Collaboratively Crowdsourcing Workflows with Turkomatic

Anand Kulkarni[1], Matthew Can[2], Björn Hartmann[3]

Department of IEOR[1], Computer Science Division[3]
University of California, Berkeley
anandk@berkeley.edu, bjoern@cs.berkeley.edu

Computer Science Department[2]
Stanford University
mattcan@cs.stanford.edu

ABSTRACT

Preparing complex jobs for crowdsourcing marketplaces requires careful attention to workflow design, the process of decomposing jobs into multiple tasks, which are solved by multiple workers. Can the crowd help design such workflows? This paper presents Turkomatic, a tool that recruits crowd workers to aid requesters in planning and solving complex jobs. While workers decompose and solve tasks, requesters can view the status of worker-designed workflows in real time; intervene to change tasks and solutions; and request new solutions to subtasks from the crowd. These features lower the threshold for crowd employers to request complex work. During two evaluations, we found that allowing the crowd to plan without requester supervision is partially successful, but that requester intervention during workflow planning and execution improves quality substantially. We argue that Turkomatic's collaborative approach can be more successful than the conventional workflow design process and discuss implications for the design of collaborative crowd planning systems.

Author Keywords

Crowdsourcing, task decomposition, workflows.

ACM Classification Keywords

H5.3. Information interfaces and presentation (e.g., HCI): Group and Organization Interfaces.

General Terms

Algorithms, Design, Human Factors

INTRODUCTION

Crowdsourcing marketplaces are increasingly used to solve computational problems that require human intelligence, such as audio transcription and data verification [11,16]. Many problems on markets like Amazon Mechanical Turk are solved as a series of *microtasks*: narrowly focused, brief tasks designed to be completed in a few minutes, such as labeling an image or checking the accuracy of data through web search. For quality assurance and to accomplish complex work, multiple microtasks are frequently chained together into *workflows*. Such workflows may decompose larger tasks into smaller subtasks, and later recompose subtask solutions into an overall work product. For example, a text editing workflow may ask one group of workers to *find* problems, another set to *fix* the identified problems, and a third to *verify* these edits [6]. A fundamental challenge in the use of crowdsourcing markets is the *workflow design problem*: how can employers divide a complex task into a set of microtasks that can be accurately solved by a pool of crowd workers?

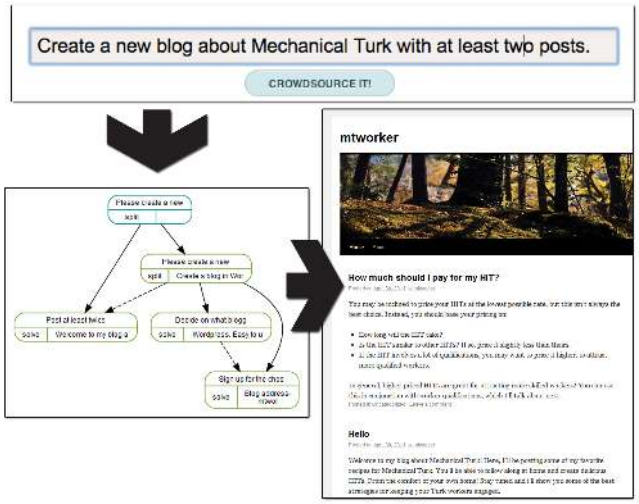

Figure 1: Turkomatic harnesses crowds to design and execute workflows in collaboration with requesters. A request (top) is subdivided into steps by the crowd. Turkomatic shows a task graph as work is progressing (left). Multiple workers execute steps of the created workflow until a global solution is produced – in this case, a new blog (right). Requesters can modify workflows in real-time.

Effective workflow design remains a major challenge today, involving both substantial planning, software development, and testing. Absent formal design methodologies, requesters commonly rely on an iterative process to construct good workflows. Requesters guess at a viable workflow, implement all of its steps as software that interfaces with a crowd platform, test it live with workers, identify points of failure, then iterate and modify the

CSCW'12, February 11–15, 2012, Seattle, Washington, USA.

workflow. The cost and complexity of this process limits participation in crowdsourcing marketplaces to experts willing to invest substantial time. These barriers to participation also limit the kinds of work that are crowdsourced today.

We propose that the responsibility for workflow design can be shared between the crowd and the requester. Turkomatic (Figure 1) is a novel crowdsourcing tool that allows the crowd to collaboratively design and execute workflows in conjunction with a requester. Turkomatic accepts a requester's specification of a broad objective, then asks workers on Amazon's Mechanical Turk to determine how to structure workflows to achieve the objective. The requester is able to monitor and edit the resulting workflows as they are produced.

Turkomatic uses a novel meta-workflow to induce the crowd to design and execute workflows. The system executes a continuous *price-divide-solve* loop that asks workers to recursively divide complex steps into simpler ones until they are at an appropriately simple level, then to solve them. Other workers are asked to verify the solutions and combine the results into a coherent answer to the original request. The use of pre-structured task templates and the participation of the crowd enables requesters to produce workflows without implementing software or designing intermediate tasks themselves. Compared to existing toolkits for programmatic workflow design, *e.g.*, TurKit [14] and Jabberwocky [4], these features lower the threshold for employer participation in crowdsourcing.

The price-divide-solve loop produces directed, acyclic *task graphs*: nodes represent subtasks and links describe task dependencies. The task graph is constructed during divide steps, and completed during solve steps. Turkomatic's graphical user interface allows the requester to visualize, monitor and interact with task graphs in real-time. Requesters can delete or modify plans made by workers, request the crowd to re-plan components of workflows, or seed the system with partial plans to evaluate their effectiveness. Real-time visualization of workflow structures and partial solutions enable requesters to debug many of the common failure modes that arise in multi-step workflows. These include cases where workers misunderstand tasks, give inadequate solutions, or where the requester's original language is unclear.

