VirtualCrowd: A Simulation Platform for Microtask Crowdsourcing Campaigns

Sihang Qiu
Delft University of Technology
Delft, Netherlands
s.qiu-1@tudelft.nl

Alessandro Bozzon
Delft University of Technology
Delft, Netherlands
a.bozzon@tudelft.nl

Geert-Jan Houben
Delft University of Technology
Delft, Netherlands
g.j.p.m.houben@tudelft.nl

ABSTRACT

This demo presents VirtualCrowd, a simulation platform for crowdsourcing campaigns. The platform allows the design, configuration, step-by-step execution, and analysis of customized tasks, worker profiles, and crowdsourcing strategies. The platform will be demonstrated through a crowd-mapping example in two cities, which will highlight the utility of VirtualCrowd for complex crowdsourcing tasks in real world settings.

KEYWORDS

Crowdsourcing, Simulation, Discrete Event System

ACM Reference Format:
Sihang Qiu, Alessandro Bozzon, and Geert-Jan Houben. 2020. VirtualCrowd: A Simulation Platform for Microtask Crowdsourcing Campaigns. In *Companion Proceedings of the Web Conference 2020 (WWW '20 Companion), April 20–24, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3366424.3383546

1 INTRODUCTION

Crowdsourcing has become an important tool for scientific and industrial research. Despite the advances made in recent years toward a better theoretical characterization of crowdsourcing processes and actors, the presence of uncertainties in the output quality, execution time, and cost of execution campaigns [1, 4] is considered a main barrier for a more extensive adoption of crowdsourcing as a a research tool [6]. At the same time, such limitations hinder the ability to develop and test novel task scheduling and assignment strategies. In these settings, the need for readily available large amount of workers makes it difficult (if not impossible) to recreate execution conditions similar enough to allow for meaningful comparison of execution performance.

With these challenges in mind, we believe that the availability of tools and processes able to support large-scale crowdsourcing campaigns in a simulated (yet realistic) fashion can support researchers in the assessment of the cost-benefits trade-offs of different execution strategies. Simulations provide an algorithmic and systematic way to evaluate how a crowdsourcing campaign would progress, in the context of given simulation parameters. By modeling tasks and workers, and then adapting task and worker models in a discrete-event simulation system with customized parameters and strategies, properties such as quality, execution time, and cost could be estimated and compared.

Contribution In this demo, we present VirtualCrowd, a platform that enables the design, configuration, execution, and assessment of crowdsourcing campaigns through simulations. VirtualCrowd differs from previous work [2] in its ability to customize and extend relevant simulation parameters (e.g. worker quality and execution time) and execution strategies (e.g. worker selection and task assignment), enabling users to design and test complex crowdsourcing tasks with novel strategies.

VirtualCrowd is built upon a discrete-event simulation system that maintains an event queue with entities (e.g. workers and tasks) and their relations, where events are ordered by the time of the simulation clock. A Web-based user interface supports the management (creation, modification, and removal) of simulation instances. In each simulation instance, users can set its simulation parameters or customize strategies via the interface.

In the demonstration, we will showcase the usage of VirtualCrowd in the context of two crowdsourcing tasks, where we will simulate the annotation of urban objects (e.g. trees) in two cities (New York City and Amsterdam respectively). The simulation results and task assignment strategies are analyzed.

2 VIRTUALCROWD ARCHITECTURE

The architecture of VirtualCrowd is shown in Figure 1. The system consists of three main parts.

