Computational and Mathematical Organization Theory, **26**
on-line Jan. 24, 2020, https://doi.org/10.1007/s10588-019-09303-7

Multiscale Online Media Simulation with SocialCube

Tarek Abdelzaher*, Jiawei Han*, Yifan Hao*, Andong Jing*, Dongxin Liu*, Shengzhong Liu*, Hoang Hai Nguyen, David M. Nicol**, Huajie Shao*, Tianshi Wang*, Shuochao Yao*, Yu Zhang*, Omar Malik[1], Stephen Dipple[1], James Flamino[1], Fred Buchanan[2], Sam Cohen[2], Gyorgy Korniss[1], Boleslaw K. Szymanski[3]**

*Department of Computer Science

**Department of Electrical and Computer Engineering, University of Illinois

[1]Department of Physics
[2]Department of Information Technology and Web Science
[3]Department of Computer Science, Rensselaer Polytechnic Institute

ABSTRACT

This paper describes the design, implementation, and early experiences with a novel agent-based simulator of online media streams, developed under DARPA's Social-Sim Program to extract and predict trends in information dissemination on online media. A hallmark of the simulator is its self-configuring property. Instead of requiring initial set-up, the input to the simulator constitutes data traces collected from the medium to be simulated. The simulator automatically learns from the data such elements as the number of agents involved, the number of objects involved, and the rate of introduction of new agents and objects. It also develops behavior models of simulated agents and objects, and their dependencies. These models are then used to run simulations allowing future extrapolation and "what if" analysis. An interesting property of the simulator is its multi-level abstraction capability that allows modeling social systems at various degrees of abstraction by lumping similar agents into larger categories. Preliminary experiences are discussed with using this system to simulate multiple social media platforms, including Twitter, Reddit, and Github.

1 INTRODUCTION

The rise of online media in the last decade democratized information broadcast, creating unprecedented opportunities for reaching large segments of the population efficiently, as well as new vulnerabilities to disinformation spread that leverage the lack of efficient mechanisms for information vetting.

While human biases, interests, and values shaped information spread since primordial times, today's information overload (created by online media) makes it increasingly harder to grab human attention as there is more information to ingest and prioritize for possible propagation. Human biases and evolutionary information filtering instincts (such as preference for spreading threat-related information over neutral information) are being increasingly exploited to break through the attention barrier. Novelty and surprise are being using to embellish content for mass appeal. Social influence factors play a key role in assessing information credibility by recipients, as time to verify received information becomes increasingly limited. The above developments significantly impact the flow of information. Given fixed human cognitive capacity, as more information becomes accessible online, a progressively smaller fraction survives the human attention bottleneck. This higher selectivity changes the nature of information that can viably spread in the age of overload. Understanding these effects is a daunting undertaking. A potential first step is to build simulation tools that model human behavior in the age of information overload, allowing one to understand larger-scale emergent phenomena, such as the evolution of opinion dynamics, emergence of polarization, proliferation of echo-chambers, the spread of radicalization, and emergence of exploitable vulnerabilities of populations to certain types of organized disinformation campaigns. Such were the goals that motivated our research on simulation of online social media information dissemination.

Information objects on online media are diverse. They include posts on social networks, such as Twitter, publication entries in databases, such as DBLP, and software updates on version control services, such as GitHub. Collectively, streams of these continually created and updated objects comprise a reflection on beliefs (e.g., on Twitter), innovation (e.g., on GitHub), and publications (e.g., on DBLP). Simulations of sufficient fidelity can help one uncover properties of information exchange dynamics, and study large-scale phenomena such as competition (e.g., between ideas/beliefs, software toolchains, or publication pedigrees), popularity changes of objects, synergistic cross-pollination of innovations, and the way this landscape of synergies and competition mold online interactions.

The simulation tool described in this paper is a step towards answering questions, such as the above, regarding information dissemination on online media. Our simulator has three innovative features. First, it is highly retargetable. It can be adapted to simulate multiple different online media that share the abstractions of agents, (information) objects, and actions. Second, it is self-configuring. While a user can decide the configuration being simulated, it is possible to generate the configuration automatically from past traces of the target online medium. Finally, it is capable of multi-scale abstraction. Rather than simulating individual objects or agents, the tool can group similarly behaved objects or agents into larger categories thereby improving scalability while reducing potential for overfitting when training agent models from the finite available noisy data.

The rest of the paper is organized as follows. Section 2 summarizes the overall architecture of the simulator. Section 3 describes the modules responsible for agent simulation and clustering. Section 4 presents initial evaluation results. Related work is reviewed in Section 5. The paper concludes with Section 6.

2 ARCHITECTURE

Our online media simulations rely on three main abstractions: *agents*, information *objects*, and *actions*. Agents perform actions on information objects. These actions may influence other actions, which is how signals spread through the medium. For example, on Twitter, tweets constitute the information objects. Users constitute agents. Individual tweets can be created or re-posted (retweeted), which constitutes actions. Some actions (e.g., a tweet) may influence other actions (e.g., subsequent retweets). On GitHub, repositories are information objects. GitHub users are agents. Users can create a repository, fork it, commit content to it, or express interest in it ("star" it), among other capabilities that constitute actions. As with Twitter, some actions (e.g., creation of a new repository by a user) may influence other actions (e.g., forking of that repository by other users).

2.1 Design Features

Our simulation architecture supports three novel features; namely, it is (i) retargetable, (ii) self-configuring, and (iii) multiscale. These features are introduced below:

– **Retargetable simulation:** Our goal is to simulate multiple types of online media, centered around the abstractions of *agents*, *objects*, and *actions*. As described later in this paper, simulated configurations can be instantiated from data learned from media traces. To make the simulator retargetable, it is therefore important to convert media traces into a unified set of supported formats. This is the responsibility of a preprocessing step. Presently, it converts media-specific data structures into both a MySQL database and JSON files. Later modules can thus pick their preferred input format independently of the original source.

– **Self-configuring simulation:** Our online media simulator is novel in that it learns its own agent and object configuration and models from traces of the target online medium. An agent in the simulator is defined by a model that features several parameters. These parameters are learned by the *analysis library*. This library reads traces of online media (from either the MySQL database or the JSON files mentioned above) and computes statistics from which agent model parameters can be determined. Developers are free to implement and experiment with different kinds of machine learning or data mining algorithms, and integrate them into the analysis library to extract behaviors of different agents from the input data. These statistics constitute the *configuration file* needed for the simulator to operate.