Following a review of related work, we introduce the algorithmic foundations of the *price-divide-solve* loop and discuss how Turkomatic operates from both the requester and worker perspectives. A first experiment explores how workers succeed and fail to plan tasks without expert supervision. A second experiment demonstrates how workers and requesters can successfully interact through Turkomatic on a variety of planning and execution tasks. We close by reporting on usage of Turkomatic outside the lab. Our findings lead to a number of design implications for collaborations on crowdsourcing platforms.

RELATED WORK

Early work in human computation emphasized its utility as a tool for efficiently processing large datasets in applications like tagging and classification that were outside the reach of autonomous algorithms [5]. Artificial intelligence researchers emphasized the utility of crowds for supporting active learning problems [17,19]. Most paid microtask work today remains batch data processing [11]. Quinn and Bederson's taxonomy of human computation does not include any creative or integrative tasks [16].

More recent research has attempted to expand the types of tasks that can be solved via distributed human computation. VizWiz is capable of handling open-ended, natural-language requests from users [7]. Soylent provides crowd-powered text editing services in a word processor [6]. Its "find-fix-verify" design pattern divides work in a manner that maintains consistency and accuracy. Legion enables multiple crowd workers to control existing user interfaces in real-time [13].

New toolkits seek to support such complex applications. TurKit aids requesters in deploying iterative tasks on Mechanical Turk; its crash-and-rerun architecture saves intermediate results to avoid redundant crowd assignments [14]. Jabberwocky can target workers in both paid and volunteer workforces [4]. In these toolkits, it is assumed that task designers (not workers) will determine how tasks are broken down in all cases.

While most crowdsourcing tasks are designed manually based on prior experience and intuition, formal techniques can also play a useful. By optimizing task parameters programmatically [10], modeling the market [8], or introducing auction mechanisms [18], response quality and response rate can be improved. Task templates can also be automatically created for data gathering tasks based on an underlying database schema [9].

Both text-based and visual workflow languages have been devised in prior work; Stohr presents a survey of such systems [20]. Some workflow languages have recently been extended to connect to crowdsourcing marketplaces. Crowdsourcing companies such as Crowdflower use the domain-specific workflow language Ruote [1] to implement complex tasks. Ruote is only accessible to expert programmers. RunMyProcess is a commercial service to author and execute business processes in the cloud [2]. It uses Business Process Modeling Notation [21], a standard diagramming convention, as the basis for a visual workflow design tool. Olsen used RunMyProcess to post tasks and receive answers from Mechanical Turk [15].

Most recently, CrowdForge introduced a map-reduce paradigm to divide complex work into smaller steps for crowdsourcing platforms [12]. 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.