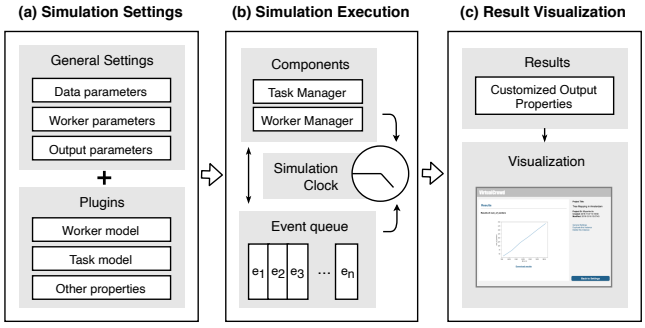
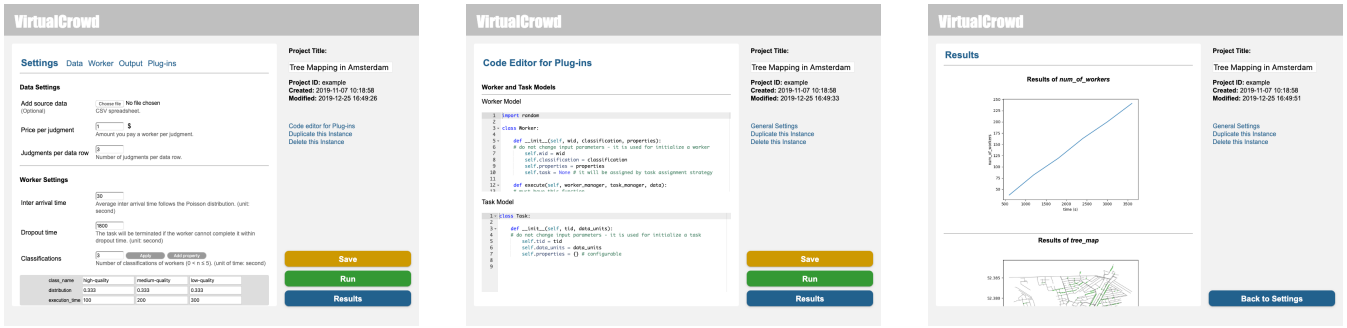


Figure 1: The architecture of the VirtualCrowd simulation system.

The *Simulation Settings* is responsible for the configuration of simulation parameters (constant values or probability distributions), worker/task models and customized strategies. The *Simulation Execution* manages simulation entities (e.g. tasks and workers) and components (e.g. task and worker managers), and executes simulation events according to the simulation clock. The *Analysis and Visualization* focuses on the processing and visualization of the simulation results. The code of VirtualCrowd is available for the benefit of the community [1].

[1] https://github.com/qiusihang/vcrowd



(a) General settings (b) Plugins settings (c) Simulation results

Figure 2: Examples of the user interfaces of the simulation platform.

2.1 Simulation Settings

The *Simulation settings* is operated on a Web-based interface, to facilitate the manipulation of configuration files processed by the *Simulation Execution*. By parsing these configuration files, the simulation system could read all the parameters including constant values, probability distributions, and customized properties. The welcome page of the interface supports managing simulation instances, where the simulation instances could be created, duplicated or deleted. Then users can set parameters of each simulation instance on the setting page, as shown in Figure 2.

2.1.1 General Settings. Configuration parameters including constant values and probability distributions that are common in the most crowdsourcing tasks can be configured via general settings. These parameters are usually estimated by real pilot tasks on crowdsourcing platforms.

The first category is **Data parameters**, i.e. the parameters about the original data provided to workers for task execution. For instance, original audio data for speech transcription, and original image data for segment annotation. *Data* can be uploaded as a spreadsheet file containing many data rows. Each data row represents an object to be assessed by workers, and the content of the data row can include.*Price per judgment* is the amount of money given to the worker per judgment. *Judgments per data row* is a number that, as the name suggests, indicate the number of unique worker needed per object.

Typical **Worker** configuration parameters are defined by default in VirtualCrowd, although new parameters can be added and customized via *Plug-ins*. *Inter arrival time* is the average time between each worker arrival. As common in queue theory, such parameter is best modeled as a Poisson distribution. *Dropout time* is the time limit given to a worker to finish a task. *Classifications* is the number of worker modeling *classes* that could be used to classify a worker. For each class of workers, the system allows the specification of 1) the *Class name*, the classes' 2) *Distribution*, and 3) the *Execution time*. Furthermore, quality metrics such as precision and recall can be added in the setting page as constant values. If the parameter follows some specific distributions, it should be defined as a plug-in as well.

Through **Output parameters**, parameters like the *Running time* and *Time step* of simulation executions can be configured. *Running time* defines the total running time (in terms of simulation clock) of one single execution. *Time step* is the time (of simulation clock) between each output action. At each output action, all the quality metrics will be re-calculated and recorded. The system will automatically calculate and visualize the value of relevant quality metrics defined via *Plug-ins*.

2.1.2 Plug-ins. Plug-ins allows for the definition of customized variables and functions written in Python 3. Users could add their own codes to enable the system become more adaptive and specific to their simulation needs. The following models or strategies might be customized.

