Turkomatic: Automatic Recursive Task Design for Mechanical Turk

Abstract
Most problems cannot be crowdsourced today without investing significant effort into task design. We investigate how to use divide-and-conquer strategies to automate the process of task design while enabling the solution of high-level and complex tasks. Unlike all existing approaches to automating task design, our strategy is recursive, asking individual workers on a crowdsourcing platform to help plan out how problems can be solved most effectively. Our experiments so far show that this strategy is successful in decomposing and solving tasks considered difficult for AI. A secondary contribution of our work is a significantly simpler interface to a popular crowdsourcing platform that abstracts away almost all task design and worker interaction; informal evaluations found this strategy to be substantially faster and easier to use for work requesters than the existing interface.

Keywords
Human computation, crowdsourcing, distributed computation, Mechanical Turk

ACM Classification Keywords
H.5.m. [Information interfaces and presentation]: Miscellaneous.

Anand Kulkarni
UC Berkeley
Department of IEOR
4141 Etcheverry Hall
Berkeley, CA 94720-1777
anandk@berkeley.edu

Matthew Can
UC Berkeley
Computer Science Department
346 Soda Hall
Berkeley, CA 94720-1776
mattwcan@berkeley.edu

Figure 1. Turkomatic automatically produces workflows for complex tasks on Mechanical Turk. Above, results from an automatic essay-generation task given to Turkomatic, “Please write a short five-paragraph essay on a topic of your choice.” The remainder of the essay is not shown to save space.
Introduction
Designing effective tasks on crowdsourcing systems like Mechanical Turk is currently a black art. It is presently unknown how to convert the kind of general, high-level tasks we do every day ("write a paper about ____; build a webpage containing ____") into tasks that can be solved effectively on crowdsourcing systems. We propose that the problem of high-level task design can be solved recursively on Mechanical Turk. We can generate Human Intelligence Tasks (HITs) that ask Mechanical Turk workers (Turkers) to decompose complex tasks into simpler ones, solve these tasks in parallel, and then piece together the results into a coherent solution.

We hypothesize that a recursive, distributed framework for crowdsourcing will be able to solve tasks that are currently considered beyond the scope existing platforms for human computation.

Distributed frameworks have been successful in traditional computing contexts at automatically reducing complex problems into forms that can be solved efficiently in parallel. We believe that an analogous process for parallelizing human computation may be able to decompose tasks into atomic pieces that individual workers are willing to carry out.

Related Work
Early work in human computation emphasized human computation as a tool for efficiently processing massive datasets in applications like tagging and classification that were outside the reach of autonomous algorithms [12,13]. The majority of work that is placed on Mechanical Turk today remains batch data processing [5]; one taxonomy prepared by Quinn and Bederson does not identify any large-scale creative or integrative tasks tractable by distributed human computation [9]. AI work around Mechanical Turk has emphasized its utility for supporting active learning rather than creative problem solving [10,11].

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 [6,7]. Follow-up work by Little et al compares the tradeoffs between iterative and parallel human computation processes [8]. However, these investigations using TurKit don't consider recursive decomposition in task design -- they assume that task designers 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 [2]. Bernstein et al [1] propose a paradigm ("Find-Fix-Verify") to divide open-ended work between workers in a manner that maintains consistency and accuracy, and is the first to integrate human computation into a creative and interactive process; however, these authors apply the work in the limited context of word processing. Huang et al [4] suggest automatically varying HIT parameters to optimize worker responses.

System Description
Our system receives a description of a general task from the end user, along with the user's email address. The system solves the task by asking Turkers to repeatedly break it down into smaller subtasks, to then
solve these manageable pieces, and to finally recombine the chunks into a coherent solution.

Our algorithm for recursively solving work is analogous to the MapReduce computing paradigm used for large-scale parallel distributed computation [3]. In MapReduce, a “map” function produces intermediate solutions to a problem by distributing it over several computational nodes, and a “reduce” function combines the intermediate solutions into a final solution. We describe the analogous steps in our algorithm.

