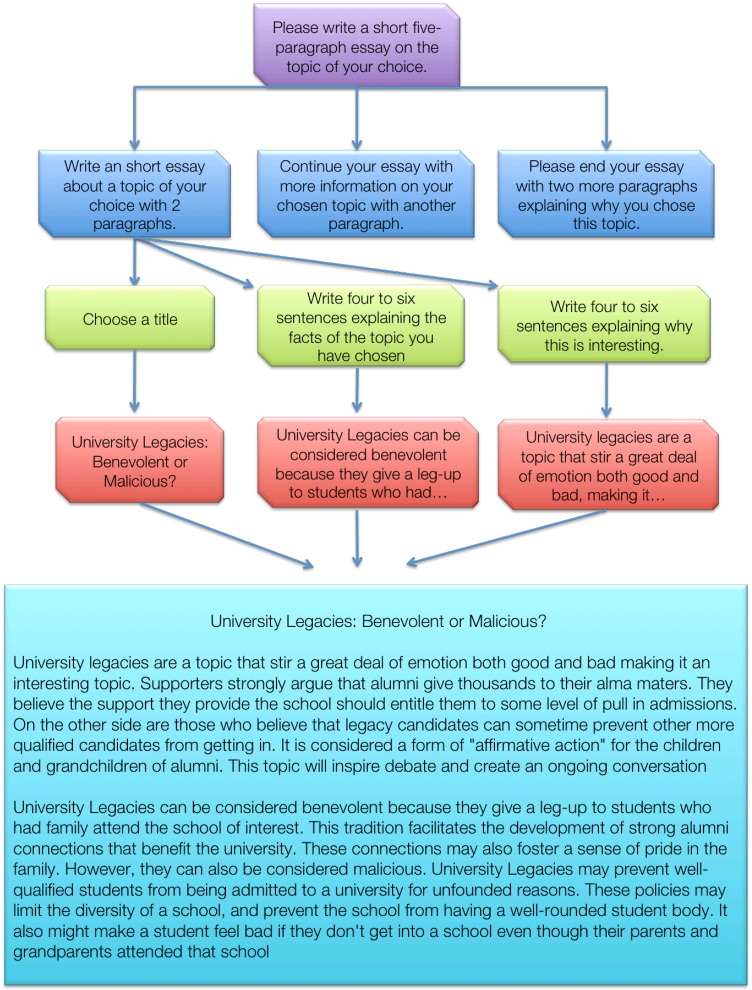

Figure 1. Turkomatic automatically produces workflows for complex tasks on Mechanical Turk. Above, results from an automatic essay-generation task given to Turkomatic, "Please write a short five-paragraph essay on a topic of your choice." The remainder of the essay is not shown to save space.

Turkomatic: Automatic Recursive Task and Workflow Design for Mechanical Turk

Anand Kulkarni[1]
Matthew Can[2]
Björn Hartmann[2]

University of California, Berkeley
1: Department of IEOR
2: Computer Science Division
Berkeley, CA 94720

{anandk,matthewcan,bjoern}
@berkeley.edu

CHI 2011, May 7–12, 2011, Vancouver, BC, Canada.
ACM 978-1-4503-0268-5/11/05.

Abstract
Completing complex tasks on crowdsourcing platforms like Mechanical Turk currently requires significant up-front investment into task decomposition and workflow design. We present a new method for automating task and workflow design for high-level, complex tasks. Unlike previous approaches, our strategy is *recursive*, recruiting workers from the crowd to help plan out how problems can be solved most effectively. Our initial experiments suggest that this strategy can successfully create workflows to solve tasks considered difficult from an AI perspective, although it is highly sensitive to the design choices made by workers. We also contribute a significantly simpler interface to Mechanical Turk that abstracts away most task design work. Requesters found this interface substantially faster and easier to use than the existing interface.

