Turkomatic: Automatic, Recursive Task and Workflow Design for Mechanical Turk

Anand Kulkarni[†], Matthew Can[*], Björn Hartmann[*]

[†]Department of Industrial Engineering and Operations Research, [*]Department of Computer Science
anandk@berkeley.edu, matthewcan@berkeley.edu, bjoern@eecs.berkeley.edu

Abstract

On today's human computation systems, designing tasks and workflows is a difficult and labor-intensive process. *Can workers from the crowd be used to help plan workflows?* We explore this question with Turkomatic, a new interface to microwork platforms that uses crowd workers to help plan workflows for complex tasks. Turkomatic uses a general-purpose divide-and-conquer algorithm to solve arbitrary natural-language requests posed by end users. The interface includes a novel *real-time visual workflow editor* that enables requesters to observe and edit workflows while the tasks are being completed. Crowd verification of work and the division of labor among members of the crowd can be handled automatically by Turkomatic, which substantially simplifies the process of using human computation systems. These features enable a novel means of interaction with crowds of online workers to support successful execution of complex work.

Introduction

Crowdsourcing marketplaces like Amazon's Mechanical Turk have demonstrated tremendous utility for batch processing tasks that require human judgment. The vast majority of work carried out on these marketplaces today consists of *microtasks*: cheaply-paid, brief tasks designed to be completed in a few seconds or minutes, *e.g.,* tagging an image or looking up data online. Microtasks typically emerge in the context of *workflows* that split larger tasks into multiple smaller steps and distribute these steps to distinct workers: for example, content generation tasks may separate outlining, writing, and verification steps. However, effective task and workflow design remain something of a black art among crowdsourcing users, involving substantial planning, software development, and testing. The complexity of this process limits participation in crowdsourcing marketplaces to experts willing to invest substantial time in both design and planning; it also limits the kinds of tasks that can be crowdsourced successfully.

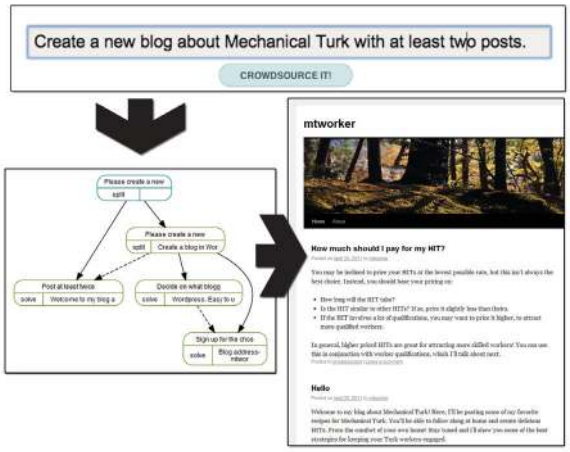

Figure 1: Turkomatic harnesses crowds to plan and execute complex work requested in natural language. A request (top) is subdivided and solved by workers. Turkomatic shows a task graph as work is progressing (left). Workers merge completed subtasks until a global solution is produced (right) – in this case, a new blog.

We propose that the problem of workflow design can be automated by partially delegating the responsibility for designing workflows to the workers themselves. *Turkomatic is a crowdsourcing interface that consults the crowd to design and execute workflows based on user requests.* Our system generates pre-structured Human Intelligence Tasks (HITs) asking Mechanical Turk workers (Turkers) to decompose complex tasks into simpler ones. Other workers solve these tasks sequentially or in parallel and later combine the results into a coherent solution. This process can be recursive, generating multiple decompositions across several steps.

During the development of Turkomatic, we found that the workflows generated by the crowd can often benefit from small, directed modifications by requesters, e.g., if Turkers misunderstood tasks, gave inadequate solutions, or where the requester's original language was unclear. Consequently, we developed a *visual workflow editor* that

enables requesters to manage and control existing workflows while they execute. Requesters can delete or modify plans made by workers, restart task branches launched by workers, or seed the system with partial plans to evaluate their effectiveness at producing desired results.

The resulting system offers both expert and non-expert users of microwork platforms a new way to interface with the crowd for executing complex work. For non-expert users, we expect that Turkomatic will offer a substantially easier way to engage the crowd, since minimal knowledge of HIT or workflow design is required to use these platforms. For expert users, the included visual workflow editor offers an immediate way to control the crowd's execution of a workflow. In both settings, Turkomatic enables requesters to solve complex, high-level tasks more readily than existing interfaces. Following a review of related work, we discuss how Turkomatic operates from the requester and worker perspectives, and describe its algorithmic foundation.