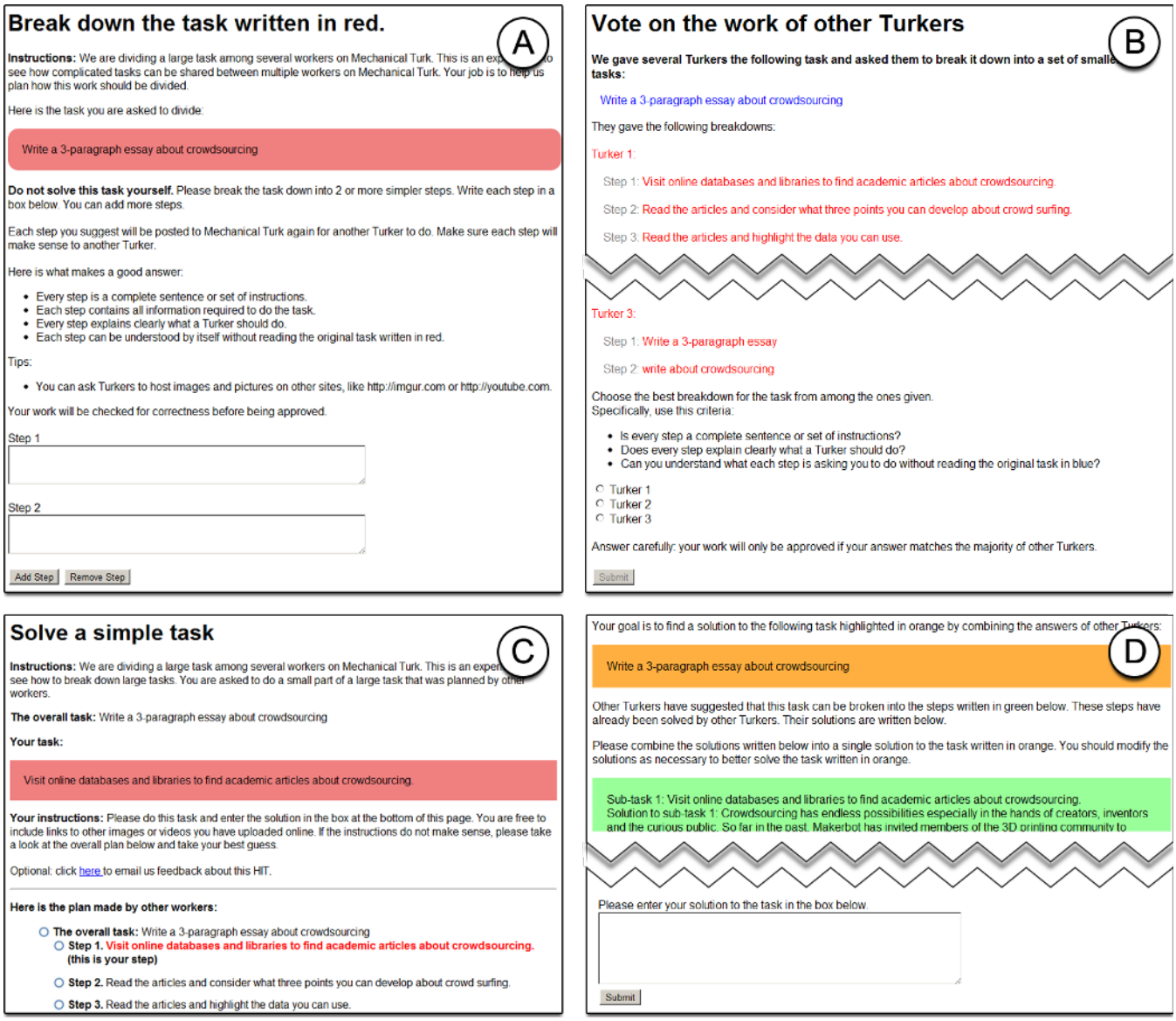

Figure 2: The worker interface: HITs corresponding to (A) subdivision, (B) verification, (C) solution and (D) merging.

THE TURKOMATIC SYSTEM

Turkomatic consists of two major components. First, a *meta-workflow* uses the crowd to assist in the design of the workflow, and to execute it. Our novel *price-divide-solve* algorithm (PDS) uses a divide-and-conquer strategy to let workers decide how to split up tasks. Second, an *editable workflow visualization* enables requesters to observe and manage work being executed by the crowd. We discuss the algorithm and the associated worker interfaces first, then describe Turkomatic's requester interfaces.

Algorithmic Model: Price-Divide-Solve, Then Merge

PDS is a general-purpose algorithm for inducing the crowd to generate workflows for tasks specified by the requester. The algorithm guides workers through the process of converting large, complex tasks into microtasks appropriate for crowd markets. Once all the microtasks for a given component of the workflow are solved, a *merge* stage asks workers to recombine work back into a coherent single solution to the original task. Both stages rely on *verification* functions that determine when work needs to be checked or repeated. For each step in the algorithm, task templates are automatically instantiated with task context and posted to a crowd marketplace (see Figure 2). Turkomatic posts Human Intelligence Tasks (HITs) on Amazon Mechanical Turk, through the algorithm is independent of this particular platform.

Price-Divide-Solve

The *price-divide-solve* phase splits work into fairly priced sub-tasks. An initial pricing HIT presents the overall task goal and asks whether or not this task can be solved for a fixed price (in our implementation, twenty cents). If a worker indicates the task can be solved directly at the stated price, a new HIT will be posted, asking another worker to

```
PDS(Task t, Price p, Redundancy r):
  if crowdIsFairPrice(t,p):
    return solve(t,r)
  else:
    subtasks = divide(t,r)
    for s in subtasks:
       subsolutions += PDS(s,p,r)
    return merge(t,subsolutions,r)

divide(Task t, Redundancy r):
    for j in [1,r]:
      divisions += crowdSubdivide(j)
    return crowdChooseBest(divisions)

solve(Task t, Redundancy r):
    for j in [1,r]:
      solutions += crowdSolve(t)
    return crowdChooseBest(solutions)

merge(Task t, Subsolutions s, Redundancy r):
    for j in [1,r]:
      merges += crowdMerge(t,s)
    return crowdChooseBest(merges)
```

Figure 3: Pseudocode for the PDS algorithm. Calls to crowd functions (blue) correspond to HITs shown in Figure 2.

do so (Figure 2C). If the task is judged too complex, the next worker is asked to divide the task into two or more subtasks that are easier to solve than the original task (Figure 2A). To capture information dependencies, workers indicate whether the subtasks can be worked on in parallel or whether they must be completed sequentially.

The set of subtasks is then posted to Mechanical Turk. This process is recursive: subtasks generated by the subdivide step may themselves be broken down by another step. Figure 3 shows pseudo-code for the entire PDS algorithm.

Merge Phase

The *merge* step combines subtask solutions elements produced during *divide* steps. Once all the subtasks produced in a given subdivide step have been solved, the solutions are listed together in a *merge* HIT (Figure 2D). The HIT instructs a worker to combine the subtask solutions in a way that solves the overall goal at the given level of the task hierarchy. 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 vote on the best of several redundant answers. For each of these functions, Turkomatic requests *r* answers from the crowd, where *r* is an initial redundancy parameter specified by the requester (see Figure 3). For simplicity, *r* is kept uniform across all types of HITs, though this parameter could be optimized for each HIT independently. In *verification* HITs (Figure 2B), a worker is presented with a task and its candidate decompositions or solutions; the worker is asked to vote on the answer that best accomplishes the task. Using workers to select from redundant answers for quality control is a common strategy in many crowdsourcing applications [6,14].

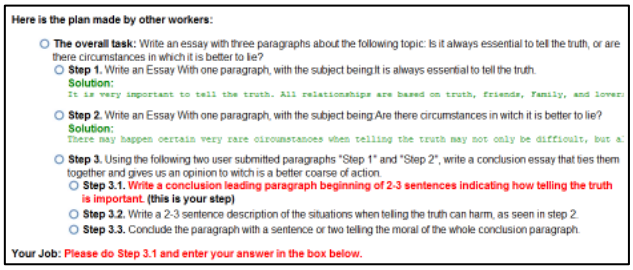

Figure 4: Turkomatic shows task context to workers.

Task Design Strategies

Designing general-purpose HITs that fit a wide variety of tasks while still conveying specific requirements of the decomposition and merge phases to a worker is challenging. In our experience, two design techniques proved especially useful and necessary: 1) including larger task context in subtasks, and 2) visually separating task language from context language.

