Dynamically Switching between Synergistic Workflows for Crowdsourcing

Christopher H. Lin Mausam Daniel S. Weld
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195
{chrislin,mausam,weld}@cs.washington.edu

Abstract

To ensure quality results from unreliable crowdsourced workers, task designers often construct complex workflows and aggregate worker responses from redundant runs. Frequently, they experiment with several alternative workflows to accomplish the task, and eventually deploy the one that achieves the best performance during early trials.

Surprisingly, this seemingly natural design paradigm does not achieve the full potential of crowdsourcing. In particular, using a *single* workflow (even the best) to accomplish a task is suboptimal. We show that alternative workflows can compose synergistically to yield much higher quality output. We formalize the insight with a novel probabilistic graphical model. Based on this model, we design and implement AGENTHUNT, a POMDP-based controller that dynamically switches between these workflows to achieve higher returns on investment. Additionally, we design offline and online methods for learning model parameters. Live experiments on Amazon Mechanical Turk demonstrate the superiority of AGENTHUNT for the task of generating NLP training data, yielding up to 50% error reduction and greater net utility compared to previous methods.

Introduction

Crowdsourcing marketplaces (*e.g.*, Amazon Mechanical Turk) enable rapid construction of complex workflows (Little et al. 2009; Bernstein et al. 2010) that seamlessly mix human computation with computer automation to accomplish practical tasks. Example tasks span the range from product categorization and photo tagging to Audio-Visual transcription and interlingual translation. One of the biggest challenges facing designers of crowdsourced applications is the variability of worker quality.

Frequently, a task designer will 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. For example, one set of instructions ("Select all correct statements") might be the negation of another ("Select all incorrect statements.") (Barowy, Berger, and McGregor 2012). For a more involved task like text-improvement, one workflow may present workers with several different improvements of a text and ask them to select the best one. A second workflow might instead present workers with one improvement and ask them to rate it on a scale from 1 to 10. Other examples include different workflows for text translation (Shahaf and Horvitz 2010), alternative interfaces for a task, different ways of wording instructions, and various ways of collecting Natural Language Processing (NLP) tagging data (our use case).

After choosing a single workflow, in order to ensure quality results, task designers often aggregate worker responses on redundant runs of the workflows (Snow et al. 2008; Whitehill et al. 2009; Dai, Mausam, and Weld 2010). For instance, to determine the best text improvement from the results of the second workflow, the task designer might select the one with the highest average rating.

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.

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. For instance, in our example above, if workers are having trouble distinguishing between two improvements, one might prefer future workers to provide absolute ratings.

Instead of selecting one a priori best workflow, a better solution should reason about this potential synergy and *dynamically* switch between different workflows. This paper explains how to do exactly that, making the following contributions:

- We formalize the intuitions underlying workflow-switching with a novel, probabilistic model relating worker responses to the accuracy of workers and the difficulty of alternative workflows for achieving a task.
- We specify the problem of switching between work-

Copyright © 2012, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

flows as a Partially-Observable Markov Decision Process (POMDP), and implement the POMDP policy in our task controller, AGENTHUNT.

- Optimal control requires estimating the parameter values for the POMDP. We describe two unsupervised methods that use an expectation-maximization (EM) algorithm for learning these latent variables: 1) an offline approach, and 2) an online approach that uses reinforcement learning (RL) to couple learning with control (Sutton and Barto 1998).
- We evaluate the benefits of our approach first in a simulated environment and then with live experiments on Amazon Mechanical Turk. We show that AGENTHUNT outperforms the state-of-the-art single-workflow task controller, TURKONTROL, (Dai, Mausam, and Weld 2011), achieving up to 50% error reduction and greater net utility for the task of generating NLP training data. Surprisingly, we also show that our adaptive RL method yields almost as high a utility as the approach requiring an explicit training phase.

By using our system, available at http://cs.washington.edu/node/7714, task designers can combine simple training-free deployment with powerful optimized control.

