ISSN: 2454-9940

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

E-Mail :
editor.ijasem@gmail.com
editor@ijasem.org

www.ijasem.org

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

An Opinion on AI with a Human Intermediary

Mr.G JACOB JAYARAJ[1], Mr.K.PRAVEEN[2] , Mrs.K.JYOTHI[3], Mrs.S.SIVA SUNAYANA[4]

Abstract

Computers still struggle or fail miserably at many things that people can do effortlessly. An unparalleled amount of human-based computing power may be harnessed using crowdsourcing platforms such as Amazon Mechanical Turk. But as a general-purpose computing platform, they aren't very useful. It is challenging to coordinate complicated or interconnected processes due to the absence of full automation. Adding human workers to the schedule in order to decrease latency is an expensive endeavor, and works need to be tracked and rescheduled when workers don't finish their assignments. The amount of time and money needed to complete a project is also not always easy to foresee. Lastly, human-based calculations may not always provide trustworthy results due to the fact that human abilities and accuracy differ greatly and employees have a financial motive to limit their effort. In this article, we present AUTOMAN, the pioneering technology for completely autonomous crowdprogramming. Human-based calculations are seamlessly integrated into a regular programming language with AUTOMAN as conventional function calls. These functions may be freely combined with traditional ones. Programmers using AUTOMAN are able to concentrate on the logic of their code thanks to this abstraction. A budget and degree of confidence in the total calculation may be defined in an AUTO-MAN software. The AUTOMAN runtime system takes care of scheduling, pricing, and quality control in an open and transparent manner. AUTOMAN keeps human workers on time, checks their progress, reprices their labor, and restarts them as needed to get the appropriate degree of confidence. It also optimizes parallelism among human workers while keeping costs down.

Keywords Crowdsourcing, Programming Languages, Qual- ity Control

1. Introduction

There are a lot of things that computers still can't do well that humans can. When it comes to vision, motion planning, and interpreting natural language, for instance, humans absolutely crush computers [22, 26]. The majority of academics believe that computers will continue to struggle with these "AI-complete" activities for some time to come [27].

Associate Professor[1], Professor[2], Asst.Professor[3,4]
CSE(DS)
DRK INSTITUTE OF SCIENCE & TECHNOLOGY

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

Hiring people to do computational jobs has been made easier by recent technologies. Amazon Mechanical Turk stands out as the most renowned example. It is a general-purpose crowd-sourcing platform that connects those looking for labor with those willing to do it [2, 15]. CastingWords and ClariTrans are domain-specific commercial services built on top of Mechanical Turk that accurately transcribe audio, while Tagasaurus and TagCow are domain-specific systems that classify images. The following obstacles, however, stand in the way of widespread and large-scale human-based computation:

• Tasks' duration and compensation are established. A task's allowed time and remuneration for completion must be decided in advance by employers. People won't take on projects with too little remuneration or too short of a deadline, so getting things right is challenging but crucial.

• Challenging scheduling. Companies need to strike a balance between the two competing priorities of cost (more employees equals more money) and latency (people are sluggish). Jobs need to be monitored and re-posted as needed to ensure workers don't miss their deadlines.

• Responses of poor quality. Because human workers' abilities and accuracy vary greatly and because they have a financial motive to reduce their labor, human-based calculations must always be double-checked. Because employees may reach an agreement at random, mere majority vote is inadequate, and manual verification is not scalable.

Contributions

The AUTOMAN programming system, which combines digital and human-based computing, is presented in this work. The difficulties of using large-scale human-based computing are tackled by AUTO-MAN:

Integrating human and digital computing in a transparent manner. Through the use of function calls in a conventional programming language, AUTOMAN integrates human-based computation. Managing budgets, schedules, and quality assurance is a breeze using the AUTO-MAN runtime system.

Planning and budgeting made easy. In order to keep costs down and make the most of human workers' parallelism, the AUTOMAN runtime system schedules jobs. When jobs aren't moving along as planned, AUTO-MAN may reschedule and re-price them.

Control of quality automatically. Manage quality control automatically using the AUTOMAN runtime system. AUTO- MAN generates a enough number of human jobs for every calculation to meet the programmer-specified confidence level.

For instance, AUTOMAN will first schedule a minimum of three jobs (human workers) when given a function with five potential solutions and a target confidence level of 95%. Given that the probability of all three parties reaching a unanimous decision is less than 5%, it would be appropriate to accept a unanimous answer. To reach a 95% confidence level, five out of six workers must agree; if all three workers disagree, AUTOMAN will schedule three more activities (Section 5).

2. Context: Platforms for Crowdsourcing

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

Given that crowdsourcing is a relatively new area of application for programming language research, we begin by providing a brief overview of the relevant literature on crowdsourcing platforms. While we'll be talking specifically about Amazon Mechanical Turk here, we can certainly draw parallels to other crowdsourcing services out there.

In order to facilitate temporary assignments, Mechanical Turk mediates between requesters—the employers—and workers—the turkers—the workers.

Tasks involving human intelligence (HITs). Human intelligence tasks (HITs) are the informal name for individual jobs on Mechanical Turk. A brief description, salary, and other data are included in HITs. On Mechanical Turk, the majority of HITs are for easy questions like "is this image a good fit for this product?" Companies often pay their employees a low wage because they believe that workers can do their tasks on a time scale that fluctuates between minutes and seconds. You may pay anything from a dime to a few dollars for HITs.

Metadata like a title, description, and search keywords are attached to each HIT, which is represented as a question form. Each HIT may include any number of questions. There are two main kinds of questions: those that allow employees to freely express themselves via text and those that require them to choose from a predetermined set of alternatives. The former are called open-ended questions, while the later are called closed-ended questions; at the moment, AUTOMAN is only compatible with closed-ended questions.

Posting HITs: A Requester's Perspective. You can manually upload HITs to Mechanical Turk, but you can also programmatically handle basic features of HITs using their web service API [2]. This includes posting HITs, collecting finished work, and paying workers. Posting several identical jobs to Mechanical Turk is a breeze using this API. It is possible to classify HITs into groups based on their shared characteristics.

Another way a requester might tell Mechanical Turk to paralize an HIT is by specifying whether or not each HIT should be allocated to several workers. Mechanical Turk makes sure that each parallel worker is distinct (i.e., that no worker may finish the same HIT more than once) by increasing the number of assignments, which enables more workers to accept work for the same HIT.

Employees: Carrying Out Tasks. Approximately 275,000 HITs are available for workers to pick from on Mechanical Turk; for more information on what qualifies them, check below. Workers select to execute a specific HIT by accepting an assignment, which reserves that task for them for a short period and prevents other workers from accepting it.