Show Context of Tasks in a Workflow

Providing workers with a birds-eye view of the overall decomposition is critical. Workers can easily become confused if their role in the larger process is not made explicit. In addition, workers may need to refer to information contained in prior answers to adequately solve their sub-task. To communicate the larger goal, and the work completed towards this goal, Turkomatic shows an indented list view of the decomposition generated by other workers (Figure 4), highlighting the current subtask. This view shows all available information. However, presenting complete task status can also lead to confusion if the task is already extensively subdivided. We reserve further study of the optimal amount of context information that should be presented to future work.

Visually Separate Task Language from Context Language

The HITs for subdivide and merge steps contain complex information and require careful attention of the worker. Workers can have difficulty identifying which task in a complex plan they were being asked to carry out. We used background colors to emphasize and separate the primary task instructions from explanatory and context information. Type treatment, e.g., bold, red text, was used as an effective second level of emphasis. In pilot testing, these attributes were more effective than indentation or whitespace.

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 1). Inspired by web search, it offers a single text box where the requesters specify what they want to accomplish.

Expert requesters can also author decompositions that serve as a starting point for the PDS algorithm. Such decompositions may be preferred if the requester believes

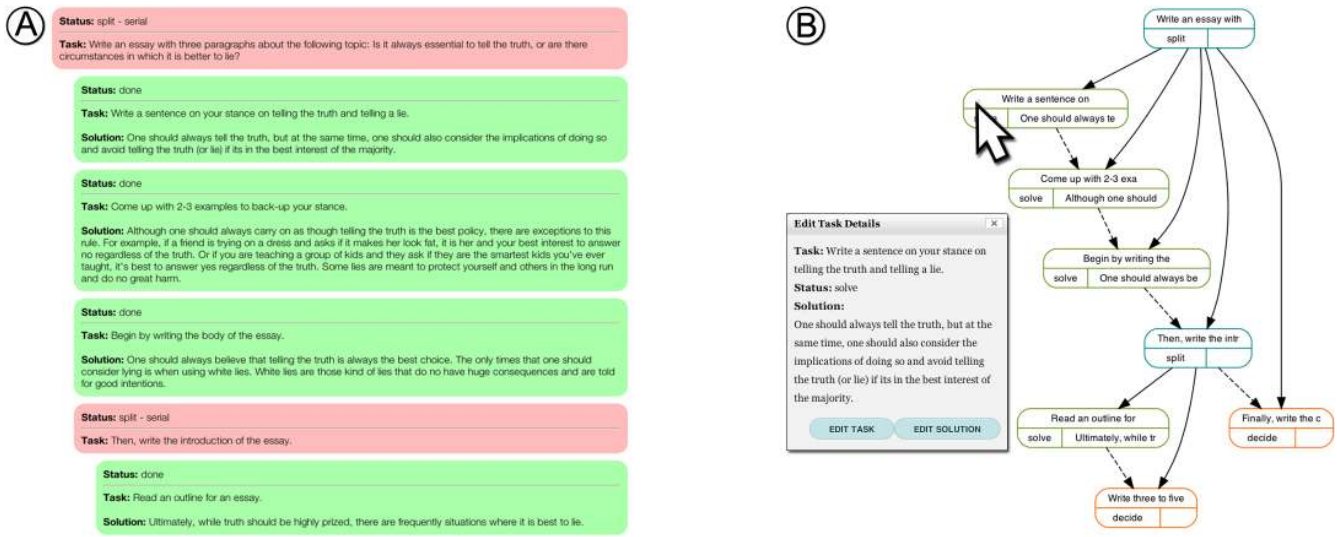

Figure 5: The requester interface: To see content, requesters use an indented list view (A). To see the dependency structure of the workflow, requesters use a graph view (B). A complete task description is shown in a floating panel. In both views, steps are color-coded to indicate their status: waiting (orange), in progress (red/cyan), completed (green).

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. Turkomatic provides an indented list interface for authoring decompositions (Figure 6). Once a new task prompt or task tree is loaded, the PDS algorithm recruits crowd workers to price, subdivide, and solve the leaf tasks, and eventually merge solutions to accomplish the root task.

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 either an indented list (Figure 5A) or a decomposition graph (Figure 5B). The list view allocates most screen space to task content (*i.e.*, written descriptions and solutions). The graph shows hierarchy, as well as parallel and serial decompositions. Both visualizations 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.

Figure 6: Requesters can author partial workflows before starting the PDS algorithm.

Turkomatic uses GraphViz [3] to render decomposition graphs. Nodes in the graph represent the component tasks of a job. Solid directed edges show the relationship between tasks and sub-tasks (*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 available), and a status indicator. A task can either be a) waiting on a decision whether to split or solve (orange in Figure 5B); b) in-progress and waiting for subtasks to complete (cyan); or c) solved (green). The tree visualization is interactive: brushing over nodes displays complete instructions and solutions in a floating panel, as the text often cannot fit in the node. This visualization remains legible for tasks with tens of nodes. For complex graphs, additional interaction techniques such as collapsible branches may become necessary.

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 (Figure 7). Requesters edit the task description, decomposition, or solution for any node. Once an edit is completed, Turkomatic computes which tasks will have to be performed again by additional workers.

When a task *description* is changed (Figure 8A), any subtasks created for this task by the crowd may also no longer be valid. Turkomatic therefore invalidates the entire sub-tree below the edited task and reissues the task (Figure 8B). If the task already had a solution, all upstream solutions of parents that used this stale information have to

Figure 7: Requesters select a node to edit a workflow (A), and enter new content (B). Turkomatic computes which subtasks have to be re-issued due to the edit and restarts the PDS algorithm (C).