Related Work

Early work in human computation emphasized its utility as a tool for efficiently processing massive datasets in applications like tagging and classification that were outside the reach of autonomous algorithms (von Ahn 2006). Most work on Mechanical Turk today remains batch data processing (Ipeirotis 2010). Quinn and Bederson's taxonomy does not consider any large-scale creative or integrative tasks to be tractable by distributed human computation (Quinn 2011). AI work around Mechanical Turk has emphasized its utility for supporting active learning rather than creative or open-ended problem solving (Sheng 2008, Sorokin 2008).

More recent research has attempted to expand the types of tasks that can be solved via distributed human computation. The TurKit project provides tools for deploying arbitrary *iterative* tasks on Mechanical Turk to enhance quality and eliminate redundant computation (Little 2009, 2010). Follow-up work by Little et al. compares the tradeoffs between iterative and parallel human computation processes (Little 2010). In these investigations, it is assumed that task designers (not workers) will determine how tasks are broken down in all cases. Bigham et al.'s VizWiz is capable of handling open-ended, natural-language requests from its users, but doesn't attempt to parallelize these queries or handle tasks more complex than short requests (Bigham 2010). Bernstein et al. propose a "Find-Fix-Verify" paradigm to divide open-ended work in a manner that maintains consistency and accuracy (Bernstein 2010). Their Soylent system is the first to integrate human computation interactively into a creative process (word processing).

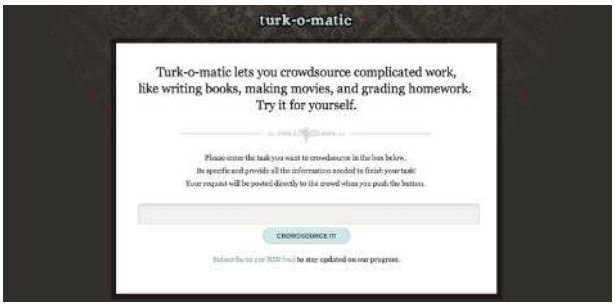

Figure 2: End users request work from Turkomatic through a natural-language interface. By eliminating the requirement for requesters to design HITs, posting work to crowd systems becomes substantially easier.

At present, few tools for managing results from the crowd exist. Mechanical Turk and Crowdflower allow users to export end results as Excel tables, but systems for managing workflows do not exist. While most crowdsourcing tasks are designed manually based on prior experience and intuition, initial work suggests that optimization techniques can play a useful role: by varying HIT parameters programmatically, response quality and response rate can be improved automatically (Huang 2010).

Recently, work-in-progress on CrowdForge introduced a map-reduce paradigm to divide complex work into smaller steps for crowdsourcing platforms (Kittur 2011). We also employ a divide-and-conquer strategy, but introduce a recursive algorithm. In addition Turkomatic adds workflow visualization and editing capabilities not present in CrowdForge.

Scenario: Working with Turkomatic

The following scenarios indicate typical cases of how Turkomatic can be used to crowdsource complex work without and with requester intervention.

Scenario One: Alice is visiting San Francisco and wants to plan a trip through the city. She types the following into the Turkomatic task interface: "Plan a day trip through San Francisco that visits the city's most famous sights." A worker on Turkomatic divides this task into two sub-tasks: 1) choosing the set of locations to visit and 2) planning a route between locations using public transportation. These sub-tasks are solved by two new workers. A fourth worker combines the partial solution into a complete itinerary.

Scenario Two: Bob wants to learn about crowdsourcing. He decides to use Turkomatic to collect information on this topic. Like Alice, he types a request into Turkomatic: "Please write a full encyclopedia article about crowdsourcing, with references". Turkomatic passes the task to workers, who subdivide the task into four sub-tasks: give the definition of crowdsourcing, give examples of

crowdsourcing, give examples of companies in crowdsourcing, and provide links to news articles about crowdsourcing. Bob is skeptical of the crowd's ability to decompose the task, so he consults the Turkomatic workflow editor. Bob realizes that workers are providing examples of companies that *use* crowdsourcing. Instead, he would like to learn about companies that *provide* crowdsourcing services. Using the workflow editor, he changes the subtask description and reissues it to the crowd. The other sub-tasks remain unaffected.

