Lecture 4:
CS 6306 / INFO 6306: Advanced Human Computation
Programming Languages
OpenOffice Contemplates Shutdown Amid Volunteer Drought

by David Z. Morris

SEPTEMBER 4, 2016, 12:49 PM EDT
Google expands 'Sheep View' project with Trekker cameras

Humans with camera-equipped backpacks, kayaks and wheelbarrows are joining the Faroe Island herd.

Without the presence of Google, Faroe Island residents had to rely on camera-equipped sheep to provide 360 degree "Sheep View" images of the bucolic isle. While that's adorable, digital tourists probably want to see more than just areas with the most delicious grass. Luckily, Google heard about their plight, and decided to give the burdened quadrupeds a hand with its Street View camera loan program.

The search giant flew in Trekker and 360 cameras, and an employee helped residents install them on trucks, bikes, backpacks, kayaks, horses and even...
Readings: Programming Languages

• Required readings:

• Additional readings:
  • Ahmad, S., Battle, A., Malkani, Z. and Kamvar, S., 2011. "The Jabberwocky programming environment for structured social computing.” In Proceedings of the 24th annual ACM symposium on user interface software and technology (pp. 53-64). ACM.
Readings: Programming Languages

• Additional readings:
  • From tasks to microtasks:
  • Tools:
    • Peer, Eyal, Gabriele Paolacci, Jesse Chandler, and Pam Mueller. "Selectively recruiting participants from Amazon Mechanical Turk using qualtrics." Available at SSRN 2100631 (2012).
  • Database perspective (next time):
Programming Languages and Environments for Micro-labor

What does software engineering look like when people are subroutines?

• Programming tools
• Debugging environments
• Runtime execution monitoring
• Design patterns
• Etc.
TurkIt

```javascript
ideas = []
for (var i = 0; i < 5; i++) {
    idea = mturk.prompt(
        "What’s fun to see in New York City?
        Ideas so far: " + ideas.join(" , "))
    ideas.push(idea)
}
ideas.sort(function (a, b) {
    v = mturk.vote("Which is better?", [a, b])
    return v == a ? -1 : 1
})
```

Figure 1: Naturally, a programmer wants to write an algorithm to help them visit New York City. TurKit lets them use Mechanical Turk as a function call to generate ideas and compare them.
TurkIt

• Javascript with API for MTurk operations

```javascript
function vote(message, options) {
    // create comparison HIT
    var h = mturk.createHITAndWait({
        ...message...options...
        assignments : 3}
    )

    // get enough votes
    while (...votes for best option < 3...) {
        mturk.extendHIT(...add assignment...)
        h = mturk.waitForHIT(h)
    }

    // cleanup and return
    mturk.deleteHIT(h)
    return ...best option...
}
```
TurkIt: Motivating Challenges

• Calling MTurk costs money
• “Runtimes” can span hours or days
• Restarts after crashes – don’t want to pay for redundant work
• Workers might not complete tasks

• “Crash-and-rerun programming model”
TurkIt: Once Primitive

```plaintext
quicksort(A)
  if A.length > 0
    pivot ← A.remove(A.randomIndex())
    left ← new array
    right ← new array
    for x in A
      if compare(x, pivot)
        left.add(x)
      else
        right.add(x)
    quicksort(left)
    quicksort(right)
    A.set(left + pivot + right)

compare(a, b)
  hitId ← createHIT(...a...b...)
  result ← getHITResult(hitId)
  return (result says a < b)
```
TurkIt: Once Primitive

• Stores result of function call in a database
• When rerun checks if the database already has the answer
• Programmer can clear database when program changes
• TurkIt detects when program and database are out of sync and crashes with appropriate error message
• Allows incremental programming – keep adding steps, saves earlier steps
• Can add retroactive print-line debugging
TurkIt: Other Primitives

• Crash:
  • A programmer-evoked exception
  • Example: Wait some period of time, then check if completed, if not crash – TurkIt
    waits, then reruns, and crashes again if not completed, etc.