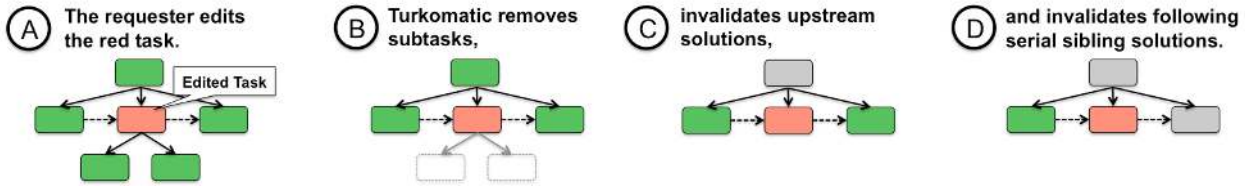

Figure 8: When requesters edit subtasks, children, parent solutions, and sibling solutions may have to be recomputed.

be reissued to the crowd as well (Figure 8C). Finally, subsequent siblings in serial decompositions also have to be reissued (Figure 8D). When a requester edits a task *solution* or *decomposition* directly, the entire sub-tree of that task is discarded. Tasks in this sub-tree that are currently being answered will also be discarded. As with task instructions, serial siblings and solutions of parents are also invalidated.

HOW WELL CAN CROWDS PLAN AND EXECUTE?

To explore how effectively crowds can be used to support the execution of complex work, we performed two evaluations. First, we examined how crowds performed in producing and solving workflows without expert intervention. Second, we looked at how expert intervention can improve the crowd's performance. Additionally, we informally observed use of the Turkomatic platform outside the lab. We discuss each of these evaluations in turn.

Unsupervised Crowd Planning

Can the crowd be guided algorithmically to plan and solve problems without any input from requesters? Undirected crowd planning seems *a priori* unlikely to produce usable workflows: design problems often require extensive communication and coordination between client (requester) and designer (worker). In addition, correct interpretation of a prompt may rely on specific domain knowledge. If that knowledge is not shared between requester and worker, instructions are likely to be misinterpreted. To provide a baseline of success and failure patterns, we ran the PDS algorithm with no monitoring by the requester.

Procedure

Via Turkomatic, we requested that the crowd plan workflows for and compute answers to several types of objectives, including essay writing, natural language queries, itinerary planning, Java programming, and multimedia content generation. We ran more than twenty distinct kinds of queries in total. If the crowd had not completed planning and solving a task within 5 days, we posted it a second time. In each of these experiments, we paid $0.05 for *price* and *verify* HITs; and $0.20 for *divide*, *solve* and *merge* HITs. Workers were given 20 minutes to complete each HIT; payments were automatically approved after 24 hours. For five of these tasks, we also systematically modified the redundancy parameter. Requests given to the crowd were phrased as follows:

- **Essay writing**: "Write a 3-paragraph essay about crowdsourcing."
- **Natural language query**: "Create a list of the names of the Department Chairs of the top 20 computer science college programs in the US."
- **Itinerary planning**: "Plan a complete road trip from San Francisco, California, to New York City. Completely include the locations of all necessary hotels, restaurants, and sights along the way."
- **Java programming**: "Please write a short piece of Java code to reverse a string. The algorithm should take as input a string and output its reverse. Make sure it compiles."

For each of these queries, we posed the objective to the crowd through Turkomatic with redundancy levels of one (no redundancy), two, and three.

Results and Observations

We observed both successes and systematic failures. Table 1 shows some task outcomes. The results can be grouped into a small number of classes, described below.

Task Description	Outcome	Subtasks	Sample Data & Observations
Three-paragraph essay: Is it always essential to tell the truth, or are there circumstances in which it is better to lie?	Success with complex planning	7	Top level breakdown was to write one paragraph arguing for one position, another paragraph arguing for the other position, and a conclusion paragraph reconciling the two.
Three paragraph essay: Do we learn more from finding out that we have made mistakes or from our successful failure?	Failure: Starvation	17	Sample response: "I've been through a lot in my life and one thing I've learned is never, ever, ever, even think about smoking or doing drugs. I spent years quitting from smoking and I've learned that lesson."
Write Java code to reverse a string	Success: Snap judgment	1	
Plan a road trip from San Francisco to New York City	Derailment	55	Cyclic behavior: workers recomputed a list of landmarks at least 3 times.
List department chairs of top 20 Computer Science programs in the US	Derailment	5	Loss of context: Solution was a list of IKEA chairs.

Table 1: Experimental results for several unsupervised tasks. Outcome indicates type of success (snap judgment, complex planning) or type of failure (derailment, starvation).