Design and Implementation

Turkomatic comprises three distinct parts. First, a *recursive divide-and-conquer algorithm* to plan work. Second, *worker interfaces* that ask workers to split, solve, merge, and verify work. Third, a *requester user interface* to create, visualize and manage work. We discuss the model and the associated worker interfaces first, then describe Turkomatic from the requester's perspective.

Algorithmic Model

Turkomatic uses a recursive divide-and-conquer algorithm to guide workers through the process of splitting work into smaller subtasks, distributing this work among the crowd and solving it, and recombining the results. This general-purpose algorithm for generating workflows can be interpreted as a *meta-workflow*, which determines when work should be verified by other users or released back to the market. Turkomatic's algorithm for solving work operates in two phases: a subdivision phase that recursively breaks down the problem into smaller components and solves them, followed by a recombination phase where these solutions are merged into a coherent solution.

Subdivide Phase

The "subdivide" phase handles decomposition of tasks and the creation of solution elements. An initial HIT provides a Turker with a task, asking whether or not it can be solved within a given amount of time (ten minutes in our prototype). Based on the worker's answer, Turkomatic will generate a new HIT. If a Turker indicates the task can be solved directly, the new HIT will ask the next Turker to do so (Figure 3b). If the task is judged too complex, the next Turker is asked to break down the task into two or more subtasks that are easier to solve than the original task (Figure 3a). These subtasks are posted again to Mechanical Turk. This process is recursive: the subtasks generated by the subdivide step may themselves be broken down by another subdivide step. To avoid ordering conflicts among the subtasks, the algorithm further asks Turkers to determine if a set of subtasks can be worked on in parallel or whether they must be completed serially.

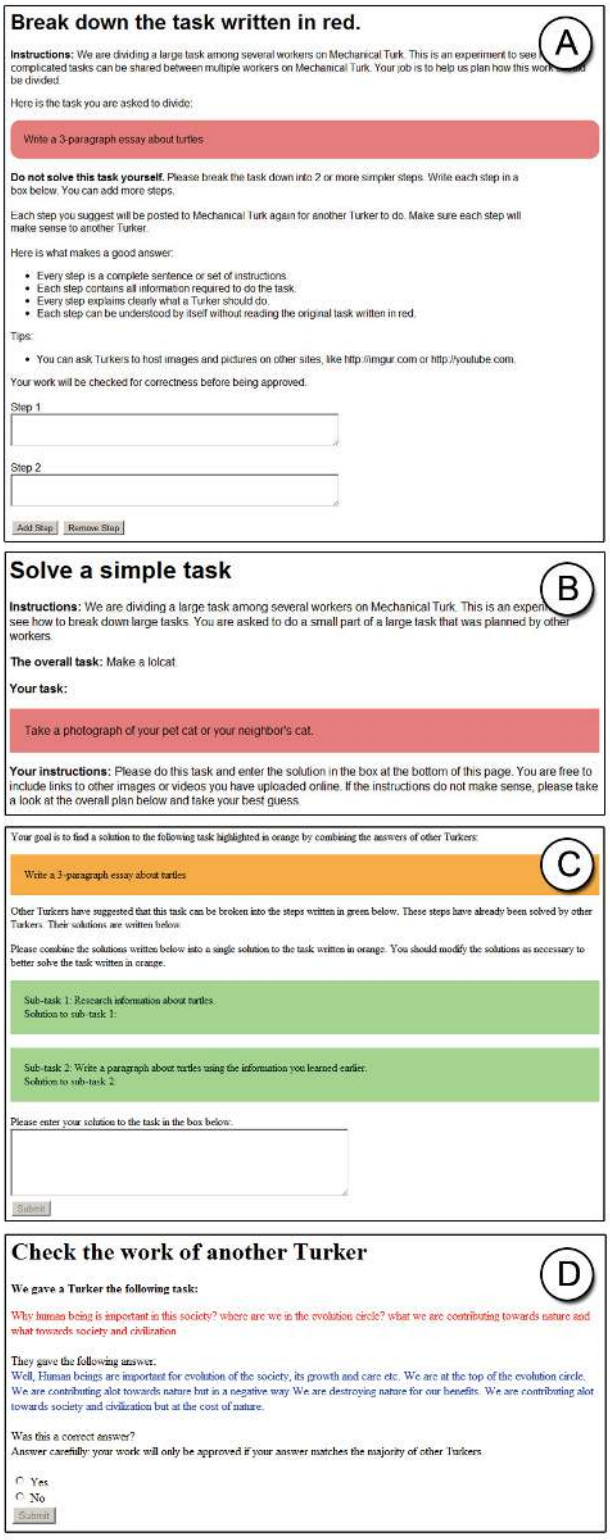

Figure 3. HITs corresponding to (A) subdivision, (B) solution, (C) merging and (D) verification.