• Fork:
  • Allows spawning a process that creates a parallel execution path
  • Useful for parallelizing MTurk HITs

• Join: Waits for parallel paths to complete

• mturk.prompt: Parameters = text to print, and number of workers

• createHIT: Calls once to create a hit

• waitForHIT: Crashes if HIT is not finished

• Providing HITs using more complex web pages hosted by TurkIt
Turklt: Issues Discovered

• Functions like `math.random` need to be in a once
• Confusion that it allowed parallelization
• Confusion on what was recorded / what could be modified when rerun
• Limited parallelization
• Last version: January 2013
TurkIt: Assessment

• Sample programs
• Runtime: Human vs computer
• Usability: anecdotal
AutoMan: Motivating Challenges

• Complex use of workers in programs
• Determining pay and time for tasks
• Quality of results
import edu.umass.cs.automan.MTurk_

object WhichOneNotBelongSimple {
    def main(args: Array[String]) {

        // AutoMan configuration for MTurk:
        val config = MTurkConfig { c =>
            c.access_key_id = "XXXX" // account info
            c.secret_access_key = "XXXX"
        }

        // Set up AutoMan parameters.
        val a = Automan { automan =>
            automan.budget = 8.00 // dollars
            automan.config = config // declared above
        }

        // Define a human function.
        val WhichOne = a.Task[String] { t =>
            t.confidence = 0.95 // the default
            t.title = "Which one of these doesn't belong?"
            t.description = t.title
            t.question = a.MultipleChoiceQuestion(
                question_text = t.title,
                selection_texts = Map('oscar -> "Oscar the Grouch",
                                       'kermit -> "Kermit",
                                       'spongebob -> "Spongebob Squarepants",
                                       'cookie -> "Cookie Monster",
                                       'count -> "The Count")
            )
        }

        // Call the human-based function.
        val fd = WhichOne()

        // Start execution and print result.
        a.run()
        println(fd.value)
    }
}
AutoMan Features

• Can specify desired confidence of results
• Randomizes choices in multiple choice questions
• Can specify timeout
  • Uses doubling strategy
• Can increase number of tasks spawned – pay more, better latency
• Automatically increases pay if HITs need longer time to accomplish
• Manages payments and non-payments
• Requires both positive and negative versions of questions
AutoMan Issues

• Worker collusion
• Impact of rejections
• Assumptions built into AutoMan
AutoMan: Assessment

- “Experience report”
- Graphs about quality management approach
TurkIt vs AutoMan
Jabberwocky: Software Stack for Human Computation

• Dormouse: Virtual machine – support different microlabor platforms
• ManReduce: Programming “framework” akin to MapReduce
• Dog: Database-looking programming language for ManReduce

• Assessment: Use examples
CrowdLang

• “Flowchart” for Human Computation

• Three components:
  • CrowdLang Library: Support code reuse
  • CrowdLang Engine: Runtime execution
  • CrowdLang Integrator: Virtual machine for human computation platforms
CrowdLang Approach

• Flowcharting primitives:
CrowdLang: Assessment

• Can express different human computation design patterns

• Could generate new combination that outperformed existing approaches on a translation task
Crowd Computer
(“Modeling, Enacting, and Integrating Custom Crowdsourcing Processes”)

• Using a business process management modeling and notation (BPMN) approach

• Elements:
  • Flexible CS platform for programming custom CS logics for individual and structured tasks
  • BPMN–based modeling language
  • Visual editor
  • Implementation on top of standard BPM technology
Crowd Computer: Requirements

**R1 Crowd tasks.** The crowdsourcer must be able to properly describe tasks and link them to external task pages.

**R2 Metadata exchange.** CS task pages must be enabled to exchange metadata with CC, so as to allow CC to coordinate tasks and propagate data.

**R3 CS tactics.** The tactics used to crowdsource a given task may differ from task to task. The crowdsourcer must therefore be able to design custom tactics, including custom quality assessment and rewarding logics.

**R4 Human tasks.** These are used when a task has to be executed by a designated human actor (not the crowd), such as the crowdsourcer or an external expert. Supporting human tasks is thus necessary to allow arbitrary human actors to participate in a CS process, such as to validate task outputs.