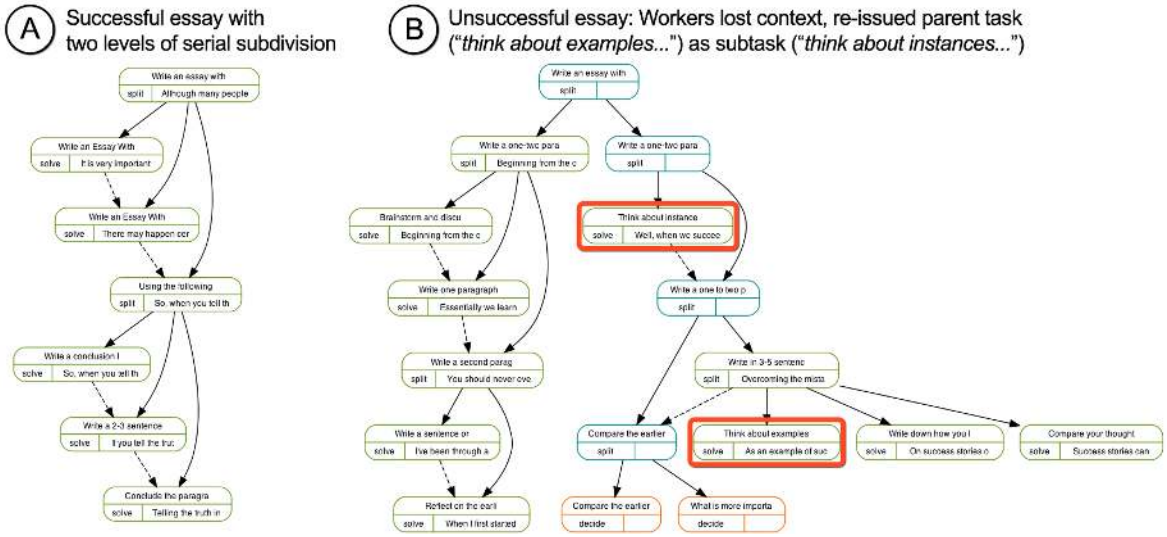

Figure 9: (A) Successful essay decomposition. (B) Derailment due to task cycling Highlighted nodes mark repeated tasks.

Snap Judgments

On a subset of the work, the crowd provided reasonable, correct solutions. One class of successes were *snap judgments*: cases where there was no decomposition of the input task, but where a worker marked the first task as solvable, and a subsequent worker provided a correct, coherent and complete answer. In these outcomes, a correct result is produced without additional decomposition or planning. This success is unsurprising, as no collaboration is involved when no workflow is required: the same worker both determines how to solve a task and solves it.

Successful Planning

In some cases, groups of workers broke down tasks in line with requester intentions, and this was sustained over multiple iterations. For two writing tasks taken from sample SAT essay questions, Turkomatic produced coherent essays with reasonable arguments (Figure 9A – here the task tree has two levels of serial subdivision; each division split the original task into three subtasks).

This outcome occurred infrequently, which is not surprising: it required a shared cognitive model of how work should be decomposed to be maintained by multiple workers in the execution process. We found that the active participation of one or more highly eager workers early in the process made a substantial difference in downstream quality. The strongest workers provided instructions that were self-contained and encapsulated all necessary information independently in each subtask. Downstream workers tended to pattern their contributions after models established by earlier workers.

Challenges:
Task Derailment, Emergent Complexity and Cycling

In the majority of cases, the unsupervised crowd produced unsuitable workflows or unsuccessful results. The most frequent type of failure was *derailment*, a phenomenon that occurred when the PDS algorithm failed to terminate and continued to produce steps indefinitely. An example of this phenomenon is shown in Figure 9B. Derailment occurred for multiple reasons. First, workers were confused about

appropriate task granularity. For instance, to write a paragraph as part of an essay-writing task, one worker determined the first subtask for another worker should be to "acquire a writing utensil or a computer" – losing track of the original objective in an effort to subdivide as small as possible. Second, workers authored subdivisions that could be executed by a single worker, but not split across separate workers (e.g. "think about what to write for the next paragraph" followed by "write it down"). Finally, workers who had lost the context of the overall workflow generated decompositions that restated previous tasks as subtasks, leading to cyclic behavior (Figure 9B, red highlight).

Task Starvation

Some workflows failed to complete due to *starvation* – after a while, no new workers attempted the available tasks, and the time limit for the experiment expired without the execution of work continuing further. Task starvation has been observed in other projects [6,7] and has been counteracted in those contexts through listing optimization or chaining sets of tasks to retain workers. In the context of the PDS algorithm, starvation occurred most often when a worker marked a task as solvable when, judging by its complexity, it should have been subdivided.

Standard Quality Assurance Techniques are Insufficient

Surprisingly, adding redundancy by asking more workers to contribute and vote on subdivisions at each step failed to yield more successful decompositions. This is in contrast to other forms of collaborative content creation such as Wikipedia, where increasing the numbers of workers tends to improve the quality of content. One explanation is that Turkomatic's model for redundancy does not allow workers to iterate towards correct answers, but simply increases the parallelism in each step. Workers tended to vote for decompositions that were more detailed and produced more subtasks, introducing more potential failure points.

Interestingly, eager workers sometimes arrived in already derailed workflows and made efforts to correct the state of the system. In several cases workers accepted Turkomatic HITs, but were not satisfied with the workflow provided by previous members of the crowd: these workers chose to email the authors directly to suggest improvements to the task. This suggests that making the Turkomatic requester interface available to motivated workers to allow the crowd to self-police and improve its own work.

COLLABORATIVE PLANNING AND EXECUTION

The results of the previous section suggest that the crowd can sometimes produce acceptable results in planning and executing solutions to general requests, but more frequently faces challenges. At least two different strategies to address these challenges are plausible. The first strategy is to *recruit more expert workers*: it may be that the pool of workers available on Mechanical Turk is not sufficiently qualified to participate in planning tasks. The second strategy is to *allow requesters to monitor and selectively intervene* in the workflow design and execution process. In this section, we describe informal experiments for both approaches: first, with an expert crowd of workers recruited at UC Berkeley; and second, with a Mechanical Turk crowd, augmented through active requester participation using Turkomatic's monitoring interface.

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)	Expert workers	Completed without intervention
	Requester monitoring	Task completed after 3 interventions
Write a 3-paragraph essay about crowdsourcing	Expert workers	Completed without intervention
	Requester monitoring	Task completed after 4 interventions
Please create a new blog about Mechanical Turk, with a post and a comment on that post.	Expert workers	Completed with 1 intervention
	Requester monitoring	Task completed with requester termination

Table 2: Results from the second set of Turkomatic Experiments.