Merge Phase

The "merge" step combines solution elements produced during "subdivide" steps into partial solutions to the problem. Once all the subtasks produced in a given subdivide step have been solved, the solutions are listed together in a "merge" HIT (Figure 3c). The HIT instructs a worker to combine the solutions to the subtasks in a way that solves the overall task. The merge process continues until the requester's original task is solved.

Verification

Turkomatic validates the quality of work produced by subdivide, solve, and merge functions by asking the crowd to determine whether an answer is valid. In "verification" HITs (Figure 3d), a single worker is presented with a task and its solution; the worker is asked to verify that the solution is acceptable. In the current version of Turkomatic, a single worker verifies each task. This choice still results in some substandard tasks passing. Future implementations of Turkomatic will utilize redundancy instead, asking multiple Turkers to produce solutions to each HIT and asking other Turkers to vote on the best. Using Turkers to select from redundant work for quality is a well-understood practice (Bernstein 2010, Little 2010).

HIT Design Findings

The biggest challenge in building a task decomposition system such as Turkomatic proved to be designing general-purpose HITs that fit a wide variety of tasks while still conveying specific requirements of the decomposition and reassembly system to a worker. We report several techniques used in the design of HITs for task decomposition:

Show context of tasks in a workflow: Providing workers with a birds-eye view of the overall decomposition proved critical. Workers can easily become confused if their perceived role did not match the one assigned to them. We ultimately included the full decomposition generated by other Turkers into the HIT itself, along with subtask solutions from other workers.

Visually separate prior work: The complex, novel HITs we used to represent *subdivide* and *merge* required substantial time for workers to comprehend. Workers sometimes had difficulty identifying which task in a complex plan they were being asked to carry out. We used strong colors (red, green) and bold text to distinguish between different kinds of information in the HIT (prior work versus specific instructions); such emphasis was more effective than indentation or whitespace. Minimizing the amount of text on-screen increased the likelihood that HITs would be solved as intended.

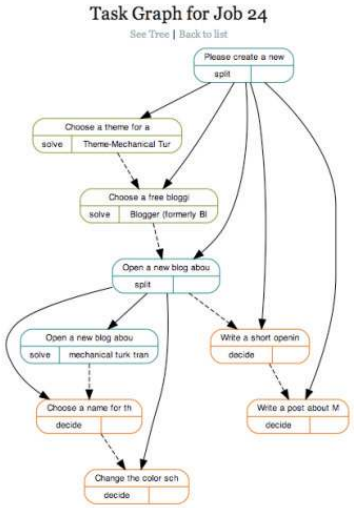

Figure 4: Turkomatic includes a real-time visualization and editing interface. Each node represents a task generated by a crowd worker; the color indicates whether it has been solved or not. Requesters can click to modify either the plan or the solutions to any step to eliminate worker errors or search for alternate solutions.

Requester Interfaces

This section reviews interaction with Turkomatic from the requester's perspective.

Requesting Work

Requesters post new Turkomatic jobs through a natural-language web interface (Figure 2). Inspired by web search, it offers a text box where the requesters specify what they want to accomplish.

Expert requesters can also author decompositions that serve as a starting point. Such decompositions may be preferred if the requester believes that crowds will not be able to properly decompose a given task (*e.g.*, because the decomposition requires significant domain expertise). A decomposition specifies a tree structure where the leaves of the tree are the tasks that have to either be solved or further sub-divided. In the current prototype, decompositions have to be authored in code; future work could provide a visual editor for this task. Once a new task prompt or task tree is loaded, the Turkomatic algorithm takes over to request further subdivisions, solutions, and merge steps.

Visualizing Ongoing Work

To enable requesters to gain insight into partially completed work, Turkomatic provides a workflow visualization that shows the current state of an ongoing job as a decomposition tree (Figure 4). Such a visualization can inform requesters how much of the work has been accomplished, what strategies have been taken, and whether subtask solutions or decompositions are of sufficient quality.

