TurKit: Tools for Iterative Tasks on Mechanical Turk

Greg Little, Lydia B. Chilton, Robert C. Miller, and Max Goldman
MIT CSAIL
32 Vassar St
Cambridge, MA 02139 USA

{glittle,hmslydia,rcm,maxg}@mit.edu

ABSTRACT

Mechanical Turk (MTurk) is an increasingly popular web service for paying people small rewards to do human computation tasks. Current uses of MTurk typically post independent parallel tasks. We are exploring an alternative *iterative* paradigm, in which workers build on or evaluate each other's work. We describe TurKit, a new toolkit for deploying iterative tasks to MTurk, with a familiar imperative programming paradigm that effectively uses MTurk workers as subroutines.

Categories and Subject Descriptors

H5.2 [**Information interfaces and presentation**]: User Interfaces. - *Prototyping*.

General Terms

Algorithms, Design, Economics, Experimentation

Keywords

Human computation, Mechanical Turk, toolkit

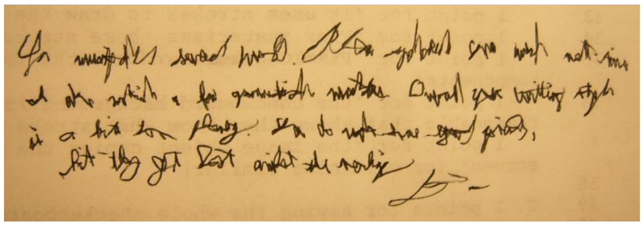
1. INTRODUCTION

MTurk is an increasingly popular web service for paying people to do simple human computation tasks. Workers on the system (*turkers*) are typically paid a few cents for Human Intelligence Tasks (HITs) that can be done in under a minute. Currently, MTurk is largely used for *independent* tasks. Task requesters post a group of HITs that can be done in parallel, such as labeling 1000 images. This demo considers a different model for employing turkers: **iterative tasks**, in which a succession of turkers do tasks that build each other. For example, turkers can take turns improving a passage of text; verify each other's work by voting on it; and implement the comparison function of an iterative sorting algorithm.

Next we will touch on some related work, followed by a presentation of an example of an iterative task on Mechanical Turk. We will finish with an overview of TurKit—the toolkit used to create this task—followed by directions for future work.

2. RELATED WORK

One challenge in writing human computation algorithms is motivating humans to do work. One approach is Games With a Purpose [1], where humans perform useful computation as a byproduct of playing computer games. User-generated content websites such as Wikipedia use human computation to generate content, and this content along with social factors seem to motivate future contributions. MTurk provides a platform for performing Human Intelligence Tasks (HITs) where humans are motivated by money.



[illegible]

Figure 1: A sample of bad handwriting.

3. ITERATIVE TEXT IMPROVEMENT

The iterative text improvement experiments take inspiration from the way some Wikipedia articles grow from a simple sentence into a fully fledged article as multiple people make small contributions [3]. In our experiments, we start with a seed of text and ask turkers to improve it according to some instructions. After each attempted improvement, additional turkers vote whether the change is indeed an improvement. The winning text is fed back into the system for further improvement, until a stopping condition is met.

We have explored a number of iterative text improvement tasks, including image description, copy editing, and brainstorming. For reasons of space, we present only one example here: handwriting recognition.

3.1 HANDWRITING RECOGNITION

Most OCR software focuses on recognizing printed fonts. The reCAPTCHA project applies human computation to correct errors in OCR [2]. Recognizing handwriting is difficult for computers. It can even be difficult for humans. Many students receive feedback on papers that they cannot decipher. A common solution to this problem is to show the bit of text to multiple people.

We wrote a passage with purposefully bad handwriting (Figure 1). Turkers were shown this image and offered $0.05 to make progress toward deciphering it. They were instructed to leave words they were unsure about in (parenthesis).