- **Multiscale simulation:** The developed simulator is also novel in allowing multiple levels of abstraction. Since simulation models are automatically learned from data, overfitting can become a serious problem that impairs construction of appropriately generalizable agent and object models. To reduce the risk of overfitting, our agent (or object) modeling algorithms can decide to form categories of agents (or objects), in lieu of attempting to model them individually, with the added advantage that more data exists on the formed categories than on the underlying individual entities they comprise. Hence, such categories can be modeled more accurately than individual agents or objects. A side effect of such grouping is that it increases the scalability of simulation, thereby simultaneously improving both simulation speed (scalability) and accuracy (less overfitting).

When the simulator starts, an agent builder module reads the configuration files using a proxy that converts the stored configuration into a run-time instantiation of agent models. The simulator core module then runs the agent models and logs the observed behaviors.

2.2 Operating Principles

In a traditional agent-based simulation, it is important to accurately reproduce two elements of the simulated world; namely, (i) behavior models of individual *agents*, and (ii) the topology of the *network* connecting them. The behavior models decide what an agent will do, given prior actions of other agents (that it has visibility into) and the state of the environment. The network topology specifies which agents have visibility into actions of which other agents. Hence, when an agent performs an action, this action becomes visible to certain other agents (according to the network topology), who in turn may perform subsequent actions. Our simulator differs in two respects from the above model:

Agent/object duality: The simulator offers duality between agents and objects. In that sense, it can either run in an *agent-based* mode or *object-based* mode. The two are dual in that one models the world as agents interacting with objects, whereas the other models the world as objects interacting with (or rather being acted upon by) agents. This duality is depicted in Figure 1. In the agent-based mode, the main abstraction of the simulator is the agent. An agent model is described by (i) a total *activity rate* (i.e., rate of actions), (ii) *affinities* (set of objects it interacts with), and (iii) *interaction rates* (frequency of interaction with each object). The total activity rate can also be broken by action type (e.g., tweets versus retweets). A Twitter user might, for example, have different rates for original tweets versus re-tweets. Users who represent official entities such as local traffic alerts might post predominantly original tweets. In contrast, propaganda publicists might predominantly curate and re-post previously published news that favor their viewpoints, thus primarily posting retweets. The core simulation loop executes agents in (simulated) concurrency,

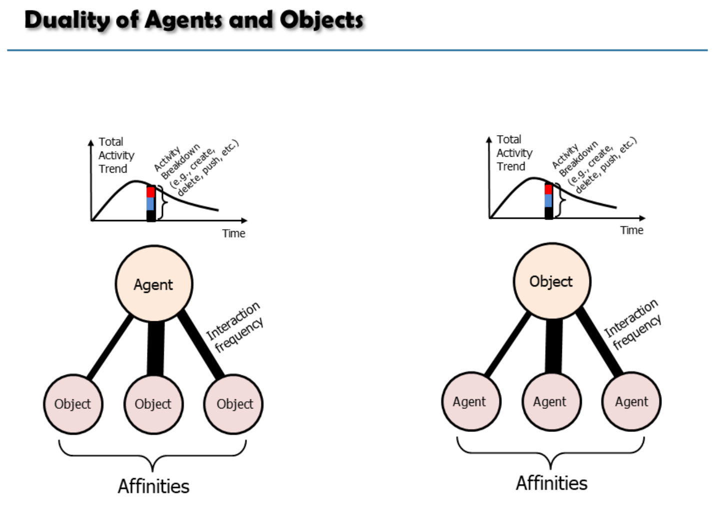


Fig. 1 SocialCube simulator architecture

one time-slot at a time. The model of each simulated agent is executed, producing the actions the agent will perform on objects during the simulated time interval. In the object-based mode, the main abstraction of the simulator is the object. The object model is given by (i) a total *activity rate* (i.e., rate of actions performed on the object), (ii) *affinities* (set of agents that interact with it), and (iii) *interaction rates* (frequency of interaction with each agent). The model of each simulated object is executed, one time slot at a time, producing the actions that agents will be perform on the object during the simulated time interval. A mixed mode is also possible. In this mode, agents are simulated first, generating potential actions, but without binding these actions to specific objects. The generated action counts are simply viewed as reflecting agent capacity or productivity. Objects are simulated next, producing actions performed on them but without binding those actions to agents. These action counts represent object popularity trends (e.g., in view of the degree of awareness that the object enjoys among agents). Actions generated from the aforementioned two respective models are then matched linking objects and agents consistently with previously observed object-agent affinities.

Aggregate trend simulation: The generation of actions in our simulator is not driven purely by agent or object models. Rather, it is also driven by aggregate trend models. These latter models capture certain macroscopic properties that are learned and predicted from observing the medium. An example is the total size of a cascade of reactions generated when a particular source performs a given action. Reproduction of macroscopic properties directly from previous observations makes the implicit assumption that the underlying factors impacting the property (such as network topology) are fairly stable over the period of simulation, or change smoothly. Thus, they need not be explicitly modeled. Rather, one can model the aggregate trend directly.

For example, cascades concerning objects of decreasing popularity will get smaller over time.

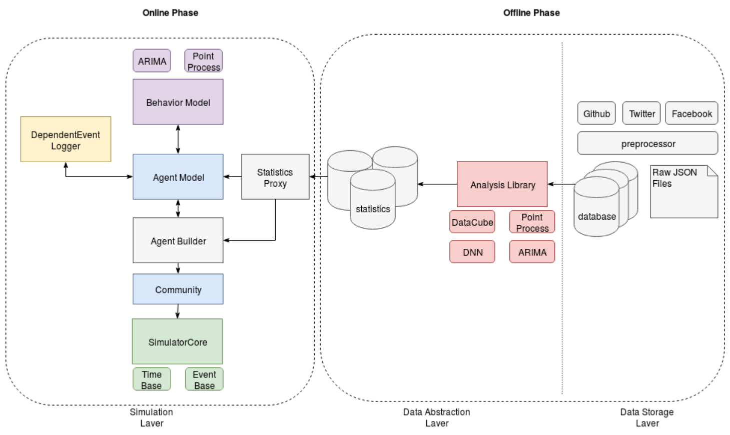


Fig. 2 SocialCube simulator architecture

2.3 Simulator Run-time

The simulator design and operation is shown in Figure 2. Once a configuration file has been generated, the simulation can start. The *statistics proxy* in Figure 2 serves as a bridge between the offline phase and the online phase. The proxy needs to understand the format of different analysis outputs. A new statistics proxy module needs to be integrated whenever a new type of analysis is implemented in the library. The architecture makes it easy to extend the simulator with new types of analysis techniques. For simplicity, below we describe an agent-based mode. Per the duality principle described above, the object-based mode is identical except for replacing “agent” with “object” and vice versa.