Probabilistic Model for Multiple Workflows

Many of the commonly employed jobs in crowdsourcing may be modeled abstractly as the task of selecting a single correct answer from a set of alternatives — for example, labeling classification data or choosing the best of two artifacts in an iterative improvement scenario. For this scenario, several previous researchers have designed probabilistic models that combine multiple answers from noisy workers (Sheng, Provost, and Ipeirotis 2008; Whitehill et al. 2009; Dai, Mausam, and Weld 2010).

We follow and extend Dai *et al.*'s probabilistic generative model (2010; 2011), which assumes that the task has 2 answer choices. They define the accuracy of a worker's answer to be: $a(d, \gamma_w) = \frac{1}{2}[1 + (1 - d)^{\gamma_w}]$. Here d is the *inherent difficulty* of the task and γ_w is the *error parameter* of worker w. Our key contribution (Figure 1) is an extension to this model, which allows for the existence of multiple workflows to complete the same task. It includes multiple, workflow-specific error parameters for each worker and workflow-specific difficulties.

For our model, 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.

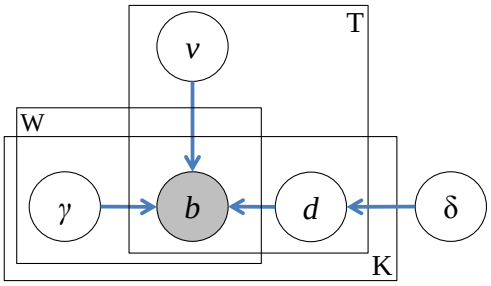

Figure 1: Worker's answer b depends on the difficulty of the question d (generated by δ), worker's error parameter γ and the question's true answer v. There are W workers, who complete T tasks, all of which can be solved using a set of K different workflows. b is the only observed variable.

We rewrite Dai *et al.*'s definition of worker accuracy accordingly:

$$a(d^k, \gamma_w^k) = \frac{1}{2}\left(1 + (1 - d^k)^{\gamma_w^k}\right) \quad (1)$$

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 stated parameters decrease, a approaches 1, when the worker always produces the correct answer.

Figure 1 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:

$$P(b_w^k = v | d^k) = a(d^k, \gamma_w^k) \quad (2)$$

$$P(b_w^k \neq v | d^k) = 1 - a(d^k, \gamma_w^k) \quad (3)$$

An underlying assumption is that given the workflow difficulty ,d^k, the b_w^k's are independent of each other. This is in line with Dai *et al.*'s assumptions for the single workflow case. 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 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.

The availability of multiple workflows with independent difficulties introduces the possibility of dynamically switching between them to obtain the highest accuracy for a given task. We now discuss our approach for taking advantage of this possibility.

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 a POMDP (Sondik 1971; Russell and Norvig 2002), which is a clean and flexible framework for sequential decision making under uncertainty that is able to capture all the assumptions of our problem.

A POMDP is defined as a set of states, a set of actions, a set of observations, a set of state transition probabilities, a set of observation probabilities, and a reward function. In this framework, the agent does not have access to the exact world state. Instead, it can only rely on observations that it receives from the world as it takes actions to infer its current *belief* — a probability distribution over world states. Solving a POMDP entails computing a *policy*, a mapping from beliefs to actions that maximizes its expected long-term reward.

For AGENTHUNT, a state is a $K+1$ tuple $(d^1, d^2, \ldots, d^K, v)$, where d^k is the difficulty of the k^{th} workflow and v is the true answer of the task. Notice that AGENTHUNT can not observe any component of its state. At each time step, AGENTHUNT has a choice of $K+2$ actions — it can submit one of two possible answers or create a new job with any of the K workflows. The process terminates when AGENTHUNT submits any answer. When AGENTHUNT creates a new job using workflow k, it will receive an observation b_w^k containing one of the 2 answers chosen by some worker w. This information allows AGENTHUNT to update its belief using b_w^k and its knowledge of γ_w^k. Figure 2 is a flow-chart of decisions that AGENTHUNT has to take.

None of the actions changes what world-state AGENTHUNT is in; this POMDP is a purely sensing POMDP, so every transition function is the identity map.