Keywords
Human computation, crowdsourcing, Mechanical Turk

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

Introduction

Designing effective workflows on microtask markets like Mechanical Turk is currently a black art. Employers who request work have to invest substantial effort and experimentation to convert the kind of open-ended high-level tasks people and organizations do every day ("write a paper about _____; build a webpage containing _____; build software that does ___") into tasks that can be solved effectively on crowdsourcing systems.

We propose that the problem of task design can be partly automated by delegating the responsibility for designing workflows to the workers themselves. Our system, Turkomatic, generates Human Intelligence Tasks (HITs) asking Mechanical Turk workers (Turkers) to decompose complex tasks into simpler ones. Other workers solve these tasks in parallel, and later combine the results into a coherent solution. This process can be recursive, generating multiple decompositions. We hypothesize that such a recursive framework for crowdsourcing will be capable of completing tasks that are currently considered beyond the scope of existing platforms for human computation.

In this paper, we describe the Turkomatic system and share results from early experiments using Turkomatic to solve two tasks: writing a brief essay and producing solutions to a high school Scholastic Aptitude Test. These experiments suggest several design choices that should be made to support automatic task decomposition and workflow design; we discuss these at the end of the paper.

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 [11]. Most work on Mechanical Turk today remains batch data processing [4]. Quinn and Bederson's taxonomy does not consider any large-scale creative or integrative tasks to be tractable by distributed human computation [8]. AI work around Mechanical Turk has emphasized its utility for supporting active learning rather than creative or open-ended problem solving [9,10].

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 [5,6]. Follow-up work by Little et al. compares the tradeoffs between iterative and parallel human computation processes [7]. 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 [2]. Bernstein et al. [1] propose a "Find-Fix-Verify" paradigm to divide open-ended work in a manner that maintains consistency and accuracy. Their Soylent system is the first to integrate human computation interactively into a creative process (word processing).

While most crowdsourcing tasks are designed manually based on prior experience and intuition, initial work by Huang et al [3] suggests that optimization techniques can play a useful role: by varying HIT parameters programmatically, response quality and response rate can be improved automatically.

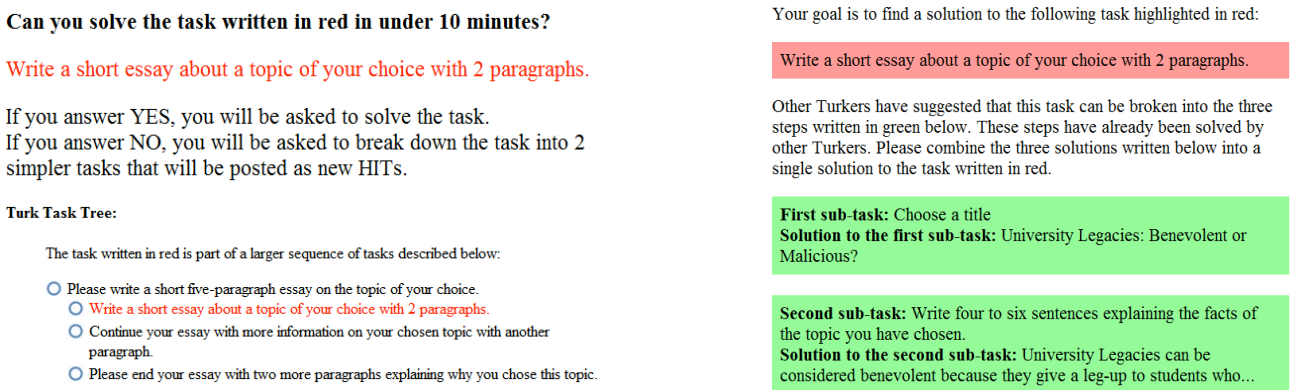

Figure 3. The "subdivide" HIT (left) asks workers to solve a task or break it into several steps; it also shows the overall task breakdown thus far. The "merge" HIT (right) shows a list of subtask solutions in green and the overall task to be solved in red.