1) *Worker model* and *Task model*. Common constants, distribution and actions have been configured in *General settings* part. However, sometimes parameters provided before cannot sufficiently describe a worker or a task. For instance, in a crowd sensing task, the mobility model of a worker in a physical space (virtual or real) is of great importance and it cannot be described by some simple distributions. 2) *Worker selection strategies*. These strategies evaluate the quality of the worker, to assess which worker is qualified for task execution. 3) *Task generation strategies*. We give users two options, if the user wants the task generation strategy to be random, only one parameter *Data rows per task* needs to be filled. Otherwise, if the user wants the strategy to be customized, a function should be specified. 4) *Task assignment strategy*. In terms of the task assignment, a customized function can also be provided. The queue of pending workers and the queue of pending tasks are the input of the function, users need to define the function to find the optimal task-worker pair(s) and return it as the output. 5) other customized variables and functions can be defined to support the system output and visualization.

2.2 Simulation Execution

The *Simulation Execution* is responsible for managing simulation components/entities, maintaining simulation clock, and executing simulation events. The simulation components, entities, events, and the pipeline of simulation execution are shown in Figure 3.

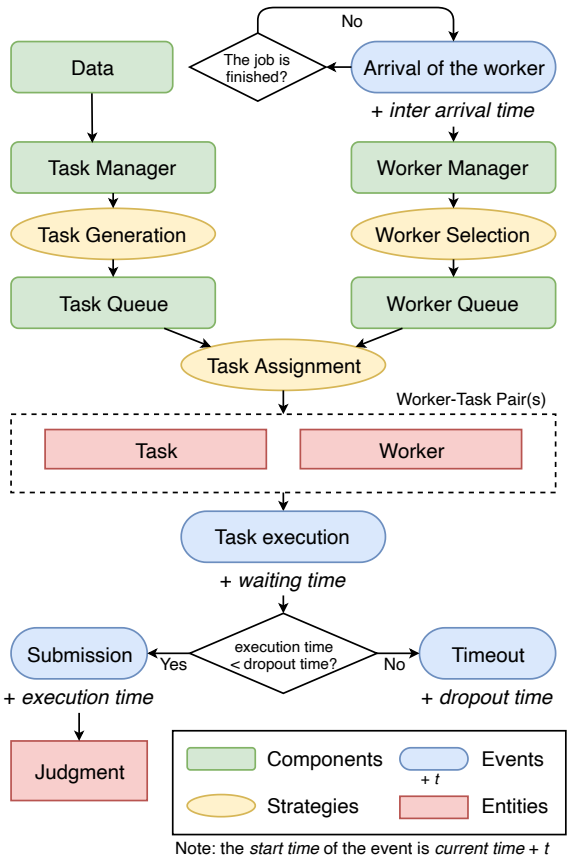


Figure 3: Simulation components, entities, events, and customized strategies used in the crowdsourcing simulation pipeline.

2.2.1 Components. After finishing simulation settings, simulation components will be created, including *data* composed by data rows, a *task manager*, and a *worker manager*. The *task manager* is used for managing all the tasks, especially for maintaining a *task queue* for storing pending tasks. It is also responsible for task generation and task assignment. Similarly, the *worker manager* for managing all the workers (where pending workers are waiting in a *worker queue*) is responsible for creating new workers, selecting qualified workers and disqualifying workers.

2.2.2 Entities. Simulation entities are items being created, updated and removed during the simulation execution. When the simulation execution begins, new entities (*workers* and *tasks*) will be generated and respectively pushed into the *worker queue* and the *task queue*. When a pair of task and worker has been assigned, they will be removed from corresponding queues. When a worker has finished a task and successfully submit the result, new *judgments* entities will be consequently created by generating values randomly drawn from the given distributions of quality metrics.

2.2.3 Events. While the simulation system attempts to create, update or remove entities, simulation events tell the system when and how to do it. A simulation event has two common properties. The first one is the start time, meaning the time of simulation clock when the event will be triggered; the second one is the type, which indicates how the simulation entities will be changed. Four types of simulation events are given. They are *arrival*, *task execution*, *submission*, and *timeout*. The task *arrival* event indicates that a new virtual worker is created; the *task execution* event reflects the successful assignment of a task to a worker, and that the worker is ready to execute the task; *submission* event indicates that the worker has been finished the task within dropout time, and judgments will be generated; *timeout* event means the worker does not finish the task in time and the task will be re-assigned. To ensure all the events are executed in chronological order, an event queue (realized by a heap) is used for maintaining pending events.