Selected iterations of this experiment are shown:

version 1:

You (?) (?) (?) (work). (?) (?) (?) work (not) (time). I (?) (?) a few grammatical mistakes. Overall your writing style is a bit too (phoney). You do (?) have good (points), but they got lost amidst the (writing). (signature)

version 4:

You (misspelled) (several) (words). (?) (?) (?) work next (time). I also notice a few grammatical mistakes. …

version 5:

You (misspelled) (several) (words). (Plan?) (spellcheck) (your) work next time. I also notice a few grammatical mistakes. Overall your writing style is a bit too phoney. You do make some good (points), but they got lost amidst the (writing). (signature)

version 6:

You (misspelled) (several) (words). Please spellcheck your work next time. I also notice a few grammatical mistakes. Overall your writing style is a bit too phoney. You do make some good (points), but they got lost amidst the (writing). (signature)

The final version has only four mistakes highlighted in light blue. According to our ground truth, these words should be "flowery", "get", "verbiage" and "B-" respectively. Some other words are still left in parentheses. Workers made good use of parentheses, and it is interesting to see how the words in them change between iterations.

4. TurKit

An overview of TurKit and related systems is shown in Figure 2. A programmer writes a JavaScript program that is executed by TurKit. TurKit stores information about the running program in the JavaScript database, so that it can restart if the system crashes.

TurKit can create Human Computation Tasks (HITs) on MTurk. These HITs may point to web pages supplied by the programmer. These web pages may access the JavaScript database before being displayed to turkers.

When turkers complete tasks, it is possible for the web server to store the results directly in the database, or pass the results back to MTurk. In the latter case, the program running in TurKit can retrieve the results from MTurk and store them in the database.

The programmer may retrieve results directly from the JavaScript database, or output them to a file.

A core component of TurKit is the *once* function, which stores information about a program's trace of execution so that when it is restarted, it can return to where it left off, without re-executing expensive code. The *once* function accepts a function as an argument, and will only execute this function once ever, in all runs of the program. If the function executes successfully (without throwing an exception), then the result is memoized so that subsequent runs of the program will not re-execute the function. A common use case for *once* is wrapping a function that creates a HIT on MTurk, and returns the HIT identifier.

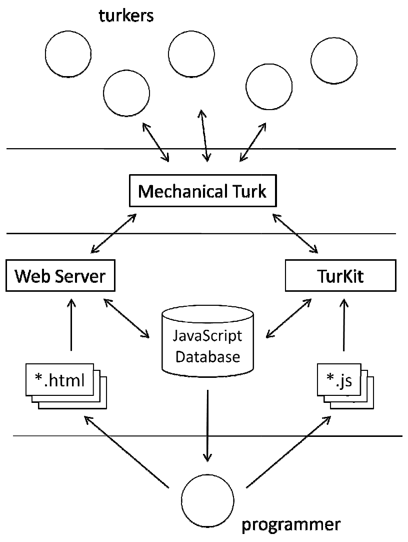


Figure 2: Overview of TurKit and related systems.

5. CONCLUSION AND FUTURE WORK

We have described TurKit, a new toolkit for programming iterative tasks on MTurk using a familiar imperative programming model, and applied it to a variety of example tasks. For future work, we plan to explore more complicated algorithms using TurKit, such as a parallel sort algorithm that is more robust to human comparison functions that may be noisy or only partially ordered. Also valuable to users of TurKit would be a detailed study of MTurk's properties as a programming system – latency, error rate, turker expertise, etc.

6. ACKNOWLEDGMENTS

We would like to thank everyone who contributed suggestions and ideas to this work, including Thomas W. Malone, Robert Laubacher, and members of the UID group. This work was supported in part by the National Science Foundation under award number IIS-0447800, by Quanta Computer as part of the TParty project, and by the MIT Center for Collective Intelligence. Any opinions, findings, conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the sponsors.

7. REFERENCES

[1] Luis von Ahn. Games With A Purpose. *IEEE Computer Magazine*, June 2006. Pages 96-98.

[2] Luis von Ahn, Ben Maurer, Colin McMillen, David Abraham and Manuel Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, September 12, 2008. pp 1465-1468.

[3] Kittur, A. and Kraut, R. E. 2008. Harnessing the wisdom of crowds in wikipedia: quality through coordination. *CSCW '08*. ACM, New York, NY, 37-46