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.

In many crowdsourcing platforms, such as Mechanical Turk, we cannot preselect the workers to answer 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 assumes 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 $\overline{\gamma}^k = \frac{1}{W}\sum_w \gamma_w^k$ where W is the number of workers.

After submitting an answer, AGENTHUNT can update its records about all the workers who participated in the task using what it believes to be the correct answer. We follow the approach of Dai *et al.* (2010), using the following update rules: For a worker w who submitted an answer using workflow k, $\gamma_w^k \leftarrow \gamma_w^k - d^k\alpha$, should the worker answer correctly, and $\gamma_w^k \leftarrow \gamma_w^k + (1-d^k)\alpha$, should the worker answer incorrectly, where α is a learning rate. Any worker that AGENTHUNT has not seen previously begins with the average $\overline{\gamma}^k$.

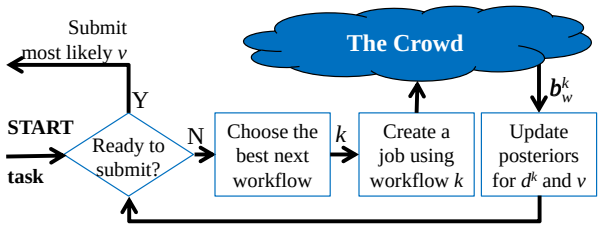

Figure 2: AGENTHUNT's decisions when executing a task.

Learning the Model

In order to behave optimally, AGENTHUNT needs to learn all $\boldsymbol{\gamma}$ values, average worker error parameters $\overline{\boldsymbol{\gamma}}$, 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.

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, $\boldsymbol{b}$. Since the true answer values $\boldsymbol{v}$ are unknown, an option is supervised learning, where experts label true answers and difficulties. This approach was used by Dai *et al.* (2011). But, this is not a scalable option, since it requires significant expert time upfront.

In contrast, we use an EM algorithm, similar to that proposed by Whitehill *et al.* (2009) to learn all parameters jointly. For EM purposes, we simplify the model by removing the joint prior δ, and treat the variables $\mathbf{d}$ 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 $\mathbf{d}$ and $\boldsymbol{\gamma}$:

$$Q(\mathbf{d}, \boldsymbol{\gamma}) = E[\ln p(\mathbf{v}, \mathbf{b}|\mathbf{d}, \boldsymbol{\gamma})] \quad (4)$$

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

After estimating all the hidden parameters, AGENTHUNT can compute $\overline{\gamma}^k$ 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 $\boldsymbol{d}$. 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.

Online Reinforcement Learning

Offline learning of our model can be very expensive, both temporally and monetarily. Moreover, we can not 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, $\text{AGENTHUNT}_{\text{RL}}$, 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 γ and $\boldsymbol{d}$ 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.

Exploration *vs.* Exploitation: 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 (Sutton and Barto 1998). 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.

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, a state-of-the-art controller for optimizing the execution of a single (best) workflow (Dai, Mausam, and Weld 2010), 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$_{RL}$.

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[1] for 300 seconds using the default Focused Real-Time Dynamic Programming search strategy (Smith and Simmons 2006). 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).

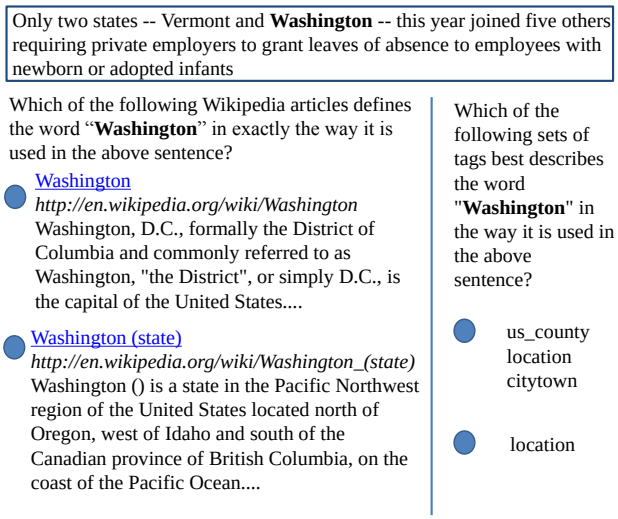

Figure 3: In this NER task, the TagFlow is considerably harder than the WikiFlow since the tags are very similar. The correct tag set is {location} since Washington State is neither a county nor a citytown.