2.2.4 Pipeline. As can be seen in Figure 3, the pipeline of simulation execution composes of five steps:

1) Arrival of the worker. A new worker is created by the worker manager and then waits for worker selection. Meanwhile, the event of the arrival of the next worker will be pushed into the event queue; the start time is calculated by summing the current time with the inter arrival time. **2) Worker selection.** The quality of the new worker is evaluated. If the new worker is qualified, the worker will be push into the worker queue. **3) Task assignment.** The system uses the task assignment strategy to find the optimal task-worker pair(s) from the worker queue and the task queue. If the worker queue is empty, the system has to wait for the arrival of the next qualified worker. However, if the task queue is empty, the task generation strategy will be applied to generate new task(s). Once the an optimal task-worker pair is found, a task execution event will be pushed into the event queue, whose start time is calculated by current time plus waiting time. **4) Task execution.** the system will estimate the execution time according to pre-defined parameters, distributions and functions. If the execution time is less than *dropout time*, a submission event will be created with start time equaling to current time plus actual execution time. Otherwise, the worker will be dropped out without giving any judgments (i.e. timeout event). **5) Submission.** Judgments of data rows in the task will be generated according to the worker model and distributions of quality metrics.

2.3 Result Visualization

The *Result Visualization* focuses on the processing and reporting of the simulation results. Users can define output metrics about workers, tasks, judgments (quality), execution time, and cost. To record these metrics for visualization, users need to calculate them and return all their names with values as a dictionary in customized `output` function. During the simulation execution, these output metrics are recorded at each output time step. As shown in Figure 2, after the execution, line graphs will be plotted on the web-based interfaces, where execution time is the x-axis; pre-defined output metrics are the y-axis.

As the visualization part of the platform is developed based on `matplotlib`[2], users can plot not only line graphs, but also any other types of graphs supported by `matplotlib` (e.g. scatter plot,

[2]https://matplotlib.org/

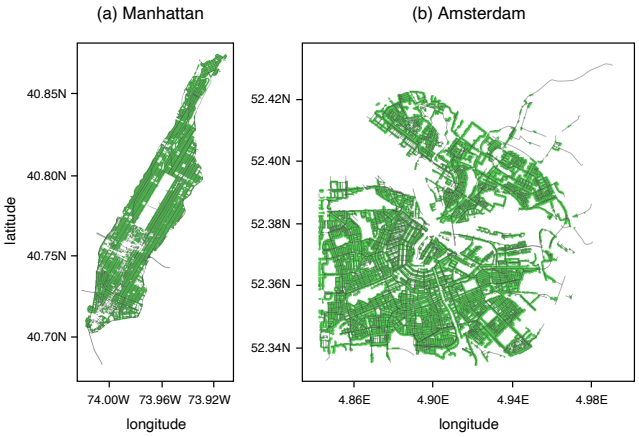


Figure 4: The tree maps of (a) Manhattan and (b) Amsterdam generated after the simulation, where lines are road networks and green points represent street trees.

heat map, bar graph, etc.). The customized graphs can be saved and shown on the platform by using provided functions. Furthermore, all the output data together with the parameters of simulation settings can be downloaded as an archive (.zip) file for further analysis.

3 DEMO HIGHLIGHTS

The We present the demonstration with a crowd-mapping example conducted in two cities. The crowd-mapping campaigns are usually conducted through crowdsensing platforms like ParticipAct [3]. This example focus on urban tree mapping in respectively Manhattan Borough (New York City) and Amsterdam. In these two crowdsourcing tasks, workers will be assigned tasks of finding and geo-locating street trees via street-level imagery platforms (like Google Street View) rather than exploring cities physically [7].

The source data, i.e. the street network of two cities, are retrieved from OpenStreetMap[3]. In each data row, a point and a polygon are contained, representing the starting point (the initial location) and the range of street segment (worker should find and locate trees in the polygon) respectively. Three classifications of workers are defined, which are low-quality, medium-quality, and high-quality workers. Parameters of worker model are measured according to a pilot task on Figure Eight. Annotating *Precision*, *Error* and *Recall* are measured as quality metrics. When a simulated worker annotates an object, the *Precision* parameter decides whether the annotation is correct or not; the *Error* (distance, unit: meter) between simulated annotations and the object follows normal distribution. The *Recall* parameter controls the number of correctly annotated trees in a segment. Two task assignment strategies (random assignment and quality-aware assignment [5]) are tested and compared.