How Long Does an HIT Last? There are two timeout settings for HITs: the lifespan of the HIT, which specifies how long it should stay in the listings, and the length of the assignment, which specifies how long a worker has to do the task after it has been accepted. A worker's reservation for an HIT is canceled and the work is returned to the pool of available assignments if they exceed the assignment's time without submitting finished work. A HIT gets withdrawn from the job board and expires after its lifespan is over without any tasks being performed.

Requesters: Accepting or Rejecting Work. A worker will notify the requester whenever an assignment is finished and submitted. Once the task is finished, the requester has the option to approve or reject it. The worker is automatically compensated for their efforts when an assignment is accepted, indicating that the job is acceptable. Withdrawal of money occurs upon denial; if desired, the requester may include a textual explanation

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

for the rejection. See Section 3.2 for information on how AUTOMAN handles acceptance and rejection automatically.

Worker Quality Control. Getting excellent workers to participate in Mechanical Turk, or at least to stay away from poor workers, is a major difficulty when trying to automate 1 task. But re-5 questers can't find certain workers on Mechanical Turk.6

2. Overview

AUTOMAN is a domain-specific language embedded in Scala [24]. AUTOMAN's goal is to abstract away the de- tails of crowdsourcing so that human computation can be as easy to invoke as a conventional function.

2.1 Using AUTOMAN

Figure 1 presents an example (toy) AUTOMAN program. The program "computes" which of a set of cartoon characters does not belong in the group. Notice that the programmer does not specify details about the chosen crowdsourcing backend (Mechanical Turk) except for account credentials.

Crucially, all details of crowdsourcing are hidden from the AUTOMAN programmer. The AUTOMAN runtime manages interfacing with the crowdsourcing platform, schedules and determines budgets (both cost and time), and automatically ensures the desired confidence level of the final result.

***Initializing* AUTOMAN.** After importing the AUTOMAN and Mechanical Turk adapter libraries, the first thing an AUTOMAN programmer does is to declare a configuration for the desired crowdsourcing platform. This configuration is then bound to an AUTOMAN runtime object, which instanti- ates any platform-specific objects.

Specifying* AUTOMAN *functions. Functions in AUTO- MAN consist of declarative descriptions of questions that the workers must answer; they may include text or images, as well as a range of question types, which we describe below.

Confidence level. An AUTOMAN programmer can option- ally specify the degree of confidence they want to have in their computation, on a per-function basis. AUTOMAN's de- fault confidence is 95% (0.95), but this can be overridden as needed. The meaning and derivation of confidence is dis- cussed in Section 5.

```
import edu.umass.cs.automan.adapters.MTurk._

object SimpleProgram extends App {
  val a = MTurkAdapter { mt =>
    mt.access_key_id = "XXXX"
    mt.secret_access_key = "XXXX"
  }

  def which_one() = a.RadioButtonQuestion { q =>
    q.budget = 8.00
    q.text = "Which one of these does not belong?"
    q.options = List(
      a.Option('oscar, "Oscar the Grouch"),
      a.Option('kermit, "Kermit the Frog"),
      a.Option('spongebob, "Spongebob Squarepants"),
      a.Option('cookie, "Cookie Monster"),
      a.Option('count, "The Count")
    )
  }

  println("The answer is " + which_one()())
}
```

Figure 1. A *complete* AUTOMAN program. This program computes, by invoking humans, which cartoon character does not belong in a given set. The AUTOMAN programmer specifies only credentials for Mechanical Turk, an overall budget, and the question itself; the AUTOMAN runtime manages all other details of execution (scheduling, budgeting, and quality control).

Metadata and question text. Each question requires a title and description, used by the crowdsourcing platform's user interface. These fields map to Mechanical Turk's fields of the same name. A question also includes a textual representation of the question, together with a map between symbolic constants and strings for possible answers.

Question variants. AUTOMAN supports multiple-choice questions, including questions where only one answer is cor- rect ("radio-button" questions), or where any number of an- swers may be correct ("checkbox"

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

questions), as well as restricted-text entry forms. Section 5 describes how AUTO-MAN's quality control algorithm handles these different types of questions.

Invoking a function. An AUTOMAN programmer can in- voke a function as if it were any ordinary (digital) func- tion. Here, the programmer calls the just-defined function which_one() with no input. The function returns a Scala future object representing the answer, which can be passed to other Questions in an AUTOMAN program before the hu- man computation is complete. AUTOMAN functions execute in the background in parallel as soon as they are invoked. The program does not block until it references the function output, and only then if the human computation is not yet finished.

2.2 AUTOMAN Execution

Figure 2 depicts an actual trace of the execution of the pro- gram from Figure 1, obtained by executing it with Amazon's Mechanical Turk. This example demonstrates that ensuring valid results even for simple programs can be complicated.

Starting Tasks. At startup, AUTOMAN examines the form of the question field defined for the task and determines that, in order to achieve a 95% confidence level for a question with five possible choices, at minimum, it needs three different workers to unanimously agree on the answer (see Section 5). AUTOMAN then spawns three tasks on the crowdsourcing backend, Mechanical Turk. To eliminate bias caused by the position of choices, AUTOMAN randomly shuffles the order of choices in each task.

AUTOMAN's default strategy is optimistic. For many tasks, human workers are likely to agree unanimously. Whenever this is true, AUTOMAN saves money by spending the least amount required to achieve the desired statistical confidence. However, AUTOMAN also allows users to choose a more aggressive strategy that trades a risk of increased cost for reduced latency; see Section 4.3.

Quality Control. At time 1:50, worker 1 accepts the task and submits "Spongebob Squarepants" as the answer. Forty seconds later, worker 2 accepts the task and submits the same answer. However, twenty seconds later, worker 3 accepts the task and submits "Kermit". In this case, AUTOMAN's Scheduling Algorithm

Figure 3 presents pseudo-code for AUTOMAN's main sched- uler loop, which comprises the algorithms that the AUTO-MAN runtime uses to manage task posting, reward and time- out calculation, and quality control.

