Abstract
Most problems cannot be crowdsourced today without investing significant effort into task design. We investigate how to use divide-and-conquer strategies to automatically solve high-level 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 early experiments show that this strategy is successful in solving tasks considered difficult for AI systems [not yet shown conclusively at the time this draft was written], but that the quality of results is highly dependent on the language of the task. 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 dramatically faster and more popular among work requesters than the existing interface [not yet shown conclusively at the time this draft was written].

Keywords
Human computation, crowdsourcing, distributed computation, Mechanical Turk
**ACM Classification Keywords**
H.5.m. [Information interfaces and presentation]: Miscellaneous.

**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. Much of the value added by companies in the crowdsourcing space comes from converting high-level tasks into a format that can be crowdsourced. 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 of 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 [3,4]. The majority of work that is placed on Mechanical Turk today remains batch data processing [23]; one taxonomy prepared by Quinn and Bederson doesn’t identify any large-scale creative or integrative tasks tractable by distributed human computation [1]. AI work around Mechanical Turk has emphasized its utility for supporting active learning rather than creative problem solving [5,6].

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 [15,16]. However, investigations using TurKit don't consider recursion or 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 [18]. Bernstein et al [17] 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 or interactive process; however, these authors apply the work in the limited context of word processing. Huang et al [12] suggest automatically varying HIT parameters to optimize worker responses.
In parallel computing, the divide-and-conquer MapReduce algorithm forms the best analogue to our work [13].

**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.

![Welcome to Turkomatic!](image)

Our human algorithm is most similar to the MapReduce computing paradigm used for large-scale parallel, distributed computation. In MapReduce, the map function produces intermediate solutions to a problem. The reduce function combines the intermediate solutions into a final solution. We now describe the analogous steps in our algorithm.

**Map Step**
Task decomposition will work as follows. Our algorithm will provide a Turker with a task, asking whether or not it can be solved within two minutes. If a Turker indicates it can be solved, then he or she will be asked to solve it. If not, the Turker will be asked to break down the task into two or more logical subtasks that are easier to solve than the original task.

In our final system, we intend to validate the quality of output produced by our map function. We will do this by asking three Turkers to perform a particular map step, generating three candidate decompositions. Then we will ask four Turkers to vote on the best decomposition. The winning decomposition will be the output of this map step.

The process we have described is recursive; the subtasks generated by the map step may themselves be broken down by another map process.

**Reduce Step**
Let us take a particular task and its decomposition as determined 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 task.

As with the map step, our final system will validate the quality of the reduce step. Three Turkers will carry out the same reduce step, and other Turkers will vote on the reduction that best solves the task. The reduce function will return the winning solution for further reduction until the end users original task is solved.

**Sample Scenario**
Suppose Jim wants to write an essay about why Napoleon lost the Battle of Waterloo. He enters in a text box on our system the following task: "Write an essay explaining why Napoleon lost the Battle of Waterloo."
He specifies three other parameters: a desired task speed, whether he would like to participate in the process interactively, and his email address. He sets his desired speed to "one day". A faster speed increases the pay assigned to individual workers to speed production of results. He chooses not to participate interactively -- if he had, he would receive frequent emails asking to choose between various decompositions of tasks. In the next several hours, he receives occasional emails produced by Turkers and our system asking for clarification on minor points. He responds to these and they are incorporated into the HITs produced by our system.

One day after submitting the original task, Jim receives his solution by email: a complete, well-written essay on the topic of his choice. Attached is a diagram indicating the entire decomposition of his task into subtasks, and the various decisions that were made to reconstruct his essay. Note: not all features of the system described in this scenario have been implemented.

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 distributed human computation.

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 Mechanical Turk site using our interface.

Results
We were able to successfully solve both tasks. The decompositions provided by remote users are displayed at left. [partly real data, partly placeholder data]

Several early examples indicated that the effectiveness of the algorithm depends strongly on the structure of the MAP and REDUCE HITs.