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) (Sang and Meulder 2003). 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.

The Two Workflows In consultation with NER domain experts we develop two workflows for the task (Figure 3). 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* (Milne and Witten 2008; Ratinov et al. 2011) 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[2] tags associated with the Wikipedia article selected by the worker.

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

[2] *www.freebase.com*

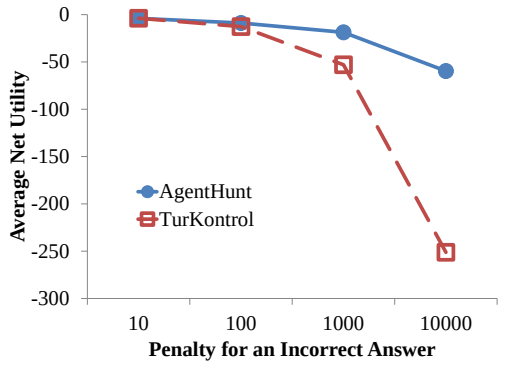

Figure 4: In simulation, as the importance of answer correctness increases, AGENTHUNT outperforms TURKONTROL by an ever-increasing margin.

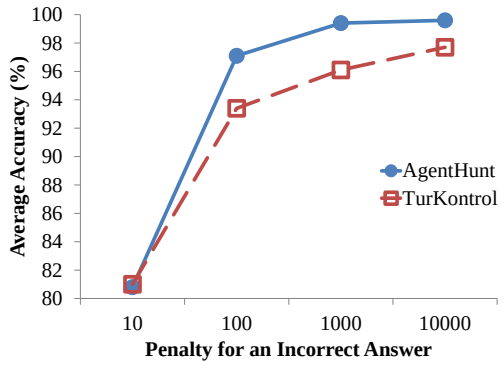

Figure 5: In simulation, as the importance of answer correctness increases, both agents converge to 100 percent accuracy, but AGENTHUNT does so more quickly.

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 {`/film/film`}, while the tags associated with "Richard Nixon" includes {`/people/person`, `/government/us-congressperson`, `/base/crime/lawyer`}. 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.

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, $\overline{\gamma}^{WF}$ and $\overline{\gamma}^{TF}$, corresponding to WikiFlow and TagFlow respectively. Somewhat to our surprise, we find that $\overline{\gamma}^{TF} = 0.538 < \overline{\gamma}^{WF} = 0.547$ — 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 $\overline{\gamma}^{TF}$ and $\overline{\gamma}^{WF}$ controls 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.

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, -1,000, 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 so that the POMDP solver converges quickly. For each setting of reward values we run 1,000 simulations and report mean net utilities (Figure 4).

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 (Figure 5). 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 (Figure 4). 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 $\overline{\gamma}^{TF} < \overline{\gamma}^{WF}$ implies that TagFlows lead to higher worker accuracy on average. Interestingly, although AGENTHUNT has 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 ter-

	AGENTHUNT	TURKONTROL	TURKONTROL_{300}	$\text{AGENTHUNT}_{\text{RL}}$
Avg Accuracy (%)	92.45	85.85	84.91	93.40
Avg Cost	5.81	4.21	6.26	7.25
Avg Net Utility	-13.36	-18.35	-21.35	-13.85

Table 1: Comparisons of accuracies, costs, and net utilities of various agents when run on Mechanical Turk.

minating, but AGENTHUNT does not do too much worse than TURKONTROL, since it correctly deduces that gathering more information is unlikely to help.

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 1 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.

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.

Adding Reinforcement Learning

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

Related Work

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 (Grier 2011). 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.