Turkomatic uses GraphViz[1] to render workflows as node-link diagrams. Nodes in the graph represent the component tasks of a job issued to Turkomatic. Directed edges denote the relationship between tasks and their subtasks (i.e. when there is a subdivision). Dashed directed edges indicate the order of tasks in a serial split, where one sibling must complete before another sibling can be posted. Each node contains a summary of the task prompt, the solution to the prompt (if already available), and a status indicator. A task can either be a) *waiting on a decision* whether to split or solve (orange in Fig 4); b) *in-progress* and waiting for subtasks to complete (cyan); or c) *solved* (green). The visualization is interactive: brushing over nodes displays complete instructions and solutions in a floating panel, as it often cannot fit into the node itself.

Editing Workflows

Ongoing work can be unsatisfactory for multiple reasons: a crowd-authored decomposition may be flawed, or a solution to a subtask may be of low quality. Requesters can edit existing workflows in real-time to address such challenges. Requesters can choose to either edit the task description or the solution for any node in the workflow. Once an edit is performed, Turkomatic computes which subsets of tasks will have to be performed again by additional workers.

When a task description is changed, any subtasks created for this task by the crowd may also no longer be valid. Turkomatic therefore invalidates the entire subtree below the edited task and reissues the task. Subsequent siblings in serial decompositions also have to be recomputed. Finally, if the task already had a solution, all upstream solutions of parents that used this stale information have to be recomputed.

When a requester edits a task solution directly, the entire worker-generated subtree of that task is discarded. Tasks in this subtree that are *in flight* (currently being worked on) will also be discarded. As in the task instruction case, serial siblings and solutions of parents are invalidated as well.

Experimenting with Turkomatic

To explore how effectively crowds can be used to support the execution of complex work, we examined how well Turkomatic's *merge* and *decomposition* steps performed both with and without expert intervention. We examined tasks that push the boundary of current single-user microwork: 1) complex content creation, specifically on-demand essay writing; 2) searching and integrating information to answer a natural-language query; 3) creating and populating a blog. For each task, we examined how successful Turkomatic was under the following scenarios:

1 planning and executing natural-language tasks without requester intervention (fully automatic mode),
2 using requester intervention to modify erroneous work by Turkers (Turkomatic as crowd support tool), and
3 using crowds to solve prepopulated workflows generated by workers (Turkomatic as an execution interface).

To isolate our results from the impact of Mechanical-Turk-specific design choices such as pricing, we also executed these tasks in a separate experiment with a pool of expert Mechanical Turk users at our university. We priced HITs at 20 cents per subdivision, merge, or execution HIT, and 5 cents per verification task.

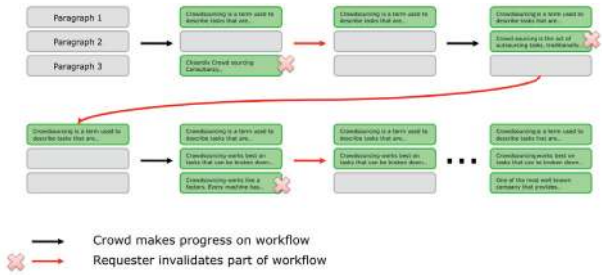

Figure 5: Requesters can intervene repeatedly to increase the quality of the work. Different paragraphs of a three-paragraph essay are independently written by workers; some work is not acceptable to the requester, so he reissues subtasks.

Task instructions	Condition	Outcome
Create a list of the names of the Department Chairs of the top 20 computer science college programs in the US (each school has 1 Department Chair)	Workers only	Failed to complete
	Requester intervention	N/A
	Requester seeding	Completed with 3 interventions
	Expert workers	Completed without intervention
Write a 3-paragraph essay about crowdsourcing	Workers only	Failed to complete
	Requester intervention	N/A
	Requester seeding	Completed with 4 interventions
	Expert workers	Completed without intervention
Please create a new blog about Mechanical Turk, with a post and a comment on that post.	Workers only	Failed to complete
	Requester intervention	Failed to complete
	Requester seeding	N/A
	Expert workers	Completed with 1 intervention

Table 1: Results from Turkomatic Experiments. N/A: not attempted.

Results

Table 1 summarizes the results we obtained: end-to-end tasks succeeded when expert workers carried out decomposition and solution of the sub tasks. In the fully-

[1] http://graphviz.org

automatic condition, some decompositions and solutions were of high quality, but tasks remained only partially completed, for two major reasons. First, some tasks were *starved* – after a while, no new workers attempted the available tasks. This occurred most often when a worker marked a task as solvable when, judging by the complexity, it should have been subdivided. Task starvation has been observed in other projects and can be counteracted through listing optimization. Second, some tasks were *derailed*: Either a worker-provided task description or a solution was of sufficiently poor quality that subsequent workers could not recover. When requesters intervened using the workflow editor, these tasks eventually succeeded. Figure 5 shows an example of the essay writing task where repeated intervention eventually led to a good solution.