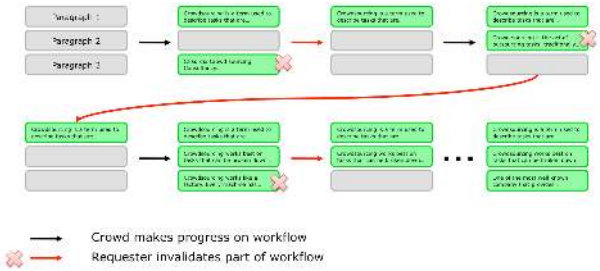

Figure 10: Requesters can intervene repeatedly to increase the quality of the work. Here, different paragraphs of a three-paragraph essay are independently written by different workers. Some work is not acceptable to the requester, so she reissues subtasks.

We re-ran Turkomatic with the essay-writing, blog creation, and natural-language query tasks discussed earlier, in two conditions. In the first condition, we used a pool of five expert workers drawn from a graduate computer science course who had reported experience using Mechanical Turk both as workers and requesters. The experts were males with ages ranging from 21 to 33. These experts were broadly aware that Turkomatic was a system for crowdsourcing complex work, but were not given additional instructions beyond those provided in the HITs. In the second condition, we continued to use Mechanical Turk to recruit workers, but used the requester interface to monitor and edit tasks. To avoid snap judgments, we seeded initial decompositions into the workflow. HITs were priced identically to the unsupervised experiments.

Outcome

Table 2 summarizes the results we obtained, counting the number of interventions required for planning and execution of the tested tasks. Both conditions avoided starvation and derailment and resulted in usable solutions. When expert workers carried out decomposition and solution of the subtasks, solutions were reached rapidly.

When requesters intervened using the workflow editor, the resulting workflows were executed correctly in all cases (*i.e.*, they reached a correct solution without derailment or starvation). Figure 10 shows an example of how an essay-writing workflow evolved during the experiment as requesters intervened to invalidate and re-request specific components of workflows. Requesters did not edit any solutions provided by workers to reach this outcome; only rejection of inconsistent work was required.

Discussion:

Why Does Collaborative Workflow Creation Succeed?

When requesters used workflow editing tools to guide the crowd's efforts, tasks completed successfully. Intervention enabled requesters to provide feedback and to iterate on unsuccessful tasks. If a workflow is executed without input from the requester (unsupervised), workers must design plans or create content with only a limited understanding of the requester's intent or preferences. This is in striking contrast to the traditional model of design where designer and clients to converge on a solution through repeated iteration. For complex work, it seems especially important for the requester to provide feedback. Manual intervention and editing of a crowd-generated workflow is an effective (if indirect) way to do so.

For example, requester instructions in our tasks were sometimes inadequate in expressing what a requester actually wanted. In an essay-writing task, when crowd workers submitted solutions to individual paragraphs that were clearly copied from Wikipedia, we used Turkomatic's editing interface to modify the task instructions (asking not to copy text from another source) and reissue the task. Improving task instructions in response to the crowd's initial failed attempts can be seen as an application of iterative design to crowd programming. Such iterations allowed tasks to succeed.

USER EXPERIENCES WITH TURKOMATIC

The authors have been using Turkomatic as a platform since its creation to test a variety of tasks. We have also shared Turkomatic with a number of end users within academia and the computer science community who requested solutions to various tasks. Two use case patterns have emerged: first, Web search and data processing, such as finding real estate listings, email addresses, or cross-checking multiple web documents; second, one-time experimentation to see if particular new classes of tasks can be solved by workers on a crowd platform. Results from some of these tasks are illustrated in Figure 11. Some users indicated that the extra monetary cost involved in using

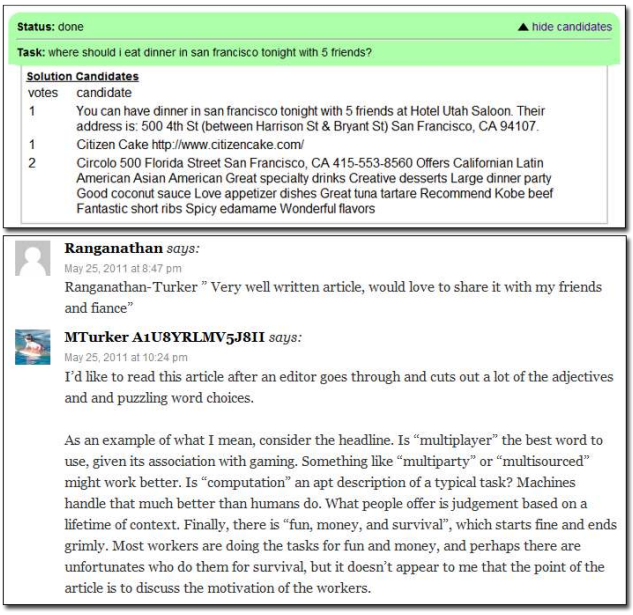

Figure 11: Results from two example tasks submitted by Turkomatic users. Above: asking for restaurant recommendations. Below: requesting blog comments.

Turkomatic over regular crowdsourcing tasks was made up for by its simple, fast interface.

These outcomes illustrate two additional benefits of Turkomatic: It has value in quickly evaluating the ability of crowds to solve particular kinds of work, and it can reduce the complexity of accessing crowd platforms for casual use. These additional benefits stem from the fact that Turkomatic's initial interface requires only a goal stated in natural language. We hypothesize that crowdsourcing marketplaces can grow substantially if the process of sending work to crowds is simplified.

DISCUSSION

We close with several observations that arise from our work with the Turkomatic system on collaboratively designing workflows with the crowd.