Quality metrics are calculated using the ground truth data (the location of trees) collected from the *New York City Street Tree Map* project[4] and the dataset from the municipality of Amsterdam[5]. Tree maps of Manhattan and Amsterdam generated after the simulation are shown in Figure 4. Quality metrics ($\mu \pm \sigma$) calculated after five simulation executions are listed in Table 1. The average precision gain of quality-aware strategy against random strategy (respectively obtained 23% increase in Manhattan and 32% increase in Amsterdam) demonstrates the feasibility of applying quality-aware task assignment strategy in real world tree mapping campaigns.

Table 1: Results ($\mu \pm \sigma$) of five simulation executions.

City	Strategy	Precision (%)	Error (m)	Recall (%)
Manhattan	Random	0.44 ± 0.00	3.04 ± 0.01	0.87 ± 0.00
	Quality-aware	0.67 ± 0.01	2.89 ± 0.01	0.87 ± 0.00
Amsterdam	Random	0.55 ± 0.00	2.36 ± 0.01	0.83 ± 0.00
	Quality-aware	0.87 ± 0.00	2.18 ± 0.01	0.83 ± 0.00

Limitations. To use the plugins (customization), users need to understand how the discrete event system works and know how to use Python 3. Future work could focus on visualization and modular programming of the customization.

Implications. VirtualCrowd could have important implications in crowdsourcing task design. Prior crowdsourcing simulation platforms only consider common microtask types with a few parameters, where users cannot test their novel strategies. With VirtualCrowd, users are able to not only test traditional microtasking campaigns (such as image annotation, sentiment analysis, etc.) easily without plugins, but also customize worker-/task-related models and corresponding strategies to achieve high flexibility in different scenarios. The demo highlights the utility of VirtualCrowd in complex crowdsourcing settings.

REFERENCES

[1] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. R. Motahari-Nezhad, E. Bertino, and S. Dustdar. 2013. Quality Control in Crowdsourcing Systems: Issues and Directions. *IEEE Internet Computing* 17, 2 (mar 2013), 76–81. https://doi.org/10.1109/MIC.2013.20

[2] Alex Bores Ricart. 2013. CrowdSim: a Crowd Sourcing Simulation System. (2013).

[3] Giuseppe Cardone, Antonio Corradi, Luca Foschini, and Raffaele Ianniello. 2015. Participact: A large-scale crowdsensing platform. *IEEE Transactions on Emerging Topics in Computing* 4, 1 (2015), 21–32.

[4] Hector Garcia-Molina, Manas Joglekar, Adam Marcus, Aditya Parameswaran, and Vasilis Verroios. 2016. Challenges in Data Crowdsourcing. *IEEE Transactions on Knowledge and Data Engineering* 28, 4 (apr 2016), 901–911. https://doi.org/10.1109/TKDE.2016.2518669

[5] Katsumi Kumai, Masaki Matsubara, Yuhki Shiraishi, Daisuke Wakatsuki, Jianwei Zhang, Takeaki Shionome, Hiroyuki Kitagawa, and Atsuyuki Morishima. 2018. Skill-and-Stress-Aware Assignment of Crowd-Worker Groups to Task Streams. *Sixth AAAI Conference on Human Computation and Crowdsourcing* (jun 2018). https://aaai.org/ocs/index.php/HCOMP/HCOMP18/paper/view/17921

[6] Edith Law, Krzysztof Z. Gajos, Andrea Wiggins, Mary L. Gray, and Alex Williams. 2017. Crowdsourcing As a Tool for Research: Implications of Uncertainty. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '17)*. ACM, New York, NY, USA, 1544–1561. https://doi.org/10.1145/2998181.2998197

[7] Sihang Qiu, Achilleas Psyllidis, Alessandro Bozzon, and Geert-Jan Houben. 2019. Crowd-Mapping Urban Objects from Street-Level Imagery. In *The World Wide Web Conference*. 1521–1531.

[3] https://www.openstreetmap.org/.

[4] https://data.cityofnewyork.us/Environment/2015-Street-Tree-Census-Tree-Data/pi5s-9p35.

[5] https://maps.amsterdam.nl/open_geodata/.