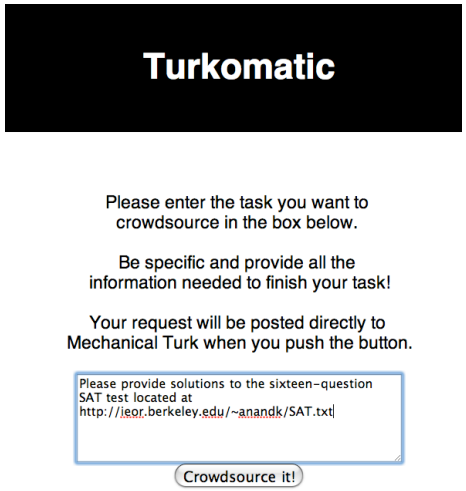

Figure 2. Requesters post tasks to Turkomatic by entering natural-language instructions.

System Description

The Turkomatic interface (Figure 2) accepts task requests written in natural language. For each request, it posts a HIT to Mechanical Turk asking workers to break the task down into a set of logical subtasks. These subtasks are automatically reposted to Mechanical Turk, where workers can choose to break tasks down further — a recursive subdivision — or to solve them directly. Once all subtasks are completed, HITs are posted asking workers to combine subtask solutions into a coherent whole, which is returned to the requester.

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. The associated HIT (Figure 3, left) 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). If a Turker indicates the task can be solved directly, the Turker is asked to do so. If the task is judged too complex, the Turker is asked to break down the task into two or more subtasks that are easier to solve than the original task. 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. We validate the quality of output produced by our subdivide function via redundancy, asking two or more Turkers to produce separate candidate solutions to each HIT and asking a pool of Turkers to vote on the best. The process of having Turkers select from redundant work for quality is a well-understood practice [1,6,7]. Therefore, the voting step was simulated by the authors in the presented experiments. To avoid ordering conflicts, the algorithm further asks Turkers to determine if a set of subtasks can be posted in parallel or must be serially executed; for the prototype, this step was also simulated by the authors.

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 3, right). The HIT instructs a worker to combine the solutions to the subtasks in a way that solves the overall task. As before, validation is handled through redundancy; each "merge" HIT is posted at least twice and separate Turkers vote to choose one. The merge process continues until the requester's original task is solved.

Task Template Design Guidelines

The biggest challenge in building a task decomposition system such as Turkomatic proved to be designing 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 findings for the design of HITs for task decomposition.

Show the context of subtasks in a workflow:
Providing workers with a birds-eye view of the overall decomposition proved critical. Workers often used different cognitive models in planning how tasks were decomposed (in particular, serial versus parallel), and could easily be 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, significantly reducing worker error.

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 contained in the HIT (prior work versus specific instructions); such emphasis was more effective than using indentation or whitespace. Minimizing the amount of text increased the likelihood that HITs would be solved as intended.

Use the best workers for planning and editing:
The quality of the overall product and the nature of the overall workflow were heavily influenced by the decompositions produced by the first workers during a subdivision. Many workers had difficulty understanding the requirements of a planning task, although most workers were able to solve leaf-level tasks. Selecting for higher-quality workers early on dramatically improved the quality of the final result. During the "merge" step, higher-quality workers were also willing to correct errors even in sections not assigned to them.

Informal Evaluation: Can Automatic Task Design Solve Complex Problems?

Our evaluation investigated whether our algorithm can produce successful workflows — and high-quality results — for high-level tasks posed in natural language. We examined the following tasks:

1. Producing a written essay in response to a prompt
2. Solving an example SAT test

These two tasks were chosen because they are difficult for existing autonomous algorithms to solve and are not currently considered to be within the scope of human computation platforms. For the first task, we fed the following statement into Turkomatic: "Please write a five-paragraph essay on the topic of your choice". For the second, we uploaded a partial practice exam for the high school Scholastic Aptitude Test to the web and posed the following task to Turkomatic:

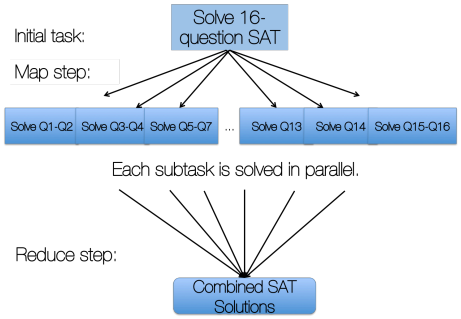

Figure 4. For the SAT task, we uploaded sixteen questions from a high school Scholastic Aptitude Test to the web and posed the following task to Turkomatic: "Please solve the 16-question SAT located at http://bit.ly/SATexam".