2.3 Calculating Timeout and Reward

AUTOMAN's overriding goal is to recruit workers quickly and at low cost in order to keep the cost of a computation within the programmer's budget. AUTOMAN posts tasks in rounds, which have a fixed timeout during which tasks must be completed. When AUTOMAN fails to recruit workers in a round, there are two possible causes: workers were not willing to complete the task for the given reward, or the time allotted was not sufficient. AUTOMAN does not distinguish between these cases. Instead, the reward for a task and the time allotted are both increased by a constant factor *k* every

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

```
wage = DEFAULT_WAGE
value_of_time = DEFAULT_VALUE_OF_TIME
duration = DEFAULT_DURATION
reward = wage * duration
budget = DEFAULT_BUDGET
cost = $0.00
tasks = []
answers = load_saved_answers()
timed_out = false
confident = false

while not confident:
  if timed_out:
    duration *= 2
    reward *= 2
    timed_out = false

  if tasks.where(state == RUNNING).size == 0:
    most_votes = answers.group_by(answer).max
    required = 0
    while min_votes(choices, most_votes + required)
        > most_votes + required:
      required += 1

    if required == 0:
      confident = true
    else
      can_afford = floor((budget - cost) / reward)
      if can_afford < required:
        throw OVER_BUDGET
      ideal = floor(value_of_time / wage)
      to_run = max(required, min(can_afford, ideal))
      cost += to_run * reward
      tasks.appendAll(spawn_tasks(to_run))

  else:
    num_timed_out = tasks.where(state == TIMEOUT).size
    if num_timed_out > 0:
      timed_out = true
      cost -= num_timed_out * reward
    foreach t in tasks.where(state == ANSWERED):
      answers.append(t.answer)
      save_answer(t.answer)
return answers.group_by(answer).argmax
```

Figure 2. Pseudo-code for AUTOMAN's scheduling loop, which handles posting and re-posting jobs, budgeting, and quality control; Section 5.2 includes a derivation of the formulas for the quality control thresholds.

time a task goes unanswered. k must be chosen carefully to ensure the following two properties:

1. The reward for a task should quickly reach a worker's minimum acceptable compensation (R_{min}), e.g., in a logarithmic number of steps.
2. The reward should not grow so quickly that it would give workers an incentive to wait for a larger reward, rather than work immediately.

Workers do not know the probability that a task will re- main unanswered until the next round. If the worker assumes even odds that a task will survive the round, a growth rate of $k = 2$ is optimal: it will reach R_{min} faster than any lower value of k, and workers never have an incentive to wait. Sec- tion 4.4 presents a detailed analysis. Lines 13-16 in Figure 3 increase the reward and duration for tasks that have timed out.

In AUTOMAN, reward and time are specified in terms of the worker's wage ($7.25/hour for all the experiments in this paper). Doubling both reward and time ensures that AUTOMAN will never exceed the minimum time and reward by more than a factor of two.

The doubling strategy may appear to run the risk that a worker will "game" the computation into paying a large sum of money for an otherwise simple task. However, once the wage reaches an acceptable level for some proportion of the worker marketplace, those workers will accept the task. Forcing AUTOMAN to continue doubling to a very high wage would require collusion between workers on a scale that we believe is infeasible, especially when the underlying crowdsourcing system provides strong guarantees that worker identities are independent.

2.4 Scheduling the Right Number of Tasks

AUTOMAN's default strategy for spawning tasks is optimistic: it creates the smallest number of tasks required to reach the desired confidence level if the results are unanimous. Line 19 in Figure 3 determines the number of votes for the most popular answer so far. Lines 20-23 iteratively compute the minimum number of additional votes required to reach confidence. If no additional votes are required, confidence has been reached and AUTOMAN can return the most popular answer (line 44).

Using the current reward, AUTOMAN computes the maxi- mum number of tasks that can be posted with the remaining budget (line 28). If the budget is insufficient, AUTOMAN will terminate the computation, leaving all tasks in an unverified state (lines 29-30). The computation can be resumed with an increased budget or abandoned. Mechanical Turk will au- tomatically pay all workers if responses are not accepted or

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

rejected after 30 days.

2.5 Trading Off Latency and Money

AUTOMAN also allows programmers to provide a *time-value* for the computation, which tells AUTOMAN to post more than the minimum number of tasks. AUTOMAN always schedules at least the minimum number of tasks required to achieve confidence in every round. If the programmer's time is worth more than the total cost of the minimum number of tasks, $\frac{\text{time value}}{\text{min wage}}$ tasks will be scheduled instead (lines 31-32). Once AUTOMAN receives enough answers to reach the specified confidence, it cancels any outstanding tasks. In the worst case, all posted tasks will be answered before AUTOMAN can cancel them, which will cost no more than time value · task timeout.

This strategy runs the risk of paying substantially more for a computation, but can yield dramatic reductions in latency. We re-ran the example program given in Figure 1 with a time- value set to \$50, 7× larger than the current U.S. minimum wage. In two separate runs, the computation completed in 68 and 168 *seconds*; we also ran the first computation with the default time-value (minimum wage), and those computations took between 1 and 3 *hours* to complete.

2.6 Derivation of Optimal Reward Growth Rate

When workers encounter a task with a posted reward of R, they may choose to accept the task, or wait for the reward to grow. Let p_a be the probability that the task will still be available after one round of waiting. We make the assumption that, if the task is still available after $i - 1$ rounds, then the probability that the task is available in the ith round is at most p_a. Hence, if the player's strategy is to wait i rounds and then complete the task,

$$\mathrm{E}[\text{reward}] \leq p^i k^i R,$$

since with probability at most p^i the reward will be $k^i R$ and otherwise the task will no longer be available.

Note that the expected reward is maximized with $i = 0$ if $k \leq 1/p_a$. Therefore, $k = 1/p_a$ is the highest value of k that does not incentivize waiting and will reach R_{min} faster than any lower value of k. Workers cannot know the true value of p_a. In the absence of any information, 1/2 will be used as an estimator for p_a and this leads to AUTOMAN's default value of $k = 2$.

However, it is possible to estimate p_a. Every time a worker accepts or waits for a task, we can treat this as an independent Bernoulli trial with the parameter p_a. The maximum likelihood estimator for p_a equals

$$\tilde{p}_a = \underset{x \in [0,1]}{\operatorname{argmax}}\, x^t (1 - x)^{n-t}$$

where t is the number of times a task has been accepted amongst the n times it has been offered so far. Solving this gives $\tilde{p}_a = t/n$.

The difficulty of accurately estimating p_a using *ad hoc* quality control is a strong argument for automatic budgeting. Implementing this estimation is a planned future enhance- ment for AUTOMAN.

3. Quality Control Algorithm

AUTOMAN's quality control algorithm is based on collecting enough consensus for a given question to rule out the possi- bility, with a desired level of confidence, that the

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

results are due to random chance. Section 5.3 justifies this approach.