The agent *behavior model* is a collection of algorithms describing how the agent parameters described in the configuration file determine agent behavior. It is a place to simulate the behaviors and actions of an agent. A new behavior model is also needed whenever a new type of analysis (and hence new agent parameters) are introduced.

The *agent model* is a class that implements individual agents. Each agent will encapsulate a certain behavior model. Each agent (instance of the agent model class) also encapsulates the variables that are specific to the agent. It is essentially a stateful wrapper around the behavior model, whereas the behavior model describes the algorithm that interprets the variables.

The *agent builder* is a factory that instantiates agents during a simulation. When the simulator starts, the agent builder gets a list of agents from the statistics proxy. It then instantiates all the agents of the appropriate type and behavior model. The agent builder also creates new users on the fly according to a pattern dictated in the configuration file.

Note that, some agents represent communities. Like normal agents, they have behavior models, except that the generated actions do not share a single user ID, but rather are distributed across multiple identities. Communities can be used to cluster similar agents that share the same behavior model parameters.

The *event logger* records key past events in the simulation. Developers can adjust the depth of the logger, which in turn will adjust the memory of the simulation. Agent behavior models can condition behaviors based on events logged.

The *simulator core* will hold all the agents and communities after the agent builder instantiates them. The simulator core manages the passage of time, making calls to agents (with reference to current time and logged events) so they can be concurrently simulated.

3 SIMULATION ALGORITHMS

Simulation proceeds at two time-scales, called *global* and *local*, respectively. At the global time-scale, aggregate models are generated for phenomena such as object popularity, cascade size, and total agent activity. At the local time-scale, individual actions are generated in accordance with agent or object models. The global time-scale is slower. It adjusts aggregate parameters such as total activity rate and affinities by extrapolating trends described by auto-regressive time-series models. The local time-scale is shorter. It adjusts changes in probabilities of individual action types. Below, we describe these models in more detail.

3.1 Baseline: The “Replay” Model

The replay model is used as a special baseline that simultaneously captures both long-term and short-term behavior. It was helpful for evaluating the performance of other more advanced models. Specifically, if a model cannot beat this baseline, then it is not useful. Replay simply copies and pastes the events that happened in the training time period repeatedly into the testing time period until the event timestamp reaches the simulation interval boundary.

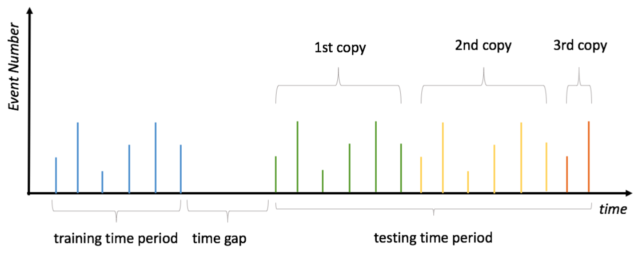


Fig. 3 The “replay” model. We copy and paste the events in the training time period to the testing time period three times and in middle of the 3rd copy, the event timestamp reaches the boundary so it stops.

The replay model is based on the assumption that the statistical properties of behaviors on the target social platform do not change much with the time. Even though this assumption is usually too strong, in some cases, it can achieve relatively good performance. One example is the GitHub platform, where the action type distribution (e.g., PushEvent, ForkEvent, WatchEvent ...) of each user remains stable over time. The replay model can well capture this aspect and approximate the ground truth. In fact, based on our observations, the replay model can achieve performance comparable to much more advanced models in GitHub simulation especially if the gap between training and testing time periods is small (less than 1 month).

A straightforward extension of the replay model is to observe that the behavior patterns of human agents are cyclical, often revolving around the structure of the week [1]. We observed this pattern for agents in multiple social media platforms, where most agents' activities would peak at certain hours of certain days, usually weekend afternoons. By defining cyclical periods with k subdivisions, the probability of agent i generating n events in the k^{th} time period could be modelled by the Poisson point process,

$$P_k^i(n) = \frac{(\Lambda_k^i)^n}{n!} e^{-\Lambda_k^i}, \tag{1}$$

where Λ_k^i is the expected number of events in period k for agent i. Unlike the basic replay model discussed earlier, the simple Poisson point process above can be extended for any simulation length, and accounts for cyclical variations in behavior.

3.2 Simulating Lifetimes: On Death and Birth of Agents and Objects

The most fundamental extension of the basic replay model is to simulate agent or object *lifetime*. The rate of actions performed by a particular agent or performed on a particular object changes over time because of changes in agent properties (e.g., total productivity, interest in a particular topic, work on a particular project, etc), as well as changes on object properties (e.g., object popularity).

Hence, when simulating information propagation on specific topics on online media, there is a constant flow of agents and objects entering and leaving the discussion over time. While the replay model, discussed earlier in the paper, can capture the overall volume of activity reasonably well, it ascribes the corresponding activities to the same group of agents and objects that it has already seen. We observe that agents' interests in a topic does not last indefinitely. A large fraction of the daily activity may involve new agents or objects. Therefore, it is important to accurately simulate the volume of agents and objects entering the system and the volume of those leaving it.

Below, let us focus on lifetime of agents. To simulate the lifetime of agents, we can modify the event generation probability in Equation (1) so that,

$$\Lambda_k^i(t) = \Lambda_k^i(0) f_i(t), \tag{2}$$

where $f_i(t)$ is some function such that $f_i(t \gg \tau_i) \approx 0$. In this case, beyond the lifetime parameter of an agent, the expected number of events goes to 0. This function varies widely across users and must be determined empirically from the data.