**Map Step**
The “map” step handles decomposition of tasks and design of solutions. It provides a Turker with a task, asking whether or not it can be solved within ten minutes. If a Turker indicates it can be solved, then he or she is asked to solve it. If not, the Turker is asked to break down the task into two or more logical subtasks that are easier to solve than the original task. This process is recursive; the subtasks generated by the map step may themselves be broken down by another map step. We validate the quality of output produced by our map function through redundancy, asking two or more Turkers to produce separate candidate solutions to each HIT and asking a separate pool of Turkers to vote on the best. Because the process of having Turkers select from redundant work for quality is a well-understood practice [7,1,8], the voting step was simulated by the authors in the presented experiments, or if appropriate, left to workers in the reduce step.

**Reduce Step**
The reduce step is invoked after a decomposition is produced by the map step. Once all the subtasks for this task have been solved, they are passed as input to the reduce step, where a Turker uses them to solve the task itself. The Turker is specifically instructed to combine the solutions to the subtasks in a way that solves the overall task. The reduce process is invoked continuously on any node whose subtasks are complete until the end user’s original task is solved.

**Evaluation**
*Can Automatic Task Design Solve Hard Problems?*
There are no general, task-agnostic methods for producing distributed workflows on Mechanical Turk today. Existing processes require the active intervention of a human task designer. We sought to show that our algorithm can produce generic workflows for examples of high-level tasks posed in natural language. We examined the following two tasks:

--Producing a written essay in response to a prompt
--Solving an example of an SAT test uploaded by a user

These two tasks were chosen because they would be difficult for existing autonomous algorithms to solve and are not tasks currently considered to be within the scope of existing human computation platforms.

**Method**
We recruited five users to act as proxies for remote Turk users in a pilot study on the Mechanical Turk Worker Sandbox. After incorporating the feedback they provided, we refined the language of our HITs and posted these tasks live on the Mechanical Turk site using our interface.
Results

Turkomatic was able to successfully solve both tasks. The decompositions provided by Turkers are displayed at left in Figure 4 and above in Figure 1 along with the scores by section and a portion of the essay produced.

The best decomposition produced for the SAT task used only one level of recursion. Workers divided the task into 8 subtasks consisting of two or three thematically linked questions. These were each solved in parallel by two workers per subtask and the results were given to a reduce worker who produced the final solution. The score on the overall solution was 12/17, with the worst performance on math and grammar questions and the best in reading and vocabulary. While obtaining a useful decomposition proved tricky for workers – many seemed confused about the nature of the planning task – once the tasks were decomposed, solution of the constituent parts and reassembly into an overall solution were straightforward for Turkers to accomplish.

In the essay task, each map HIT was posted three times by Turkomatic and the best of the three was selected by experimenters (simulated voting) to continue the solution process. The proposed decompositions were overwhelmingly linear and chose to break the task down either by paragraph or by activity (brainstorm, create outline, write topic sentences, fill in facts). The decomposition used in the final essay chose to use two levels of recursion, shown at the start of the paper. As groups of subtasks were completed, Turkomatic passed solutions to reduce workers for reassembly. The resulting essay is complete and coherent, although somewhat lacking in cohesion.

We allowed essay-writers to pick a topic; the chosen topic was somewhat specialized, but the final essay displayed a reasonably good understanding of the topic, even if the writing quality was often mixed.

Informal evaluation of our interface

As a second, informal study, we examined whether reducing user involvement in the HIT design improved ease of use and efficiency. We hypothesized that the high level of abstraction enabled by automatic task design would make it easier for requesters to crowdsource their work.

We asked a pool of four users to try to carry a basic brainstorming task on Mechanical Turk, first, using Turkomatic to post tasks and obtain results, then, using Mechanical Turk’s web interface. The task was, “Generate five ideas of topics for an essay.” No instruction on either interface was provided. We examined how long it took the user to post the task. We found that reducing the level of user involvement in task design provides a striking improvement; with Turkomatic, our users finished posting their tasks in an average of 37 seconds. However, when low-level task design was required, users needed an average of 244.2 seconds to post their tasks. Crucially, the two users who were not previously familiar with Mechanical Turk failed to post HITs that would produce any meaningful results for the task.

We allowed essay-writers to pick a topic; the chosen topic was somewhat specialized, but the final essay displayed a reasonably good understanding of the topic, even if the writing quality was often mixed.