Initially, AUTOMAN spawns enough tasks to meet the de- sired confidence level if all workers who complete the tasks agree on the same answer. Figure 5 depicts the initial confi- dence level function. Computing this value is straightforward: if k is the number of choices, and n is the number of tasks, the confidence level reached is $1 - k(1/k)^n$. AUTOMAN computes the lowest value of n where the desired confidence level is reached.

Question Variants. For ordinary multiple choice questions where only one choice is possible ("radio-button" questions),

k is exactly the number of possible answers. However, hu- mans are capable of answering a richer variety of question types. Each of these additional question types requires its own probability analysis.

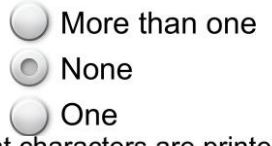
Checkbox Questions. For multiple choice questions with c choices and any or all may be chosen ("checkbox" questions), k is much larger: $k = 2^c$.

For these questions, k is so high that a very small number of workers are required to reject the null hypothesis (ran- dom choice). However, it is reasonably likely that two lazy workers will simply select no answers, and AUTOMAN will erroneously accept that answer is correct.

To compensate for this possibility, AUTOMAN treats

(a) A checkbox question. $k = 2^c$.

checkbox questions specially. The AUTOMAN programmer must specify not only the question text, but also an *inverted* question. For instance, if the question is "Select which of



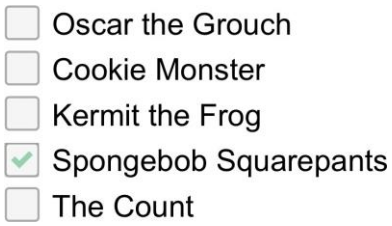


Figure 3. Question types handled by AUTOMAN.

3.1 Overview of the Quality Control Algorithm

In order to return results within the user-defined confidence interval, AUTOMAN iteratively calculates two threshold functions that tell it whether it has enough confidence to terminate, and if not, how many additional workers it must recruit. Formally, the quality control algorithm depends on two functions, t and l, and associated parameters α, β, and p. The $t(n, \alpha)$ and $l(p, \beta)$ functions are defined such that:

- $t(m, \alpha)$ is the threshold number of agreeing votes. If the workers vote randomly (i.e., each answer is chosen with equal probability), then the probability that an answer meets the threshold of $t(n, \alpha)$ when n votes are cast is at most α. α will be determined based on the confidence parameter chosen by the programmer (α = 1 - confidence).
- $l(p, \beta)$ is the minimum number of additional workers to recruit for the next step. If there is a "popular" option such that the probability a worker chooses it is p and $p > p$ (and all other options are equally likely), then if AUTO- MAN receives votes from $n \geq l(p, \beta)$ workers some answer will meet the threshold $t(n, \alpha)$ with probability at least $1 - \beta$.

workers who participated in previous instantiations of that task are excluded from future instantiations.

Our workaround for this shortcoming is to use Mechani- cal Turk's "qualification"

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

feature in an inverse sense. Once

Note that $E_1(n, n) = 1 - 1/k^{n-1}$ and define

$$t(n, \alpha) := \begin{cases} \min\{t : E_1(n, t) \geq \delta\} & \text{if } E_1(n, n) \geq \delta \\ \infty & \text{if } E_1(n, n) < \delta \end{cases}.$$

where $\delta = 1 - \alpha$. This ensures that when n voters each randomly chose an option, the probability that an option meets or exceeds the threshold $t(n, \alpha)$ is at most α.

Next we define

$$l(p^*, \beta) := \min\{n : E_2(p^*, n, t(n, \alpha)) \geq 1 - \beta\} .$$

If the voters have a bias of at least p^* towards a certain popular option, and all other options are equally weighted, then by requiring $l(p^*, \beta)$ voters, AUTOMAN ensures that the number of votes cast for the popular option crosses the threshold (and all other options are below threshold) with probability at least $1 - \beta$.

3.2 Quality Control Discussion

For AUTOMAN's quality control algorithm to work, two assumptions must hold:

- Workers are independent.
- Random choice is the worst-case behavior for workers; that is, they will not deliberately pick the wrong answer.

Workers may break the assumption of independence in three ways: (1) a single worker may masquerade as multiple workers; (2) a worker may perform multiple tasks; and (3) workers may collude when working on a task.

a worker completes a HIT that is a part of a larger compu- tation, AUTOMAN grants that worker special qualification (effectively, a "disqualification") that *precludes* them from participating in future tasks of the same kind. Our system ensures that workers are not able to request reauthorization.

Scenario 3: Worker Collusion. While it would be possible to attempt to lower the risk of worker collusion by ensuring that they are geographically separate (e.g., by filtering work- ers using IP geolocation), AUTOMAN currently does not take any particular action to prevent worker collusion. Prevent- ing this scenario is essentially impossible. Nothing prevents workers from colluding via external channels (e-mail, phone, word-of-mouth) to thwart the assumption of independence. Instead, the system should be designed to make the effort of thwarting defenses undesirable given the payout.

By spawning large numbers of tasks, AUTOMAN makes it difficult for any single group to monopolize them. In Mechanical Turk, no one worker has a global view of the system, thus the state of AUTOMAN's scheduler is unknown to the worker. Without this information, workers cannot game the system. The prevalent behavior is that people try to do as little work as possible to get compensated: previous studies of Mechanical Turk indicate random-answer spammers are the primary threat. [28].

3.2.1 Random as Worst Case

AUTOMAN's quality control function is based on excluding the possibility of random choices by workers; that is, workers who minimize their effort or make errors. It is possible that workers could instead act maliciously and deliberately choose incorrect answers. Participants in crowdsourcing systems have both short-term and long-term economic incentives to not deliberately choose incorrect answers, and thus random choice is a reasonable worst-case scenario.

First, a correct response to a given task yields an immedi- ate monetary reward. If a worker has any information about what the correct answer is, it is against their own short-term economic self-interest to deliberately avoid it. In fact, as long as there is a substantial bias towards the correct answer, AUTOMAN's algorithm will eventually accept it.

Second, while a participant might out of malice choose to forego the immediate economic reward, there are long- term

implications for deliberately choosing incorrect answers. Crowdsourcing systems like Mechanical Turk maintain an overall ratio of accepted answers to total answers submitted, and many requesters place high qualification bars on these ratios (typically around 90%). Incorrect answers thus have a lasting negative impact on workers, who, as mentioned earlier, cannot easily discard their identity and adopt a new one.

Anecdotal evidence from our experience supports these assumptions. Mechanical Turk workers have contacted us when AUTOMAN rejects their answers (AUTOMAN provides the correct answer in its rejection notice). Many workers sent us e-mails justifying their answers or apologizing for having misunderstood the question, requesting approval to maintain their overall ratio of correct to incorrect responses. We typically approved payment for workers who justified their incorrect answers, but Mechanical Turk does not allow us to accept already-rejected HITs.