It is interesting to think of function, $f_i(t)$, as an *envelope* of activity. Within the envelope, different activity models can produce the actual actions. For example, if actions are independent, the Poisson model can be used. Section 3.3 describes different choices for computing the envelop function, $f_i(t)$. Section 3.4 describes microscopic activity models that fill out actions inside an envelope (in a manner more advanced than simply Poisson). Finally, to model the introduction of new agents into the system, an accurate simulator needs to determine the rate at which new agents are introduced and instantiate the corresponding affinities (objects they will access). This is described in Section 3.5. The resulting overall simulation framework is described in Figure 4. As shown in figure, the simulator models existing objects and agents, understands trends in their macroscopic activity envelopes, and generates fine-grained activities within each envelope. It also groups agents and objects semantically and generates new members of the identified groups.

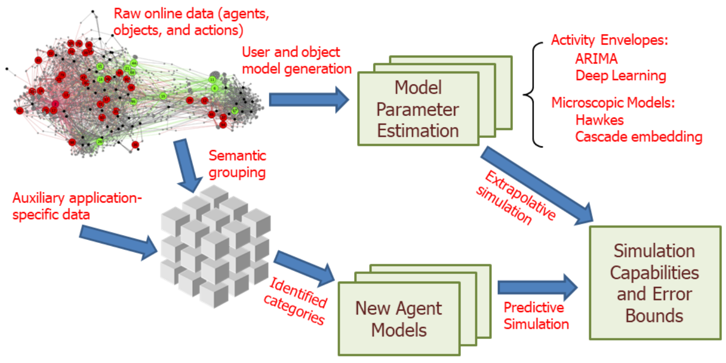

Raw online data (agents, objects, and actions)
User and object model generation
Model Parameter Estimation
Activity Envelopes:
ARIMA
Deep Learning
Microscopic Models:
Hawkes
Cascade embedding
Semantic grouping
Extrapolative simulation
Auxiliary application-specific data
Identified categories
New Agent Models
Predictive Simulation
Simulation Capabilities and Error Bounds

Fig. 4 An Extrapolative and Predictive Simulation Framework

3.3 Extrapolating Activity Envelopes: The Long-term Trends

These trends describe high-level tendencies in observed online media data. There are multiple ways such high-level tendencies can be extracted from the data. Below, we give two examples in increasing complexity.

3.3.1 ARIMA Models

We predict the long-term trends (*e.g.*, trends in total activity rate and affinities of agents) by applying time-series analysis models to extrapolate from observations. Here, we use the autoregressive integrated moving average (ARIMA) for the extrapolation.

We consider the aggregate parameters for each agent (or group of agents) a time series, and use X_t to denote the data vector at time slot t, where t is an integer. The

ARIMA model can be expressed by "$ARIMA(p,d,q)$", where p refers to the autoregressive terms, d refers to the order of differences in the model, and q refers to white noise terms.

We first introduce the difference [2] of a time series. Let X_t^d denote the d^{th} order difference of X_t:

$$X_t^d = \begin{cases} X_t, & if d = 0, \\ X_t^{d-1} - X_{t-1}^{d-1}, & if d \geq 1. \end{cases} \tag{3}$$

Then, the ARIMA(p,d,q) model can be written as:

$$X_t^d - \sum_{i=1}^{p} \phi_i X_{t-i}^d = c + \varepsilon_t - \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} \tag{4}$$

where c is a constant, the sequence $\{\varepsilon_t\}$ consists of independently identically distributed (iid) Gaussian noise terms with mean zero and variance σ^2, and ϕ_i and θ_j refer to the coefficients of X_{t-i}^d and ε_{t-j}, respectively.

The parameters of ARIMA can be learned by maximizing likelihood. Assuming, without loss of generality, that $c = 0$, the likelihood function for the parameters $\phi_1, \phi_2, \ldots, \phi_p$, $\theta_1, \theta_2, \ldots, \theta_q$ and σ is:

$$L(\phi, \theta, \sigma | X) = \frac{1}{\sqrt{2\pi\sigma^2 |\Sigma|^2}} \exp(-\frac{1}{2\sigma^2} x^T \Sigma^{-1} x), \tag{5}$$

where $\phi = [\phi_1, \phi_2, \ldots, \phi_p]^T$, $\theta = [\theta_1, \theta_2, \ldots, \theta_q]^T$, $x = [X_1^d, X_2^d, \ldots, X_n^d]^T$, $\Sigma = \sigma^{-2} cov(x)$, and n is the number of the training data samples. The exact form of $L(\phi, \theta, \sigma|X)$ is too complicated to maximize directly. Hence, several approximations and modifications were proposed [3–5]. After getting the parameters of the ARIMA model, we can predict the value of X_t^d using Equation (4).

For each agent (or group of agents), the analysis library builds an ARIMA model to predict the rate of each type of action and the interaction rates with individual objects over time.

3.3.2 Deep Neural Network Models

The ARIMA model works for specific agents, which means that we need to build an ARIMA model for each of the agents (or groups of agents). We also design a generalized model to predict the long-term trends of agents. It's a deep neural network (DNN) based model which takes the data of K consecutive time slots as input and then predict the corresponding data in the next K consecutive time slots. Similarly to the ARIMA model, we use

$$X_t = [X_{t1}, X_{t2}, X_{t3}, ..., X_{tn}]^T \tag{6}$$

to denote the data vector at time slot t. Here n refers to the dimension of the data vector. If we only predict the activity level of each agent, then $n = 1$, which means X_t is a scalar instead of vector. The DNN model takes the data vectors of consecutive K days as input. So the input vector is a K vector:

$$X_{in} = [X_{t1}, ..., X_{tn},\ X_{(t+1)1}, ..., X_{(t+1)n},\ ...,\ X_{(t+K)1}, ..., X_{(t+K)n}]^T. \tag{7}$$

The output of the DNN model X_{out} is similar to the input expect the time slots are between $t+K+1$ and $t+2K$ instead of $[t,t+K]$ in the input. The input vector X_{in}. There're multiple kinds of candidate deep neural networks (DNN), such as convolutional neural networks (CNN) and recurrent neural network (RNN). In our work, we build fully-connected deep neural network and an RNN model.