Instructional Writing is Difficult for Workers

Composing good instructions is not trivial and takes time and effort. 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, few other tasks on Mechanical Turk require such careful attention, and workers may thus be disincentivized from delivering nuanced work.

Reputation Effects Arise from Worker Planning

Turkomatic assigns responsibility for wording tasks to the workers. Interestingly, this means that poor choices by Turkers may negatively affect the requester's reputation. We noticed that workers on Turker Nation, a discussion forum, 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.

Excessive Structure Limits the Effectiveness of Leaders

We noticed that expert workers provided more detailed instructions in their subdivisions and attempted to communicate corrections to requesters through external channels of communication. The PDS algorithm does not yet offer workers the ability to contribute at different levels commensurate with skill and interest. This phenomenon suggests that more effective platforms for collaboration with the crowd should permit workers to edit workflows much as requesters do in the current systems.

Scaling Requires Context-Free Workflows

Batch processes send many different problem instances through the same workflow (*e.g.,* filtering a database of business addresses through an address verification workflow). This implies that such workflows must be designed in a *context free* manner – the instructions have to be written independently of any particular problem instance. Turkomatic currently does not offer a guarantee that crowd-planned workflows are context free. One promising idea for future work is to first ask the crowd to produce a concrete workflow; and afterwards generalize it to fit multiple problem instances.

CONCLUSION

This paper introduced Turkomatic, a system that harnesses crowds to design and execute workflows for complex tasks. Turkomatic is based on a price-divide-solve algorithm that guides workers through the steps of decomposing and solving tasks; and on generic task templates that are instantiated with particular task contexts. Our experience with Turkomatic suggests that unsupervised workers face planning challenges. However, experiments also showed that planning succeeds for an interesting range of tasks when more knowledgeable workers are recruited and when requesters can review and edit crowd work in real-time.

However, the one-size-fits-all model of Turkomatic trades off simplicity of use for runtime supervision: workflows can be generated without exhaustive planning, but require requester monitoring 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, and how the pricing structure of Turkomatic can be optimized. Effective workflow design is among the most common problems facing crowdsourcing researchers today. Why not collaborate with workers in solving it?

ACKNOWLEDGMENTS

This work was supported by gifts from Intel and Google.

REFERENCES

1. Ruote. http://ruote.rubyforge.org/.
2. RunMyProcess. http://www.runmyprocess.com/.
3. Graphviz. http://www.graphviz.org/.
4. Ahmad, S., Battle, A., Malkani, Z., and Kamvar, S. The Jabberwocky programming environment for structured social computing. *Proceedings of UIST 2011*, (2011), 53-64.
5. Ahn, L. von. Games with a Purpose. *IEEE Computer 39*, (2006), 92–94.
6. Bernstein, M.S., Little, G., Miller, R.C., et al. Soylent: a word processor with a crowd inside. *Proceedings of UIST 2010*, ACM (2010), 313–322.
7. Bigham, J.P., Jayant, C., Ji, H., et al. VizWiz: nearly real-time answers to visual questions. *Proceedings of UIST 2010*, ACM (2010), 333–342.
8. Faridani, S., Hartmann, B., and Ipeirotis, P.G. What's the Right Price? Pricing Tasks for Finishing on Time. *Proceedings of HCOMP11: The 3rd Workshop on Human Computation*.
9. Franklin, M.J., Kossmann, D., Kraska, T., Ramesh, S., and Xin, R. CrowdDB: answering queries with crowdsourcing. *Proceedings of SIGMOD 2011*, (2011), 61-72.
10. Huang, E., Zhang, H., Parkes, D.C., Gajos, K.Z., and Chen, Y. Toward automatic task design: a progress report. *Proceedings of the ACM SIGKDD Workshop on Human Computation*, ACM (2010), 77–85.
11. Ipeirotis, P.G. Analyzing the Amazon Mechanical Turk marketplace. *XRDS: Crossroads, The ACM Magazine for Students 17*, (2010), 16–21.
12. Kittur, A., Smus, B., Khamkar, S., and Kraut, R.E. CrowdForge: Crowdsourcing Complex Work. *Proceedings of UIST 2011*, ACM (2011), 43-52.
13. Lasecki, W.S., Murray, K.I., White, S., Miller, R.C., and Bigham, J.P. Real-time crowd control of existing interfaces. *Proceedings of UIST 2011*, (2011), 23.
14. Little, G., Chilton, L.B., Goldman, M., and Miller, R.C. TurKit: human computation algorithms on mechanical turk. *Proceedings of UIST 2010*, ACM (2010), 57–66.
15. Olsen, T. Incorporating crowdsourcing into business processes. *Adjunct Proceedings of CSCW 2011*, (2011).
16. Quinn, A.J. and Bederson, B.B. Human computation: a survey and taxonomy of a growing field. *Proceedings of CHI 2011*, ACM (2011), 1403–1412.
17. Sheng, V.S., Provost, F., and Ipeirotis, P.G. Get another label? Improving data quality and data mining using multiple, noisy labelers. *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM (2008), 614–622.
18. Singer, Y. and Mittal, M. Pricing Tasks in Online Labor Markets. *Proceedings of HCOMP11: The 3rd Workshop on Human Computation*, (2011).
19. Sorokin, A. and Forsyth, D. Utility Data Annotation with Amazon Mechanical Turk. *Proceedings of CVPR 2008*, (2008).
20. Stohr, E.A. and Zhao, J.L. Workflow Automation: Overview and Research Issues. *Information Systems Frontiers 3*, (2001), 281–296.
21. White, S. *BPMN modeling and reference guide*. Future Strategies Inc., Lighthouse Point Fla., 2008.