"Please solve the 16-question SAT located at http://bit.ly/SATexam". In both cases, we paid workers between $0.10 and $0.40 per HIT. Each "subdivide" or "merge" HIT received answers within 4 hours; solutions to the initial task were complete within 72 hours.

Results

The decompositions produced by Turkers while running Turkomatic are displayed in Figure 1 (essay-writing) and Figure 4 (SAT).

In the essay task, each "subdivide" HIT was posted three times by Turkomatic and the best of the three was selected by experimenters (simulating Turker 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 (for example, one Turker proposed: brainstorm, create outline, write topic sentences, fill in facts). The decomposition used in the final essay used two levels of recursion. As groups of subtasks were completed, Turkomatic passed solutions to merge 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 one (university legacy admissions) was somewhat specialized, but the final essay displayed a reasonably good understanding of the topic, even if the writing quality was often mixed. The decomposition selected for the SAT task used only one level of recursion. Workers divided the task into 12 subtasks consisting of 1 to 3 thematically linked questions. These were each solved in parallel by distinct workers and the results were given to a merge 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.

Obtaining useful decompositions proved tricky for workers – many seemed confused about the nature of the planning task. However, once the tasks were decomposed, solution of the constituent parts and reassembly into an overall solution were straightforward for Turkers to accomplish.

Evaluation: Interface

In 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 collect answers for a basic brainstorming task on Mechanical Turk. The task asked our participants to generate five ideas of topics for an essay. Participants performed this task twice, 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.

With Turkomatic, our users finished posting their tasks in an average of 37 seconds. On Mechanical Turk, where low-level task design was required, users needed an average of 244.2 seconds to post their tasks. More importantly, the HITs posted by two users who were not familiar with Mechanical Turk would not have produced any meaningful results. 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.

Participants 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.

Future Directions

Having elucidated strengths and challenges of fully automatic task design, we believe that an intermediate approach that facilitates dynamic interaction between the requester and the Turkomatic system while tasks are running may be promising. For instance, the requester could choose among several suggested decompositions produced by Turkers or provide clarification on unclear steps suggested by workers; the requester's answers could be incorporated into the HITs. This kind of interaction can enable Turkomatic to function as a requester support tool. Alternatively, these management functions could be delegated to workers with proven track records.

Saving high-quality workflows generated by Turkomatic and making them available to other requesters as templates is also promising. For instance, the essay-writing workflow we produced might be useful in a wider range of content generation tasks.

Finally, given that Turkomatic is capable of producing solutions to tasks considered conventionally difficult to crowdsource, it would also be interesting to examine whether Turkomatic offers a cost, quality, or speed advantage over existing methods. Such evaluations would provide further evidence that recursive interfaces offer strong potential as an approach to crowdsourcing.

References

[1] 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*.

[2] 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*.

[3] Huang, E., Zhang, H., Parkes, D.C., Gajos, K.Z., and Chen, Y. Toward automatic task design: A progress report. *KDD-HCOMP'10*. ACM, 2010.

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

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

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

[7] Little, G., Chilton, L.B., Goldman, M., and Miller, R.C. Exploring Iterative and Parallel Human Computation Processes. *KDD-HCOMP'10*. ACM, 2010.

[8] Quinn, A.J. and Bederson, B.B. A Taxonomy of Distributed Human Computation. UMD TR 2009.

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

[10] Sorokin, A. and Forsyth, D. Utility data annotation with Amazon Mechanical Turk. *CVPRW 2008*, 1-8.

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