And we observe that the agents who have very few actions have very different long-term trend patterns with other agents. Hence, we divide all the agents into two parts: active agents (the activity levels of which are larger than a threshold) and inactive agents, according to their activity level. For those inactive agents, their actions are more likely in a random pattern and are difficult to predict. So we apply the "Replay" model to them. For those active agents, we apply the DNN model to them because we believe there exist long term patterns that can be studied by the DNN model.

3.4 Generating Fine-grained Activities: Microscopic Models of Cross-action Influence

Models described in the previous subsection capture macroscopic trends in user activity and object popularity. They do not capture microscopic relations between different actions. For example, it may be that user A and user B each send 100 emails a day on average. Macroscopic trends described above would reflect such daily counts. However, it may also be that user B always sends an email within a few minutes of receiving one from user A. such microscopic relations between individual actions associated with different agents or objects are further captured using the models described in this section. They constitute a refinement of traces generated by macroscopic models.

In our simulation, we separate *originality* and *independence*. An action could be original but not independent. For example, in a conflict, government supporters might issue an original statement describing the stare of affairs. An opposing party might respond by issuing an entirely different statement that contradicts that of the government. We say that both statements are *original* (as opposed to copied). However, they are not *independent*. Indeed, one statement might have incited the publication of the other.

In modeling influences among actions, we therefore separate original (although possibly non-independent) actions and copy-like reactions. An original action could be an original tweet on Twitter, post on Reddit, or paper entry in DBLP. A reaction might be a retweet (of an original tweet) on Twitter, a ditto comment (on an original post) on Reddit, or a citation (of an original paper) on DBLP. As noted above, original actions are not necessarily independent. For example, one tweet might trigger another different tweet. A paper might inspire a different follow-up publication. A post on one sub-reddit might incite a new different post on another sub-reddit.

We find that co-occurrence models and Hawkes processes, described in Section 3.4.1 and Section 3.4.2 below, are better-suited for capturing non-independence (i.e., triggering characteristics) among different original actions. In contrast, copy-like reactions are best modeled by cascade models, such as cascade embedding described in Section 3.4.3. The above distinction is because co-occurrence models and

Hawkes processes are designed to model dependencies among *heterogeneous* events or action types. In contrast, cascade models capture propagation of a *homogeneous* action type.

3.4.1 Co-occurrence: An Optimal Batch Model

Actions performed by agents on objects may be correlated due to external stimuli that one might not be directly monitoring. For example, two agents may act at the same time if they are both reacting to an exogenous event of interest to them, such as a change in price of a cryptocurrency.

Such exogenous connections create correlations that need to be captured by our microscopic models in order for the timing of actions to be properly generated. The total activity for each agent and the total popularity of each object might be modelled correctly, but these subtle timing correlations may be lost unless explicitly modeled as well. In order to quantify such temporal relationships, we need a cooccurrence model. For simplicity, consider an agent-centric model. We define the cooccurrence between two agent actions as:

$$\rho_{ij} = \frac{\sum_{t=0}^{T} min\left(e_i(t), e_j(t)\right)}{\sum_{t=0}^{T} e_i(t)}, \tag{8}$$

where $e_i(t)$ are the number of actions generated by some agent-based model for agent i at time t. For a given time interval, the cooccurrence, which is bounded by $[0,1]$, measures how many of agent i's actions occur in the same discrete time step as agent j's. Using this pairwise definition, we may define a global error quantity,

$$Er = \frac{1}{N^2} \sum_{i,j=0}^{N} \left|\rho_{ij} - \hat{\rho}_{ij}\right|, \tag{9}$$

where N is the total number of users, ρ_{ij} is the observed cooccurence between two agents, and $\hat{\rho}_{ij}$ is the simulated cooccurrence. Our goal is to move around each agent's actions until we arrive at a configuration that minimizes this error without changing the number of simulated actions. We also wish to preserve the probability distribution of the activity for each user.

For an agent with a lifetime of τ_i and P_i actions, there are $\binom{P_i}{\tau_i}$ ways of arranging their actions. Thus for N agents, there are $\prod_{i=0}^{N} \binom{P_i}{\tau_i}$ configurations. An exhaustive search through this space is computationally prohibitive. Therefore, we employ a genetic algorithm to search for a configuration which minimizes the error. The "individuals" in our algorithm were matrices of dimension $N \times T$, where each row represented an agent's time series of events. Then, we generated a population of matrices by randomly assigning each agent's simulated total actions over their lifetime. Thus, the sum of entries in each row was preserved, but that in each column was not. Each matrix was then scored according to Eq. (9) and pairs were randomly selected from the best scoring matrices. "Children" of these matrices were then created to replace the lowest scoring matrices. Children were created by randomly selecting rows from each parent. Each child could also be mutated, which involved a randomly selected row having its entries shuffled over the allowed columns.

3.4.2 Hawkes Process Models

The above batch manipulation may be time-consuming. Instead, it would be advantageous to define a streaming model that captures cooccurrence statistics. To do so, we use the temporal point process model, specifically the Hawkes process model, proposed by Hawkes [6]. Point processes provide the statistical language to describe the timing and properties of events [7]. They proved to be successful in many fields, including financial applications [8], geophysics analysis [9], and information propagation on online social media [10].

The Hawkes process is a natural extension of the well-known Poisson process. Instead of assuming that events (or, in our case, actions) happen independently as in a Poisson process, the probability of each coming event is dependent on the every past event in the Hawkes process (with a decay function that effectively results in finite memory). A Hawkes process sequence can be completely specified by the distribution of its inter-event times:

$$f(t_1,\ldots,t_n) = \prod_{k=1}^{n} f(t_k|t_1,\ldots,t_{k-1}) = \prod_{k=1}^{n} f(t_k|\mathscr{H}_{t_k}). \tag{10}$$

where $\mathscr{H}(t)$ denotes the event history before time t and $f(t_k)$ denotes the probability density function (PDF) of the kth event instant. To derive $f(t|\mathscr{H}(t))$, we need to define the conditional intensity $\lambda(t)$ first, which is heuristically defined as the arrival rate of new events:

$$\lambda = \lim_{\Delta t \to 0} \frac{E(N(t+\Delta t) - N(t)|\mathscr{H}_t)}{\Delta t} = \lim_{\Delta t \to 0} \frac{P(N(t+\Delta t) - N(t) = 1)|\mathscr{H}_t)}{\Delta t}. \tag{11}$$

where, $N(t)$ is the number of events up to time t. The relationship between the conditional intensity function $\lambda(t)$ and the probability density function $f(t|\mathscr{H}_t)$ is given by:

$$\lambda(t) = \frac{f(t|\mathscr{H}_t)}{1 - F(t)}. \tag{12}$$

where $F(t)$ is the cumulative distribution function (CDF) of the given event sequence up to time t. It is also the integral of $f(t|\mathscr{H}_t)$ from the start to time t. Therefore, the conditional intensity function is all we need in order to define a Hawkes process.