4. System Architecture and Implementation

In order to cleanly separate the concerns of delivering reli- able data to the end-user, interfacing with an arbitrary crowd- sourcing system, and specifying validation strategies in a crowdsourcing system-agnostic manner, AUTOMAN is im- plemented in tiers.

4.1 Domain-specific language

The programmer's interface to AUTOMAN is a set of function calls, implemented as an embedded domain-specific language for the Scala programming language. The choice of Scala as a host language was motivated primarily by the desire to have access to a rich set of language features while maintaining compatibility with existing code. Scala is fully interoperable with existing Java code; the crowdsourcing system compatibility layer heavily utilizes this feature to communicate with Amazon's Mechanical Turk system. Scala also provides access to powerful functional language features that simplify the task of implementing a complicated system. These function calls act as syntactic sugar, strengthening the illusion that crowdsourcing tasks really are just a kind of function call with an extra error tolerance parameter. Scala was explicitly designed to host domain-specific languages [5]. It has been used to implement a variety of sublanguages, from a declarative syntax for probabilistic models to a BASIC interpreter [13, 23].

When using the AUTOMAN DSL, programmers first create an AutoMan instance, specifying a backend adapter, which indicates the crowdsourcing system that should be used (e.g., Mechanical Turk) and how it should be configured (e.g., user credentials, etc.). Next, the Question function is declared with the desired statistical confidence level and any other crowdsourcing backend-specific task parameters as required. When programmers call their Question function with some input data, the AUTOMAN scheduler launches the task asynchronously, allowing the main program to continue while the slower human computation runs in the background.

Since our aim was to make task specification simple, and to automate as many functions as possible, our Mechanical Turk compatibility layer provides sane defaults for many of the parameters. Additionally, we delegate control of task time- outs and rewards to AUTOMAN, which will automatically adjust them to incentivize workers. Maximizing automation allows for concise task specification for the common cases. When our defaults are not appropriate for a particular pro- gram, the programmer may override them.

4.2 Abstract Questions and Concrete Questions

The main purpose of the DSL is to help programmers construct Question objects,

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

which represent the system's promise to return a scalar Answer to the end-user. In re- ality, many concrete instantiations of this question, which we call ConcreteQuestions, may be created in the pro- cess of computing a single Question. The details of in- teracting with third-party crowdsourcing backends is han- dled by the AutomanAdapter layer, which describes how ConcreteQuestions are marshalled.

AUTOMAN controls scheduling of all ConcreteQuestions in the target crowdsourcing system. After programmers have defined a Question, they can then call the resulting object as if it were a standard programming language function. In other words, they provide input as arguments to the function, and receive output as a return value from the function, which can be fed as input to other tasks as desired.

From this point on, AUTOMAN handles communication with the crowdsourcing backend, task scheduling, quality control, and returning a result back to the programmer un- der budget and in a timely manner. Question threads are implemented using Scala Futures. After calling a Question, program control returns to the calling function, and execu- tion of the human function proceeds in the background, in parallel.

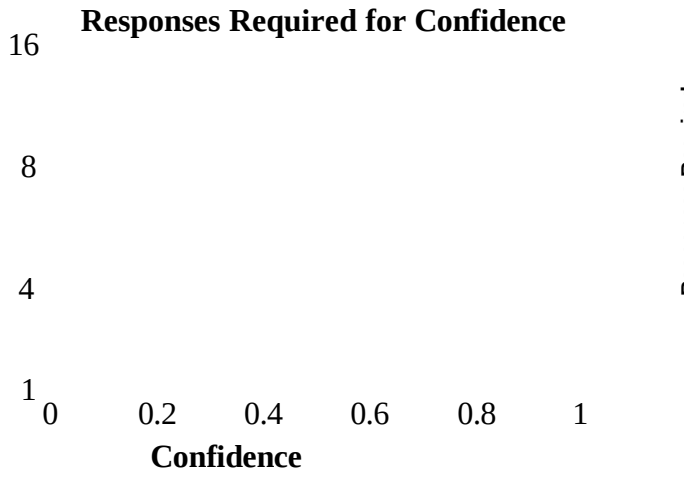


Figure 4. 10000 random sequences of worker responses to a five-option radio button question were used to simulate AUTOMAN's quality control algorithm at each confidence value. The trace lines use real worker responses to the "Which one does not belong?" application described in Section 7.1, while the 33%, 50%, and 75% lines were generated from synthetic traces where 33%, 50%, and 75% of workers chose the correct response, respectively. These graphs show that AUTOMAN is able to maintain the accuracy of final answers even when individual workers have low accuracy. Increasing confidence and decreasing worker accuracy both lead to exponential growth in the number of responses required to select a final result.

4.3 Memoization of Results

AUTOMAN's automatic memoization stores Answer data in a lightweight Apache Derby database. Implementors of third party AutomanAdapters must provide a mapping be- tween their concrete Answer representation and AUTOMAN's canonical form for Answer data.

If a program halts abnormally, when that program is resumed, AUTOMAN first checks the memoization database for answers matching the program's Question signature before attempting to schedule more tasks. If a programmer changes the Question before restarting the program, this signature will no longer match, and AUTOMAN will treat the Question as if it had never been asked. Any future use of a memoized function amortizes the initial cost of the function by reusing the stored value, and as long as the programmer preserves the memo database, reuse of memoized functions works across program invocations, even for different programs.

It is incumbent on the user to ensure that they define side- effect-free AUTOMAN functions. Scala does not currently provide a keyword to enforce functional purity.

4.4 Validation strategies

The manner in which jobs are scheduled and

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

errors handled depends on the chosen ValidationStrategy. If a strategy is not specified, AUTOMAN automatically uses the validation routines in the DefaultStrategy, which performs the form of statistical error handling we outlined in earlier sections. However, in the event that more sophisticated error handling is required, the programmer may either extend or completely replace our base error-handling strategy by implementing the ValidationStrategy interface.

4.5 Third-party implementors

Implementors who wish to adapt the AUTOMAN runtime for additional crowdsourcing backends need only implement the AutomanAdapter and ConcreteQuestion interfaces. Pro- grams for one crowdsourcing backend thus can be ported to a new system by including the appropriate AutomanAdapter library and specifying the proprietary system's configuration details.

5. Evaluation

We implemented three sample applications using AUTOMAN: a semantic image-classification task using checkboxes, an image-counting task using radio buttons, and an optical character recognition (OCR) pipeline. These applications were chosen to be representative of the kinds of problems which remain difficult even for state-of-the-art algorithms.