Informal evaluation of our interface

As a second, informal study, we examined whether reducing user involvement in the HIT design improved ease of use and efficiency. We hypothesized that the high level of abstraction enabled by automatic task design would make it easier for requesters to crowdsource their work.

We asked a pool of four users to try to carry a basic brainstorming task on Mechanical Turk, first, using Turkomatic to post tasks and obtain results, then, using Mechanical Turk’s web interface. The task was, “Generate five ideas of topics for an essay.” No instruction on either interface was provided. We examined how long it took the user to post the task. We found that reducing the level of user involvement in task design provides a striking improvement; with Turkomatic, our users finished posting their tasks in an average of 37 seconds. However, when low-level task design was required, users needed an average of 244.2 seconds to post their tasks. Crucially, the two users who were not previously familiar with Mechanical Turk failed to post HITs that would produce any meaningful results for the task. One user posted minor variations of the default templates provided on Mechanical Turk and the other incorrectly concluded he had posted the task after only creating a HIT template.

The user pool gave generally positive responses when asked whether Turkomatic was easier to use than
Mechanical Turk. The two users with prior Mechanical Turk experience indicated that the lack of detailed control offered by the abstract interface was somewhat worrying, but the two novice users preferred the more abstract representation unconditionally.

Discussion
Design Implications
The biggest challenge in building a task decomposition system such as Turkomatic proved to be designing HITs that could handle arbitrary, general tasks while still conveying specific requirements of the decomposition / reassembly system to a worker. We report several findings from our experiences and experiments that apply generally to the design of task decompositions systems.

- **Show context:** Providing workers with a birds-eye view of the overall decomposition was critical. Workers often used different cognitive models in planning how tasks were decomposed (in particular, serial versus parallel decompositions of tasks), and could easily be confused if their perceived role did not match the one assigned to them. We provided workers with an overall visualization on how the task was being decomposed and solved by other workers, significantly reducing worker error.

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

- **Use the best workers first and last:** We found the quality of the overall product and the nature of the overall workflow was strongly influenced by the decompositions produced by first workers. While leaf nodes in the solution tree were successfully solved by a wide variety of workers, many workers didn’t seem to understand the requirements of a planning task. Selecting for higher-quality workers early on dramatically improved the quality of the final result. Higher-quality reduce workers were willing to correct errors even in sections not assigned to them, giving the greatest impact if used in the final reduce steps.

- **Make atomicity salient:** We used the condition “can this task be completed in under ten minutes?” as a means to stop recursion and produce solutions. This had mixed success. Time tended to be ignored by workers in deciding if they would recurse or solve, and many workers were confused by the requirement.

- **Price adequately, not excessively:** The relationship between price, quality, and task completion time is complex and poorly understood; we found workers were willing to do our HITs for as little as five cents and as much as forty cents without a consistent difference in quality or speed of completion, which was highly variable.

- **Require equal task size:** In our early experiments Turkers often chose to break off small, minimal pieces as a first step rather than providing decompositions that were easy to parallelize. In subsequent HITs, we explicitly asked workers to break up tasks into roughly equal-sized chunks.
**Future Work**

Our next steps involve development of extended, dynamic interaction between the requester and the Turkomatic system. We expect these kinds of interaction can enable immediate access to crowdsourcing for nontechnical users as a productivity support tool, as well as improving the quality of tasks produced and increasing the range of tasks solvable within the system.

The next version of our system will operate as follows. At various points during execution, the requester will receive emails from our system asking for clarification on subtasks or to make a choice between several suggested decompositions or subtask solutions; the user’s answers can be incorporated into the HITs seen by Turkers. Similarly, it would be useful to let Turkers make queries directly back to the original requester in cases where a task is vague due to requester or Turker error.

Given that Turkomatic is capable of producing solutions to tasks considered conventionally difficult to crowdsourcera, it will also be interesting to show that Turkomatic offers either a cost, quality, or speed advantage over traditional methods of getting work done. We expect that these evaluations, in conjunction with solving an expanded range of tasks through the system, demonstrate that recursive interfaces offer strong potential as a new approach to crowdsourcing.

**References**