Now let's come back to our scenario. We want to use the point process model to describe the dependencies between actions of different types. For simplicity, consider heterogeneous actions (event types) performed by different agents on the same object. For each dimension (event type), i, the conditional intensity function is:

$$\lambda_i(t) = \mu_i + \sum_{j=1}^{D} \sum_{t_k^j < t} \phi_{ji}(t - t_k^j), \tag{13}$$

where the triggering kernel $\phi_{ji}(t)$ from event type j to event type i is an exponential decay function with respect to the time elapsed:

$$\phi_{ji}(t) = \alpha^{ji} \beta^i \exp(-\beta^i t) 1_{t>0}. \tag{14}$$

In this definition, for an object with L event types, we have three parts of parameters in total:

– μ: a vector of size L. It represents the base intensity of each dimension. It determines the arrival rate of events triggered by extraneous factors.
– α: a matrix of shape $L \times L$. It denotes the triggering coefficients within and across each dimension.
– β: a vector of size L. It represents the decay factor of the triggering effect. We have to set the triggering effect between dimensions as a time decay function, otherwise it will result in an infinite loop of event triggering.

With the definition of our object Hawkes process model, we are able to derive the log likelihood function of seeing the observed event time series. We maximize this function with respect to model parameters using the Broyden-Fletcher-Glodfarb-Shanno(BFGS) algorithm [11], a widely used quasi-Newtons method for unconstrained optimization problems without requiring the computation of a second-order Hessian matrix. The resulting mode is used to predict future events using Equation (10).

When both the ARIMA and the point process model are jointly used, we use the former to define long-term trends in total action budgets, and the latter to define action timing within those budgets.

3.4.3 Cascade Embedding Models

To generate reaction cascades, we propose a cascade embedding model that aims at learning the order in which agents join a copy-like reaction propagating through the medium. Many cascade models are inspired by epidemiology. An opinion may spread through a population like a virus, giving rise to infection-like models where the key is to find who gets infected next, given some history of recently infected agents.

Many cascade models capture influence relations between pairs of individuals. Hence, once an individual is infected, the probability of infections of their network neighbors is increased accordingly. Such models are, in the worst case, quadratic in population size, making them harder to learn from limited observed cascade data. Instead, we use dimensionality reduction.

Like the word embedding model (word2Vec) [12], which embeds words into vectors, our model embeds agents into a lower-dimensional space that captures their co-occurrence relations. More specifically, the set of recently infected agents are mapped into a point in space that helps predict the next agent to join the cascade, much the way a sentence prefix can help predict the next word.

In word2Vec, the embedding vector of each word is randomly initialized, and then converges to its optimized value by maximizing the cosine similarity with a list of its neighbors, called *context*. This can be achieved by building a neural network with one hidden layer and using gradient descent to optimize its parameters. The details of this process are shown in [12]. We embed the agents into vectors in a similar way expect for the definition of context. In word embedding, the context of a word are the words that occur ahead of it and behind it in the sentence.

In our case, the context of an action are those actions that occur ahead of it and behind it in cascade order. Figure 5 shows an example. There are four actions (a, b, c, d) in the figure, each represents a tweet. The original cascade might have a tree

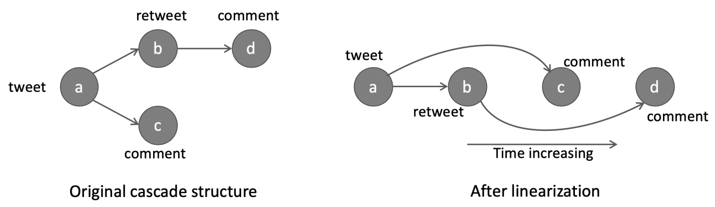


Fig. 5 Cascade example

structure. In this work, we linearize the original cascade by sorting the actions in ascending order by timestamp. Hence, in Figure 5, the timestamp of tweets satisfy: $t_a < t_b < t_c < t_d$. The context of action c are actions b and node d in Figure 5. There are two other things that we should consider:

- The root of a cascade usually has a disproportionate impact on all other actions in the cascade. This means that the root should also be considered as part of the context when calculating the embedding.
- We only need to consider the actions that occur ahead of the target one. This is because only those action could have impact the one simulated next.

Hence, we consider the previous p actions (nodes) and the root node when calculating the embedding of an agent ID (userID).

Given an embedding of the agents, we can assign an agent ID to each node of the cascade (expect to the root node) by calculating the cosine similarity between the target node and its previous p neighbors as well as the root node, and assign the agent that maximize the cosine similarity to the target node. The cascade embedding model can thus generate an agent ID for each of the node in a cascade. The total number of nodes in the cascade is decided by a higher-level model such as a popularity model for the corresponding object, computed using methods of Section 3.3.

3.5 Aggregation, Classes, and New Objects/Agents

Aggregation is a central component for agent-based simulations, as this allows for grouping of both agents and objects by similarity in behavior, which naturally reduces overhead at runtime and can even capture behavior of communities more accurately, if insufficient data is present on individual members.

3.5.1 Multi-scale Aggregation

The simulator leaves it open to define meta-agents that correspond to clusters of agents with similar behaviors. Ideally, such meta agents are used to model communities of agents when one does not have a sufficient number of data points to derive parameters of their individual agent model (e.g., per-agent ARIMA model parameters). If agents are grouped by predicted similarity, the group can collectively have enough data to be modeled accurately. The prediction of similarity is up to individual routines in the analysis library. For example, it may be based on organization, role, time-zone, or a combination thereof.