5.1 Which one does not belong?

Our first sample application asks users to identify which object does not belong in a collection of items (Figure 1). This kind of task requires both image- and semantic-classification capability, and is a component in clustering and automated construction of ontologies. Because tuning of AUTOMAN's parameters is largely unnecessary, relatively little code is required to implement this functionality (about 20 lines).

We gathered 93 responses from workers during our sam- pling runs. Runtimes for this program were on the order of minutes, but there is substantial variation in runtime given

```
import edu.umass.cs.automan.adapters.MTurk._
import com.amazonaws.services.s3.AmazonS3Client
import java.awt.image.BufferedImage
import java.io.File

object HowManyThings {
  def main(args: Array[String]) {
    val a = MTurkAdapter { mt =>
      mt.access_key_id = "XXXX"
      mt.secret_access_key = "XXXX"
      mt.sandbox_mode = true
    }
    def how_many(url: String) = a.RadioButtonQuestion { q=>
      q.text="How many "+args(0)+" are in this image?"
      q.image_url = url
      q.options = List(
        a.Option("zero", "None"),
        a.Option("one", "One"),
        a.Option("more", "More than one")
      )
    }
    // Search for a bunch of images
    val urls = TinyImageSearch.search(args(0))
    // download each image
    val images = urls.map(download_image(_))
    // resize each image
    val scaled = images.map(resize(_))
    // store each image in S3
    val s3 = new AmazonS3Client(new BasicAWSCredentials("XXXX","XXXX"))
    val s3_urls = scaled.map{ i =>
      store_in_s3(i, s3, bucket)
    }
    // ask humans for answers
    val answers_urls = s3_urls.map { url =>
      (how_many(getTinyURL(url.toString)) -> url)
    }
    // print answers
    answers_urls.foreach { case(a,url) =>
      println("url: " + url
        + ", answer: " + a().value)
    }
  }
}
```

Figure 5. An application that counts the number of searched- for objects in the image. Amazon S3, Google, TinyURL, and image-manipulation helper functions have been omitted. the time of the day. Demographic studies of Mechanical Turk have shown that the majority of workers on Mechanical Turk are located in the United States and in India [16]. These find- ings largely agree

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

with our experience, as we found that this program (and variants) took upward of several hours during the late evening hours in the United States.

Results from this application were used to test AUTO- MAN's quality control algorithm at different confidence lev- els. AUTOMAN ensures that each worker's response to this application is independent of all other responses. Because responses are independent, we can shuffle the order of re- sponses and re-run AUTOMAN to approximate many different runs of the same application with the same worker accuracy. Figure 6 shows the average accuracy of AUTOMAN's final answer at each confidence level, and the average number of responses required before AUTOMAN was confident in the final answer. Figure 6 also includes results for three sets of synthetic responses. Synthetic traces are generated by return- ing a correct answer with probability equal to the specified worker accuracy (33%, 50%, and 75%). Incorrect answers are uniformly distributed over the four remaining choices.

These results show that AUTOMAN's quality control is highly pessimistic. Even with extremely low worker accuracy, AUTOMAN is able to maintain high accuracy of final results. Real worker responses are typically quite accurate (over 80% in this case), and AUTOMAN rarely needs to exceed the first two rounds of three questions to reach a very high confidence.

5.2 How many items are in this picture?

Counting the number of items in an image also remains difficult for state-of-the-art machine learning algorithms. Machine-learning algorithms must integrate a variety of feature detection and contextual reasoning algorithms in order to achieve a fraction of the accuracy of human classifiers [26]. Moreover, vision algorithms that work well for all objects remain elusive.

This kind of task is trivial in AUTOMAN. We set up an image processing pipeline using the code in Figure 7. This application takes a search string as input, downloads images using Google Image Search, resizes the images, uploads the images to Amazon S3, ambiguates the URLs using TinyURL, and then posts the question "How many $items are in this image?"

We ran this task 8 times, spawning 71 question instances, and employing 861 workers, at the same time of the day (10 a.m. EST). AUTOMAN ensured that for each of the 71 questions asked, a worker was not able to participate more than once. We found that the mean runtime was 8 minutes, 20 seconds and that the median runtime was 2 minutes, 35 seconds. Overall, the typical task latency was surprisingly short.

The mean is skewed upward by the presence of one long- running task which asked "How many spoiled apples are in this image?". The difference of opinion caused by the ambiguity of the word "spoiled" caused worker answers to be nearly evenly distributed between two answers. This ambiguity forced AUTOMAN to collect a large number of responses to be able to meet the desired confidence level. AUTOMAN handled this unexpected behavior correctly, running until statistical confidence was reached.

5.3 Automatic number plate recognition (ANPR)

Our last application is a reimplementation of a common—and controversial—image recognition algorithm for automatic number plate recognition (ANPR). ANPR is widely deployed using distributed networks of traffic cameras. Academic literature on the subject suggests that state-of-the-art systems can achieve accuracy near 90% under ideal conditions [11]. False positives can have dramatic negative consequences in unsupervised ANPR systems as tickets are issued to motorists

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

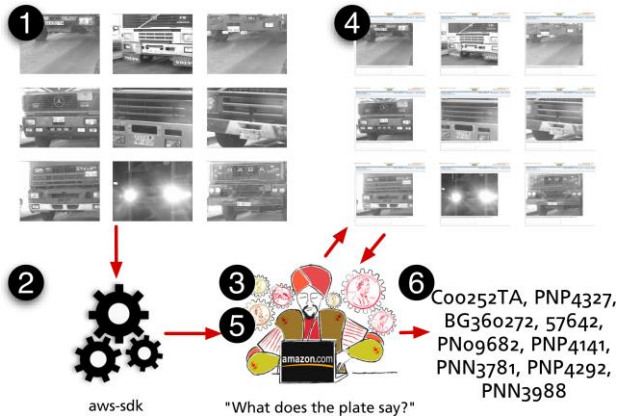


Figure 6. A number-plate recognition program. Amazon S3, command-line option parsing, and other helper functions have been omitted for clarity.

automatically. A natural consequence of this fact is the need for human supervision to audit results and limit false positives.

Figure 8 shows an ANPR application written in AUTO-

(a) Visual representation of the license-plate-recognition workflow. 1) Images are read from disk. 2) Images are uploaded to Amazon S3. 3) HITs are posted to MTurk by AUTOMAN. 4) Workers complete the posted HITs.
5) Responses are gathered by AUTOMAN.
6) AUTOMAN chooses the correct answers, repeating steps 3-5 as necessary, and prints them to the screen.