Modeling repeated labeling in the face of noisy workers has received significant attention. Romney *et al.* (1986) are one of the first to incorporate a worker accuracy model to improve label quality. Sheng *et al.* (2008) explore when it is necessary to get another label for the purpose of machine learning. Raykar *et al.* (2010) propose a model in which the parameters for worker accuracy depend on the true answer. Whitehill *et al.* (2009) and Dai *et al.* (2010) address the concern that worker labels should not be modeled as independent of each other unless given problem difficulty. Welinder *et al.* (2010) design a multidimensional model for workers that takes into account competence, expertise, and annotator bias. Kamar *et al.* (2012) extracts features from the task at hand and use Bayesian Structure Learning to learn the worker response model. Parameswaran *et al.* (2010) conduct a policy search to find an optimal dynamic control policy with respect to constraints like cost or accuracy. Karger *et al.* (2011) develop an algorithm based on low-rank matrix approximation to assign tasks to workers and infer correct answers, and analytically prove the optimality of their algorithm at minimizing a budget given a reliability constraint.

Snow *et al.* (2008) show that for labeling tasks, a small number of Mechanical Turk workers can achieve an accuracy comparable to that of an expert labeler. For more complex tasks, innovative workflows have been designed, for example, an iterative improvement workflow for creating complex artifacts (Little et al. 2009), find-fix-verify for an intelligent editor (Bernstein et al. 2010), iterative dual pathways for speech-to-text transcription (Liem, Zhang, and Chen 2011) and others for counting calories on a food plate (Noronha et al. 2011). Lasecki *et al.* (2011) 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.* (2012) show the crowd itself can help with the design and execution of complex workflows.

An AI agent makes an efficient controller for these crowdsourced workflows. Dai *et al.* (2010; 2011) create a POMDP-based agent to control an iterative improvement workflow. Shahaf and Horvitz (2010) develop a planning-based task allocator to assign subtasks to specific humans or computers with known abilities.

Weld *et al.* (2011) discuss a broad vision for the use of AI techniques in crowdsourcing that includes workflow optimization, interface optimization, workflow selection and

intelligent control for general crowdsourced workflows. Our work reifies their proposal for workflow selection.

Conclusion & Future Work

We demonstrate that alternative workflows can compose synergistically to produce much higher quality results from crowdsourced workers. We design AGENTHUNT, a POMDP-based agent that dynamically switches between these workflows to obtain the best cost-quality tradeoffs. Live experiments on Mechanical Turk demonstrate the effectiveness of AGENTHUNT. At comparable costs, it yields up to 50% error reduction compared to TURKONTROL, a strong baseline agent that uses the best single workflow. Moreover, for a new task, AGENTHUNT can operate out of the box since it does not require any explicit learning phase to tune its parameters. A software package of our implementation is available for general use at http://cs.washington.edu/node/7714.

In the future, we wish to understand the limits of AGENTHUNT by experimenting with tasks that may have many more alternative workflows. We also hope to extend the ability of AGENTHUNT in solving tasks that go beyond choosing one of 2 known alternatives. Another interesting direction for the future is to apply Bayesian Reinforcement Learning (Ross, Chalb-draa, and Pineau 2008), an elegant way to handle the exploration-exploitation tradeoff. We also wish to work on task allocation where we can preselect different workers for different tasks.

Acknowledgements

We thank Xiao Ling for generating our training and testing data. We thank Lydia Chilton, Mitchell Koch, Peng Dai, Rob Miller, Michael Bernstein, Eytan Adar, Haoqi Zhang, Carlos Guestrin, David Alan Grier, and the anonymous reviewers for helpful comments. We thank Trey Smith for making ZMDP available and Jacob Whitehill for making the software for GLAD available. This work was supported by the WRF / TJ Cable Professorship, Office of Naval Research grant N00014-12-1-0211, and National Science Foundation grants IIS 1016713 and IIS 1016465.

References

Barowy, D. W.; Berger, E. D.; and McGregor, A. 2012. Automan: A platform for integrating human-based and digital computation. Technical report, University of Massachusetts, Amherst.