Taking time-based similarity calculation as an example, we divide the whole timeline into fine-grained time slots (e.g., days or hours). As mentioned above, if two agents often post events in the same slots, they tend to be similar with each other. Therefore, a bipartite graph can be constructed with agents on one side and time slots on the other. If agent a_i has activities in time slot t_k, there is an edge between the two nodes and the edge weight w_{ik} is the number of events a_i posts in t_k.

Given the bipartite graph, for any two agents a_i and a_j, we adopt the weighted Jaccard similarity to model their proximity:

$$J(a_i, a_j) = \frac{\sum_k \min(w_{ik}, w_{jk})}{\sum_k \max(w_{ik}, w_{jk})}. \tag{15}$$

Intuitively, the score checks all the time slots. For each slot, a_i and a_j will be measured as similar if they have comparable numbers of activities. Note that, the usage of this score is not limited to time-based similarity calculation. For example, on GitHub, two users can be viewed as similar if they frequently contribute to the same repository. From this perspective, we can construct a bipartite graph between users and repositories and adopt the same formula to compute ”repository-based” similarity scores.

3.5.2 Introduction of New Agents and Objects

Aggregation allows one to generate new objects and agents with realistic behaviors. Given a newly generated agent a_{new} with no historical data, one can determine an appropriate behavior model for the agent from a model of the class to which the agent belongs. Hence, the problem is reduced to one of computing the arrival rates of objects and agents of each class. The calculation can use envelope extrapolation techniques covered earlier in the paper, except that only new agents and actions are counted separately from others. For space considerations, we skip details of computing new object and agent generation trends based on the aforementioned techniques.

3.6 Model Selection

In preceding sections, we describe multiple alternative simulation models without commenting on the challenge of model selection. For example, models may exist at different levels of aggregations, such as individual agent models and models of communities (at different levels of aggregation) to which an agent belongs. Models can also differ in the nature of the extrapolation functions. Some functions might work well during vacation time, for example, whereas others might work better during typical work periods. In some cases, a principled approach is possible for selecting the best model to use in a given window.

As we have seen, estimating the parameters for each regression function (such as an envelope, Y, describing future activity trend) may rely on traditional techniques

like minimizing the residuals[1] in term of sum of squares. One may further partition the data, and divide the values of the regressors X into q non-intersecting intervals, which results in the following model Y = $\{f_1(X;\theta_1) + \varepsilon if \tau_0 < X_1,\dots,X_n \le \tau_1$
...
f$_q(X;\theta_q) + \varepsilon if \tau_{q-1} < X_1,\dots,X_n \le \tau_q$ where f_i's are different regression functions with known functional forms, $\tau_i < \tau_{i+1}$, $\tau_0 = -\infty$ and $\tau_q = \infty$, and ε is a zero-mean noise [13]. Key to segmented regression is to determine proper subdomain boundaries τ_i (also known as thresholds or breakpoints).

There are situations in which the piece-wise relationship between X and Y is dictated not only by the value of X but also by another, usually hidden factor, such as political polarity of a tweet [14]. While the exact state of the hidden factor is not available in many cases, a "good" candidate usually allows us to partition the dataset in such a way that results in a small total sum of residuals. This suggests that one can estimate the hidden factor by minimizing the total sum of residuals. Suppose we know the hidden factor belongs to one of q distinct states $[q] = \{1,\dots,q\}$. Define the state vector $s = (s_t)_{t=1}^n \in [q]^n$ where $s_t \in [q]$ is the state of the hidden factor at the data point (x_t, y_t) and n is the number of data points. The state vector partitions $[n] = \{1,\dots,n\}$ into q subsets according to $[n] = \bigcup_{i=1}^{q} n_i(s)$ where $n_i(s) = \{1 \le t \le n : s_t = i\}$. We fit the data with indices in $n_i(s)$ by a regression function f_i parameterized by $\theta_i \in \Theta_i$ where Θ_i is a predefined parameter space. In particular, let $\theta_i(s)$ be a/the minimizer of the residuals defined as

$$\theta_i(s) =_{\theta \in \Theta_i} \sum_{t \in n_i(s)} ||f_i(x_t;\theta_i) - y_t||_2, \tag{16}$$

the state vector that minimizes the total sum of squared residuals can be obtained by solving the following bi-level optimization problem

$$s^* =_{s \in [q]^n} \sum_{i=1}^{q} \Big(\sum_{t \in n_i(s)} ||f_i(x_t;\theta_i(s)) - y_t||_2 \Big). \tag{17}$$

Solving for s^* in Equation 17 is challenging while the size of the solution space is q^n, which is exponential in the number of data points. Heuristic search techniques like simulated annealing and genetic algorithms have been used in similar situations. However, both techniques require an extra step to fine tune the parameters and can be slow to converge. Otherwise, one can solve for s^* using an algorithm similar to the Expectation-Maximization (EM) algorithm. Specifically, at iteration $k \ge 1$, we first (i) estimate the residual-minimizing parameter $\theta_i^{(k)} = \theta_i(s^{(k-1)})$ according to Equation 16 where $s^{(k-1)}$ is the state vector computed from the previous iteration, then (ii) reevaluate the value of the state vector at each data point (x_t, y_t) according to $s_t^{(k)} =_{i \in [q]} |f_i(x_t;\theta_i^{(k)}) - y_t|$. The algorithm stops when it converges to a single solution or (optionally) reaches a predefined number of iterations. Similar to EM, the final estimate of s^* is also sensitive to the random initialization $s^{(0)}$ of the state vector.

The expectation maximization algorithm

[1] defined as the magnitude of the difference between model estimate and observation at a point.

The model selection approach based on hidden factor-driven regression described so far allows us to fit and analyze the data from some unique angles. Being a generalization of segmented regression, it supports more aggressive data partitioning by allowing different states to be assigned to different data points that otherwise belong to the same interval. This approach bears some similarity to data clustering. Unlike centroid-based clustering however, cluster membership is determined not by whether a data point is sufficiently close to the cluster center but by whether it follows the behavior of those in the cluster, where as such a behavior is described by a parameterized regression function. Third, this approach provides some intuition to the explanation of how the data was generated and goes beyond the goal of minimizing the total sum of residuals, which can be achieved using arbitrarily large NNs.