(b) A sample HIT on Mechanical Turk for OCR. In all of our trial runs, AUTOMAN correctly identified this hard-to-read plate.

quality control ensures that it delivers results that match or exceed the state-of-the-art on even the most difficult cases.

6. Related Work

Example Uses of Crowdsourcing. The computational power of many software applications is significantly en- hanced with human supervision at critical steps. The innate ability of humans to quickly and correctly provide answers to otherwise intractable problems has resulted in a great deal of interest in hybrid human-computer applications. We describe a representative sample of such applications below. Note

Feature	AUTOMAN	TurkIt [21]	CrowdFlower [25]	Jabberwocky [19]	Turkomatic	CrowdDB [14]
Quality Control Guarantee	X					
High Performance	X					
Automatic Budgeting	X					
Automatic Time Optimization	X					

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

Automatic Task Accept/Reject	X					
General Purpose	X	X	X	X	X	
Memoizes Results	X	X				
Interfaces with Existing	XJava	XJavascript		XRuby		XSQL
Code Type-Safe	X					
Platform Agnostic	X		*n/a*	X		

Table 1. A comparison between AUTOMAN and other crowdsourcing-based systems.

it doesn't depend on open labor markets-based general-purpose crowdsourcing platforms; rather, it only uses human computation. To find out which way faraway galaxies spin, Galaxy Zoo employs people as picture classifiers [20]. People who haven't had much practice classifying images submit their opinions (or "votes"), which are then ranked according to how accurate they were. Where conventional optical-character recognition algorithms have failed, reCAPTCHA repurposes the well-known CAPTCHA web-based "human Turing test" to categorize text pictures from scanned books [29]. With an estimated average accuracy surpassing 99%, reCAPTCHA has identified millions of phrases since the program was deployed.

CrowdSearch is a web-based, near-real-time picture search engine that uses human judgment to increase search accuracy [30]. Human workers are progressively and sequentially hired using arrival-rate estimations and a majority-voting system.

FoldIt! is an online game that challenges players to solve challenging protein-folding puzzles. By working together with cutting-edge protein-folding models, participants may traverse the state space of protein configurations more quickly than unsupervised algorithms [6].

Crowd Programming. Crowdsourcing services like Amazon Mechanical Turk have seen heavy ad hoc use, while programmatic worker management has received less attention. To facilitate the submission, tracking, and checking of tasks, Amazon Mechanical Turk offers a low-level API.

One way to streamline the administration of Mechanical Turk activities is via TurKit, a scripting system [21]. As an extension of JavaScript, TurKit Script provides a templating capability for typical Mechanical Turk activities and incorporates checkpointing to prevent the re-submission of failed scripts. To simplify Mechanical Turk jobs, the online application CrowdForge applies an abstraction similar to MapReduce [8, 18]. Partition, map, and reduce are the three main ways in which programmers break down jobs. Collecting results and assigning tasks to many people is handled automatically by CrowdForge. Automan handles scheduling, pricing, and other tasks automatically; TurKit and CrowdForge don't.

quality assurance; moreover, TurkIt's incorporation into JavaScript further restricts its applicability to applications requiring extensive computation.

By adding annotations to standard SQL queries, CrowdDB simulates crowdsourcing as an add-on to relational databases, allowing users to outsource database cleanup duties to the SQL runtime [14]. In order for SQL's query planner to minimize costly operations, the SQL runtime is crowdsourcing-aware. Because it is not a general-purpose computing platform, CrowdDB uses majority voting as its only quality control mechanism, in contrast to AUTO-MAN.

An whole calculation, including the "programming" of the job, is what Turkomatic hopes to crowdsource [19]. In Turkomatic, the system is given tasks in plain English and the runtime consists of two steps: mapping and reducing. Workers lay out a strategy for execution in the map stage, and the reduce step is when it all comes to a halt. Similar to AUTOMAN, Turko-matic can build computationally sophisticated systems with no limits. On the other side, Turkomatic isn't compatible with traditional programming languages and can't manage budgets or quality control.

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

An alternative to MapReduce for parallel programming, Jabberwocky offers a human-computation stack on top of ManReduce [1]. To facilitate programmers' interactions with Dormouse, a resource-management layer that schedules jobs on ManReduce, a Ruby DSL named Dog is available. Jabberwocky has a set price structure and a fixed, optional quality control system that is based on majority vote.

Checking for Defects in Quality. Commercial crowdsourcing systems are the focus of CrowdFlower, a closed-source online service [25]. In order to improve quality, CrowdFlower employs a "gold-seeding" strategy, which involves introducing questions into the question pipeline that already have known answers, in order to identify workers who are prone to make mistakes. In an attempt to alleviate the requester's load associated with gold-generation, Crowd-Flower adds techniques to automate produce this data via "fuzzing" while the system executes actual work. Finding previously unseen kinds of mistakes still need human intervention. Trust in the quality of a certain worker is the core of this strategy, as it is in previous work in this field [17]. Rather

AUTOMAN tackles the issue of job quality head-on, rather than relying on the assumption that one can predict how well a worker would do a new assignment based on their previous results.

In order to teach people to execute a certain job successfully, Shepherd facilitates direct feedback between task researchers and task workers, with the goal of increasing quality [10]. On the other hand, AUTOMAN doesn't need constant communication between requesters and workers.

A new method of document quality control called "find-fix-verify" is introduced by Soylent. There are three separate steps that may be crowdsourced: discovering faults, repairing them, and verifying the changes [4]. While AUTOMAN does not yet allow open-ended queries, Soylent is able to manage them. While AUTOMAN does provide quantifiable assurances on the output quality, Soylent's method does not.

7. Future Work

The current AUTOMAN prototype will serve as a foundation for future developments in the following areas:
Broader question classes. Question types supported by the current AUTOMAN system include limited free-text, radio-button, and checkbox options. Unrestricted free-text replies, often known as open-ended questions, will soon be a part of AUTOMAN's capabilities. An additional stage that relies on workers rating responses and then doing quality control on those rankings will be included into the validation process.
Extra tweaking that is automated. The current state of AUTOMAN prevents us from differentiating between two possible scenarios: one in which workers do not show up because we do not provide a strong incentive, and another in which workers are just unavailable because of the time of day. We should avoid increasing incentives in the second scenario as it would be ineffective. In a future release, we want to look at ways to modify this behavior.