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

Dai, P.; Mausam; and Weld, D. S. 2010. Decision-theoretic control of crowd-sourced workflows. In *AAAI*.

Dai, P.; Mausam; and Weld, D. S. 2011. Artificial intelligence for artificial intelligence. In *AAAI*.

Grier, D. A. 2011. Error identification and correction in human computation: Lessons from the WPA. In *HCOMP*.

Kamar, E.; Hacker, S.; and Horvitz, E. 2012. Combining human and machine intelligence in large-scale crowdsourcing. In *AAMAS*.

Karger, D. R.; Oh, S.; ; and Shah, D. 2011. Budget-optimal crowdsourcing using low-rank matrix approximations. In *Allerton*.

Kulkarni, A.; Can, M.; and Hartmann, B. 2012. Collaboratively crowdsourcing workflows with turkomatic. In *Proceedings of CSCW*.

Lasecki, W. S.; Murray, K. I.; White, S.; Miller, R. C.; and Bigham, J. P. 2011. Real-time crowd control of existing interfaces. In *Proceedings of UIST*.

Liem, B.; Zhang, H.; and Chen, Y. 2011. An iterative dual pathway structure for speech-to-text transcription. In *HCOMP*.

Little, G.; Chilton, L. B.; Goldman, M.; and Miller, R. C. 2009. Turkit: tools for iterative tasks on mechanical turk. In *KDD-HCOMP*, 29–30.

Milne, D., and Witten, I. H. 2008. Learning to link with wikipedia. In *Proceedings of the ACM Conference on Information and Knowledge Management*.

Noronha, J.; Hysen, E.; Zhang, H.; and Gajos, K. Z. 2011. Platemate: Crowdsourcing nutrition analysis from food photographs. In *UIST*.

Parameswaran, A.; Garcia-Molina, H.; Park, H.; Polyzotis, N.; Ramesh, A.; and Widom, J. 2010. Crowdscreen: Algorithms for filtering data with humans. In *VLDB*.

Ratinov, L.; Roth, D.; Downey, D.; and Anderson, M. 2011. Local and global algorithms for disambiguation to wikipedia. In *Proceedings of the Annual Meeting of the Association of Computational Linguistics*.

Raykar, V. C.; Yu, S.; Zhao, L. H.; and Valadez, G. 2010. Learning from crowds. *Journal of Machine Learning Research* 11:1297–1322.

Romney, A. K.; Weller, S. C.; and Batchelder, W. H. 1986. Culture as consensus: A theory of culture and informant accuracy. *American Anthropologist* 88(2):313 – 338.

Ross, S.; Chalb-draa, B.; and Pineau, J. 2008. Bayes-adaptive POMDPs. In *NIPS*.

Russell, S., and Norvig, P. 2002. *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Sang, E. T. K., and Meulder, F. D. 2003. Introduction to the CoNNL-2003 shared task: Language-independent named entity recognition. In *Proceedings of CoNLL*.

Shahaf, D., and Horvitz, E. 2010. Generlized markets for human and machine computation. In *AAAI*.

Sheng, V. S.; Provost, F.; and Ipeirotis, P. G. 2008. 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*.

Smith, T., and Simmons, R. G. 2006. Focused real-time dynamic programming for MDPs: Squeezing more out of a heuristic. In *AAAI*.

Snow, R.; O'Connor, B.; Jurafsky, D.; and Ng, A. Y. 2008. Cheap and fast - but is it good? evaluating non-expert annotations for natural language tasks. In *EMNLP*, 254–263.

Sondik, E. J. 1971. *The Optimal Control of Partially Observable markov Processes*. Ph.D. Dissertation, Stanford.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement Learning*. The MIT Press.

Weld, D. S.; Mausam; and Dai, P. 2011. Human intelligence needs artificial intelligence. In *HCOMP*.

Welinder, P.; Branson, S.; Belongie, S.; and Perona, P. 2010. The multidimensional wisdom of crowds. In *NIPS*.

Whitehill, J.; Ruvolo, P.; Bergsma, J.; Wu, T.; and Movellan, J. 2009. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *NIPS*.