**R5 Machine tasks.** Similarly, it is necessary to support machine tasks that enable the integration of computations performed by a machine, such as an operation to compute the average of a series of data extracted from crowd-provided data.

**R6 Control flow.** Processes are composed of a set of tasks that need to be coordinated. It is necessary to be able to specify the order in which tasks are executed and possible decision points that allow one to split and merge the execution flow.

**R7 Dataflow.** Tasks may consume data as input and produce data as output. It is thus necessary to enable the crowdsourcer to clearly define which data are produced and consumed by which task and specify suitable data propagation logics.

**R8 Data management.** CS typically produces large amounts of data. Propagating data among tasks with different data constraints (e.g., show only three photos out of a given set of photos) requires being able to suitably cut, slice, merge, and format data.
Crowd Computer: Requirements

R9 Modeling language. The model-driven design of CS processes with the preceding features requires the conception of a formalism that allows the crowdsourcer to model her own CS processes (BPMN4Crowd).

R10 Modeling editor. To turn the modeling language into an instrument that can also be used in practice, it is necessary to equip the language with a suitable graphical editor that allows the crowdsourcer to model CS processes.

R11 Runtime environment. The execution of CS processes then requires the implementation of a runtime environment that must be able to cater to the preceding features. We call this environment Crowd Computer.

R12 Deployment. Turning a high-level process model into a running process requires the implementation of a dedicated model compiler and support for the automatic deployment of the generated artifacts.
Crowd Computer: Architecture
Crowd Computer: Architecture
Crowd Computer: Metadata Model
Crowd Computer: Process Diagram
Crowd Computer: (Simplified) Process Diagram of the Marketplace Tactic
**Crowd Computer: Assessment**

- **Case studies**
- **Comparison**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>crowd tasks</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>machine tasks</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>human tasks</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>social network tasks</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>tactics</td>
<td>marketplace</td>
<td>marketplace</td>
<td>marketplace</td>
<td>marketplace, social network</td>
<td>—</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Control flow support</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>single task</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>multi-instance task</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sequential execution</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>parallel execution</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>looping/iterative</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>execution</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>decision points</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>subprocess</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>—</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data management support</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>data splitting</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td>data aggregating</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>data passing among tasks</td>
<td>—</td>
<td>—</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>data hosting</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development support</th>
<th>wizard or manual</th>
<th>manual</th>
<th>wizard</th>
<th>wizard</th>
<th>manual</th>
</tr>
</thead>
<tbody>
<tr>
<td>task design</td>
<td>manual</td>
<td>automatic</td>
<td>automatic</td>
<td>automatic</td>
<td>automatic + manual</td>
</tr>
<tr>
<td>task deployment</td>
<td>—</td>
<td>JavaScript-like</td>
<td>custom processes</td>
<td>automatic</td>
<td>BPMN extension</td>
</tr>
<tr>
<td>process definition</td>
<td>—</td>
<td>language</td>
<td>via Python</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>language</td>
<td>—</td>
<td>—</td>
<td>wizard</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>process settings</td>
<td>—</td>
<td>JavaScript-like</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>language</td>
<td>—</td>
<td>language</td>
<td>vote task</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>validation support</td>
<td>—</td>
<td>—</td>
<td>consensus</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>control questions</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

CrowdForge: Motivation

- Dynamic partitioning by workers themselves
- Multi-level
- Complex workflows
- Multiple quality assurance approaches
- Aggregation can be either automated or by workers
CrowdForge: Motivation


• Analogizes between organizations and distributed computing
  • Partitioning work into parallelizable tasks
  • Mapping tasks to workers/processors
  • Managing the dependencies between tasks
  • Maintaining quality controls
CrowdForge

• Specify programs using a MapReduce model
• Break up tasks into task primitives:
  • Partition tasks into smaller subtasks
  • Map tasks performed by workers
  • Reduce tasks that merge work into a single result
Crowdforge Limitations

• No recursion or iteration
• Presumes decomposability that may not be achievable
• Subtasks may not be as independent as initially believed
• Map/Reduce framing may make things more complicated rather than less
CrowdForge: Assessment

• One case study
Turkomatic

- Writing divide-and-conquer programs where the workers define the program and execute it
Break down the task written in red.