Visualization tools. Although AUTOMAN conceals the management details of human-based computing behind an abstraction layer, it might be helpful to remove this layer during debugging to see the progress of activities. Currently, AUTOMAN has a basic logging system; but, when the number of tasks grows, it becomes more difficult to navigate the logs. Adding a web service that AUTOMAN programmers may use to access the system's tasks is our proposal to expand the AUTOMAN runtime system, which already functions as a server. Our early plans include making it possible to search for tasks that meet certain criteria and providing visualizations of the execution graph, which will include summaries.

Event planner. Currently, in order to debug AUTOMAN apps, one must run their program with the sandbox mode flag

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

enabled. This will cause Mechanical Turk to send tasks to their sandbox website, where developers may work on their programs. The number of separate workers needed to engage in a calculation might be large, thus although this is useful for previewing a job, it is not suitable for driving the behavior of the whole program. This sandbox functionality may not be present in other crowdsourcing backends also. We want to construct an application trace replayer that can mimic the backend with actual data and an event simulator that can produce worker replies from random distributions.

8. Conclusion

Computers still struggle or fail miserably at many things that people can do effortlessly. In this article, the pioneering crowdprogramming system, AUTOMAN, is introduced. Combining human and computer computing is what crowdprogramming is all about. Programmers may quickly incorporate human-based computation into their applications with the help of AUTOMAN, which automates quality control, scheduling, and budgeting.

.

References

S. Ahmad, A. Battle, Z. Malkani, and S. Kamvar are the authors of the study. Structured social computing using the Jabberwocky programming environment. This is the twenty-fourth annual conference on user interface software and technology, UIST '11, published in 2011 in New York, USA, on pages 53–64. The article is published by ACM and has the ISBN 978-1-4503-0716-1. You may access it online at http://doi.acm.org/10.1145/2047196.2047203.

Mechanical Turk is available on Amazon. Website created in June 2011 by Mturk. Anagnostopoulos, C. N. #3. LPR Database by MediaLab.

The authors of the cited work are [4] Bernstein, G., Little, R. C., Hartmann, B., Ackerman, D. R., Crowell, D., and Panovich, K. An Inside Crowd in a Word Processor: Soylent. In

The editors of UIST are K. Perlin, M. Czerwinski, and R. Miller. The pages numbered 313–322. 2010 by ACM. Has the ISBN number of 978-1-4503-0271;

5. This is the URL: http://dblp.uni-trier.de/db/conf/uist/ uist2010.html#BernsteinLMHAKCP10.

H. Chafi, Z. DeVito, A. Moors, T. Rompf, A. K. Sujeeth, P. Han-rahan, M. Odersky, and K. Olukotun are the authors of the cited work. Parallel Computing with Heterogeneous Languages via Virtualization. "Onward!" in 2010.

The authors of the statement are S. Cooper, F. Khatib, A. Treuille, J. Barbero, J. Lee, and M. Beenen.

F. Players, Z. Popovi, D. Baker, and A. Leaver-Fay. Utilizing a multiplayer online game for the projection of protein structures. Nature, 2010; 466(7307):756-760. The URL for the article is http://www.nature.com/doifinder/10.1038/nature09304.

Seventh, A. DasGupta's Probability: A Primer for Statistics and Machine Learning. Springer, 2011—1st edition.

For example, J. Dean and S. Ghemawat. "MapReduce" stands for "Simplified Data Processing on Large Clusters." Presented in OSDI, 2004. Pages 137–150.

[9] The Sybil Attack by J. R. Douceur. Chapters 251-260 from the 2002 London, UK, International Workshop on Peer-to-Peer Systems (ITPS '01) Revised Papers. Retrieved from http://dl.acm.org/citation.cfm?id=646334.687813, this publication is by Springer-Verlag and has the ISBN 3-540-44179-4.

ISSN: 2454-9940www.ijsem.org
Vol 14, Issuse.4 Dec 2020

The authors of the cited work are S. Dow, A. Kulkarni, B. Bunge, T. Nguyen, S. Klemmer, and B. Hartmann. Keeping the Crowd in Check: Leading and Instructing Those Working in Crowds. New York, NY, USA, 2011, on pages 1669-1674, in Extended Abstracts of the 2011 Annual Conference on Human Factors in Computing Systems (CHI EA '11). Published by ACM with the ISBN 978-1-4503-0268-5.

This file's DOI is http://doi.acm.org/10.1145/1979742.1979826.

Reference: [11] Due, Due, Ibrahim, Shehata, and Badawy, W. Automated License Plate Recognition (ALPR): A Current Horizons Assessment. 12th IEEE International Symposium on Video Technology, Paper No.

Volume 1, "An Introduction to Probability Theory and Applications" by W. Feller, published in 1968 by John Wiley & Sons Publishers, is the source in question.

BAYSICK, a Scala DSL that implements a subset of BASIC, was developed by M. Fogus [13]. In March 2009, it was published at https://github.com/fogus/baysick1.

14 M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Database for Crowdsourced Question Answering (CrowdDB). Articles 61–72 from the SIGMOD Conference, edited by T. K. Sellis, R. J. Miller, A. Kementsietsidis, and Y. Velegrakis. 2011 by ACM. ISBN 978-1-4503-0661-4. On page.

[15] Howe, J. (2016). Crowdsourcing: A Rising Star. June 2006, Wired Magazine, 14(6): 176–178. 1059-1028 is the ISSN number.

The Population of Mechanical Turk, by P. G. Ipeirotis [16]. New York University's Center for Digital Economy Research, Technical Report Working Paper CeDER-10-01, 2010.

[17]With J. Wang, F. Provost, and P. G. Ipeirotis. Oversight of Amazon Mechanical Turk quality. Presented at the 2010 ACM SIGKDD Workshop on Human Computation (HCOMP '10) in New York, USA, on pages 64–67. Modern Computer Society. ISBN

https://doi.acm.org/10.1145/1837885. ISBN: 978-1-4503-0222-7.

the DOI: http://doi.acm.org/10.1145/1837885. Article number 1837906.

1,383,906 euros.

[18] Katur, A., Smus, B., and Kraut, R. E. CrowdForge: Finding Complex Work Through Crowdsourcing. Submitted to the Human-Computer Interaction Institute's Technical Report CMU-HCII-11-100 in February 2011 by Carnegie Mellon University's School of Computer Science.

[19]B. Hartmann, M. Can, and A. P. Kulkarni. Automation of Recursive Tasks and Workflows for Mechanical Turk: Turkomatic. New York, NY, USA, 2011, on pages 2053–2058, in Extended Abstracts of the 2011 Annual Conference on Human Factors in Computing Systems (CHI EA '11). publication by ACM, with the ISBN 978-1-4503-0268-5.