alexander Sorokin, David Forsyth, Utility data annotation with Amazon Mechanical Turk, in: Computer Vision and Pattern Recognition Workshop, 2008, pp. 1–8.
[61] Matthijs T.J. Spaan, Nikos Vlassis, Perseus: Randomized point-based value iteration for POMDPs, J. Artificial Intelligence Res. 24 (2005) 195–220.
[62] Richard S. Sutton, Andrew G. Barto, Reinforcement Learning, The MIT Press, 1998.
[63] Luis von Ahn, Games with a purpose, IEEE Comput. Mag. (June 2006) 96–98.

[64] Fabian L. Wauthier, Michael I. Jordan, Bayesian bias mitigation for crowdsourcing, in: NIPS, 2011.
[65] Daniel S. Mausam, Mausam, Peng Dai, Human intelligence needs artificial intelligence, in: HCOMP, 2011.
[66] Peter Welinder, Steve Branson, Serge Belongie, Pietro Perona, The multidimensional wisdom of crowds, in: NIPS, 2010.
[67] Jacob Whitehill, Paul Ruvolo, Jacob Bergsma, Tingfan Wu, Javier Movellan, Whose vote should count more: Optimal integration of labels from labelers of unknown expertise, in: NIPS, 2009.
[68] Jacob Whitehill, Paul Ruvolo, Tingfan Wu, Jacob Bergsma, Javier Movellan, Whose vote should count more: Optimal integration of labels from labelers of unknown expertise, in: Proc. of NIPS, 2009, pp. 2035–2043.
[69] Yan Yan, Romer Rosales, Glenn Fung, Jennifer G. Dy, Active learning from crowds, in: ICML, 2011.