Instructions: We are dividing a large task among several workers on Mechanical Turk. This is an experiment to see how complicated tasks can be shared between multiple workers on Mechanical Turk. Your job is to help us plan how this work will be divided.

Here is the task you are asked to divide:

Write a 3-paragraph essay about turtles

Do not solve this task yourself. Please break the task down into 2 or more simpler steps. Write each step in a box below. You can add more steps.

Each step you suggest will be posted to Mechanical Turk again for another Turk to do. Make sure each step will make sense to another Turk.

Here is what makes a good answer:

- Every step is a complete sentence or set of instructions.
- Each step contains all information required to do the task.
- Every step explains clearly what a Turk should do.
- Each step can be understood by itself without reading the original task written in red.

Tips:

- You can ask Turkers to host images and pictures on other sites, like http://imgur.com or http://youtube.com.

Your work will be checked for correctness before being approved.

Step 1

Step 2

Add Step  Remove Step
Solve a simple task

Instructions: We are dividing a large task among several workers on Mechanical Turk. This is an experiment to see how to break down large tasks. You are asked to do a small part of a large task that was planned by other workers.

The overall task: Make a lolcat.

Your task:

Take a photograph of your pet cat or your neighbor’s cat.

Your instructions: Please do this task and enter the solution in the box at the bottom of this page. You are free to include links to other images or videos you have uploaded online. If the instructions do not make sense, please take a look at the overall plan below and take your best guess.
Your goal is to find a solution to the following task highlighted in orange by combining the answers of other Turkers:

Write a 3-paragraph essay about turtles.

Other Turkers have suggested that this task can be broken into the steps written in green below. These steps have already been solved by other Turkers. Their solutions are written below.

Please combine the solutions written below into a single solution to the task written in orange. You should modify the solutions as necessary to better solve the task written in orange.

Sub-task 1: Research information about turtles.
Solution to sub-task 1:

Sub-task 2: Write a paragraph about turtles using the information you learned earlier.
Solution to sub-task 2:

Please enter your solution to the task in the box below.
Check the work of another Turker

We gave a Turker the following task:

Why human being is important in this society? where are we in the evolution circle? what we are contributing towards nature and what towards society and civilization

They gave the following answer:
Well, Human beings are important for evolution of the society, its growth and care etc. We are at the top of the evolution circle. We are contributing alot towards nature but in a negative way. We are destroying nature for our benefits. We are contributing alot towards society and civilization but at the cost of nature.

Was this a correct answer?
Answer carefully: your work will only be approved if your answer matches the majority of other Turkers.

- Yes
- No
Submit
Turkomatic
Turkomatic Challenges

- Success usually required requester intervention
- Starved tasks
- Derailed tasks
  - Bad work accepted
- Poor instructional writing
- One-size-fits-all model
Turkomatic: Assessment

• Three case studies
  • Tasks completed?
  • How many interventions?
SurveyMan

• (AMT often used for surveys/questionnaires)
• Challenges for surveys:
  • Question order effects
  • Question wording effects
  • Survey abandonment
  • Inattentive/random responses
• Observation: Analogous to programming
• Solution: Treat survey creation as programming
• Assessment: Case studies
Other Tools for AMT

• psiTurk:

• Qualtrics:
  • Peer, Eyal, Gabriele Paolacci, Jesse Chandler, and Pam Mueller. "Selectively recruiting participants from Amazon Mechanical Turk using qualtrics." Available at SSRN 2100631 (2012).

• TurkGate:
  • http://gideongoldin.github.io/TurkGate/

• TurkPrime:
Next Time: Database Perspective

• Required Readings:

• Additional Readings:
  • Davidson, S.B., Khanna, S., Milo, T. and Roy, S., 2013. “Using the crowd for top-k and group-by queries.” In Proceedings of the 16th International Conference on Database Theory (pp. 225-236). ACM.