The full content created by our workers is available at http://husk.eecs.berkeley.edu/projects/turkomatic/hcomp11

Discussion

The utility of Turkomatic's requester interface is apparent: work generally completed successfully with requester intervention. When unusable results arrived, requesters were able to rapidly iterate by modifying local structure without redeploying an entire workflow. But why did tasks without expert users or requester intervention starve or derail? We discuss several possible explanations.

Crowd Verification Underperformed

The current design uses a single crowd worker to verify each division or solution step. We found numerous instances where these verifications failed – bad work was accepted. The verifier may lack sufficient context to judge quality. We believe that having the verifier choose between multiple possible alternative solutions is a more promising strategy as it allows comparison and contrast.

Instructional Writing is Uncommon for Workers

We noticed that expert workers provided more detailed instructions in their subdivisions; they were also more careful to ensure subtasks were self-contained and did not require reference to parent tasks. Composing good instructions is not trivial and takes time. This requirement stands in tension with the goal of workers to maximize the number of tasks they complete per unit of time. While we observed that some workers were able to write excellent instructions, the majority of other tasks on Mechanical Turk do not require such careful attention, and workers may thus be disincentivized from delivering nuanced work.

Reputational Effects

Turkomatic assigns responsibility for wording tasks to the workers. Interestingly, this means that bad choices by Turkers may negatively affect the requester's reputation. We noticed that workers on Turker Nation, a discussion forum for workers, posted messages about tasks created by Turkomatic, complaining about poor wording or excessive scope. Workers were not aware that these tasks were in fact created by other workers and assigned their dissatisfaction to the requester.

Conclusion and Future Work

Turkomatic's primary advantages appear to lie in both its generality and its flexibility – crowd-produced workflows and generic HITs can handle an interesting range of tasks. However, this one-size-fits-all model trades off ease-of-specification for required runtime supervision: Workflows can be generated without exhaustive planning, but require some requester intervention at runtime to guarantee quality of results. In future work, we plan to investigate to what extent this supervisory function can again be assigned to crowd workers.

Turkomatic's crowd-assisted workflow generation also makes no guarantees for optimality in pricing or execution time. The question of *optimizing* crowd-produced workflows is fascinating and merits future investigation.

References

Bernstein, M., Little, G., Miller, R.C., Hartmann, B., Ackerman, M., Karger, D.R., Crowell, D., and Panovich, K. Soylent: A Word Processor with a Crowd Inside. *UIST 2010*.

Bigham, J.P., Jayant, C., Ji, H., Little, G., Miller, A., Miller, R.C., Miller, R., Tatrowicz, A., White, B., White, S., and Yeh, T. VizWiz: Nearly Real-time Answers to Visual Questions. *UIST 2010*.

Huang, E., Zhang, H., Parkes, D.C., Gajos, K.Z., and Chen, Y. Toward automatic task design: A progress report. *HCOMP 2010*.

Ipeirotis, P. Analyzing the Amazon Mechanical Turk Marketplace. Working paper, http://hdl.handle.net/2451/29801. 2010.

Kittur, A., Smus, B., and Kraut, R. CrowdForge: Crowdsourcing Complex Work. *CHI 2011 Work-in-Progress.*

Little, G., Chilton, L.B., Goldman, M., and Miller, R.C. TurKit: Tools for Iterative Tasks on Mechanical Turk. *HCOMP 2009*.

Little, G., Chilton, L.B., Goldman, M., and Miller, R.C. TurKit: Human Computation Algorithms on Mechanical Turk. *UIST 2010*.

Little, G., Chilton, L.B., Goldman, M., and Miller, R.C. Exploring Iterative and Parallel Human Computation Processes. *HCOMP 2010*.

Quinn, A.J. and Bederson, B.B. Human Computation: A Survey and Taxonomy of a Growing Field. *CHI 2011*.

Sheng, V.S, Provost, F., and Ipeirotis, P.G. Get Another Label? Improving Data Quality and Data Mining Using Multiple, Noisy Labelers. *KDD 2008*.

Sorokin, A. and Forsyth, D. Utility data annotation with Amazon Mechanical Turk. *CVPRW 2008*.

von Ahn, L. Games with a Purpose. *IEEE Computer* (2006).