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

Some sidebar data
(this callout is dummy text)
Evaluate long algebraic formulas at a given value (easy computing task but difficult to crowdsource accurately, demonstrates increased accuracy from parallelized crowdsourcing)
Transcribe these 200 audio files (conventional crowdsourcing task that is not automatically parallelized, difficult AI task)
Write a homework assignment for an algebra 2 class (difficult AI task)
Generate perfect solutions to this SAT exam (difficult AI task)
Write a short story with a twist ending (tests integration of results between multiple users, “AI-Complete” task)

SAT task [still going]
Show output, explain what does not work, what worked, what did not, show the decomposition the Turkers came up with, how much did it cost, how long did it take.

Essay writing task [still going]
Show output, explain what does not work, what worked, what did not, show the decomposition the Turkers came up with, how much did it cost, how long did it take.
Informal evaluation of our interface [expected data]
As a second, informal study, we examined whether reducing user involvement in the HIT design improved ease of use and efficiency without affecting quality. 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 five 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. No instruction on either interface was provided. We examined how long it took the user to post the task and how long it took them to obtain results. Following this, we asked each user to evaluate the quality of results obtained by each method and which interface they preferred.

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 60 seconds and received their results in an average of 30 minutes. However, when low-level task design was required, three out of five users were unable to post their tasks at all within the allotted 20 minute session and the rest required at least ten minutes; these two received results an average of three hours later.

The five users surveyed gave a uniformly positive response (7) to the question of whether they preferred the abstract interface versus the low-level interface and indicated that the two systems produced equivalent quality results (4/7).

Discussion
Serial v. parallel
We found workers overwhelmingly gravitated towards a serial representation of tasks versus a parallel representation of tasks. Unless asked to do otherwise...

Equal task size
In our early experiments Turkers often chose to break off small, minimal steps of their decomposition as a first step and post the rest, rather than providing decompositions that were easy to parallelize. This approach fails to take advantage of the parallel potential of Turk. To control for this factor, we constructed HITs that asked workers to break up tasks into roughly equal-sized chunks. Turkers took these requirements seriously, producing work that was equally-sized. This is an important condition...

Does our system do what we said it would
Comments on Informal user eval of our interface
Comments on how our distributed system compared to a single crowdworker
Branching factor...
We’ll vary the branching factor that determines how many subtasks should be generated for each task (two, three, or as many as the Turker deems necessary) to see how this affects completion speed.

Consequences
Our study told us whether or not our system is capable of solving the kinds of complex tasks that are difficult to solve with the standard crowdsourcing approach.
These tasks are considered difficult and beyond the capability of existing crowdsourcing systems, so a simple demonstration of their capacity will suffice. Beyond this, comparison of the quality of the results with those produced in a non-distributed manner is an appropriate means of determining if the approach is superior.

Discussion of implications for applications
Foo. Applying our method to several applications.

Future Work
A natural extension of this work is in the creation of productivity support tools. It will be important to show that these systems offer either a cost, quality, or speed advantage over traditional methods of getting this work done. For our two target tasks, we will conduct an objective quality evaluation as follows. For the SAT task, we will grade the solution produced by the workers. For the essay writing task, we can use the Flesch–Kincaid grade level to objectively measure writing quality. For a more holistic quality evaluation in the latter task, we’ll ask Turkers to examine how well the essays produced by our system compare to essays produced by a single Turk user. We obtain comparative speed and cost assessments by posting these HITs to Mechanical Turk alongside conventional HITs where we ask a single user to carry out the task. It is likely that for many of the tasks we propose, a single user will not be willing to carry out the task for less than a substantial amount of money.

Here are features we plan to implement in future versions of our system:

- At various points during execution, the user may receive emails from our system asking for clarification on subtasks or to make a choice between two components; the user’s answers are incorporated into the HITs seen by Turkers.
- Allow map Turkers to specify an ordering on the sub tasks they create when they are serial.
- Allow the user to upload files as part of the task description.

References
[8] Seth Cooper, Firas Khatib, Adrien Treuille, Janos Barbero, Jeehyung Lee, Michael Beenen, Andrew