4 EVALUATION

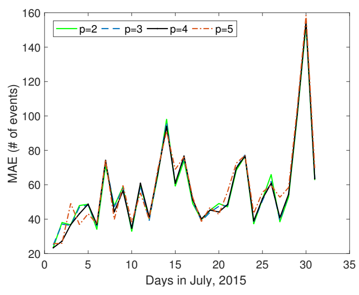
In this section, we present some of the basic results of running selected components of the analysis library and simulation on data obtained (primarily) from GitHub. A comprehensive evaluation of all methods described in this manuscript is beyond the scope of this paper. It is in fact, the goal of the SocialSim program. Unless noted otherwise, the dataset used in this section is provided by Leidos. It covers records from Jan 2015 to Aug 2017. All the records have been anonymized to protect privacy of Github users. Each record specifies information of actor (user agent), payload, timestamp, action type, as well as repository id (object id).

4.1 Predicting Activity Rate and Type

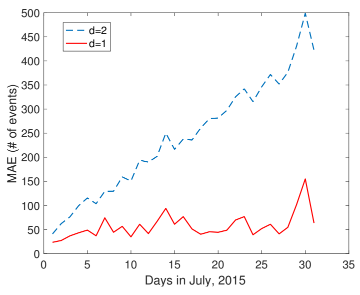
In this experiment, we are going to predict the number of different event types for a test period (month) for each user on Github. On Github, there are about 10 different event types such as PushEvent, ForkEvent, DeleteEvent and PullRequest. Due to space limitations, we use only the three more common event types, PushEvent, ForkEvent and PullRequest, as representatives. We compare ARIMA and a stationary method (where the future prediction is set identical to the current rates). First, we investigate the impact of model structure (i.e., parameters, p, d, and q) on accuracy of the ARIMA model. We use the data from January to June in 2015 to train the ARIMA model, then test its performance on the data in July 2015. Fig. 6 shows the mean absolute error (MAE) for different values of parameters p, d, q in predicting PushEvent rates. From these (and similar experiments), we see that $p = 4$, $d = 1$, and $q = 0$ give best accuracy.

The next step is to predict the number of event types for each user using ARIMA model. It took about 2 hours to train the analysis library to generate models of 2000 agents based on six months of training data. Fig. 7 shows the mean absolute error (MAE) in predicting the rate of different event types of the top 2000 users using ARIMA and the stationary method (where future is equal to the present). As expected, we observe that the ARIMA method outperforms the stationary method.

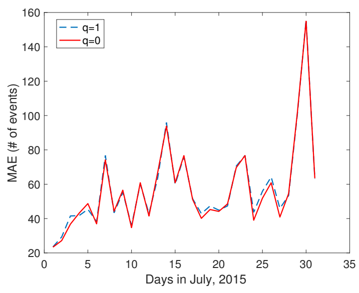
After predicting the number of different event types for each user, we can compute the probability distribution of each event type. We take one user as an example in



(a) Different p for ARIMA for PushEvent



(b) Different d for ARIMA for PushEvent



(c) Different q for ARIMA for PushEvent

Fig. 6 Parameters tuning for ARIMA model

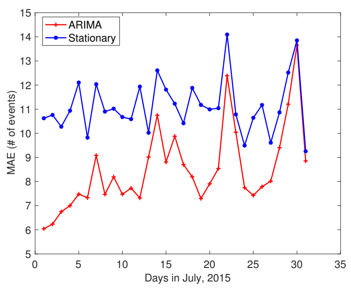
Table 1. The second row is the number of events predicted by the ARIMA model for this user while the last row is the corresponding probability distribution. This probability distribution would then be part of the configuration file for simulating the desired epoch of the user.

Table 1 Probability distribution of different events for a certain user

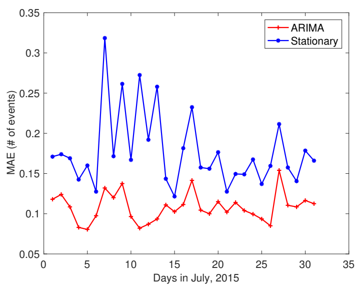
Issues	PullReviewComment	Push	PullRequest	IssueComment	Create	Watch	Fork	Delete	CommitComment
95	197	3796	45	893	25	0	0	4	0
.0188	.0389	.7509	.0089	.1766	.0049	0	0	.0008	0

4.2 Predicting Affinity

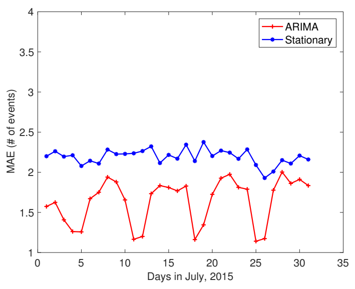
In this part, we evaluate our ARIMA model for predicting affinities of the agents, and then compare the performance with the stationary agent model. We use all the Github events from January to June in 2015 to train our ARIMA model and use the data in July as the test set. Per affinity definition in Section 3, for Github, affinities refers to the set of repositories a user contributes to. In the evaluation, we build the agent model for 10^5 users who are active in June 2015, and predict their interaction rates with their repositories in July 2015 based on our ARIMA model. We then compare to the



(a) Comparison of MAE for Push event



(b) Comparison of MAE for Fork event



(c) Comparison of MAE for PullRequest event

Fig. 7 Comparison of MAE for ARIMA and stationary

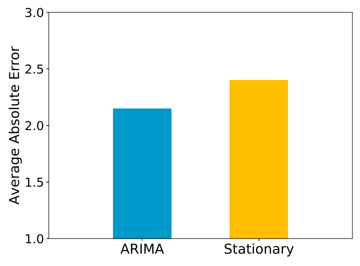
stationary model. We use the difference between the ground truth and the prediction values as the absolute errors, and define the accumulated error as the accumulation of the absolute errors over users. We run our ARIMA model using two cores of an Intel i7 CPU. It took less than one hour to train the analysis library to generate models of 10^5 agents based on the 6 months of training data.

Figure 8(a) shows the average absolute error of ARIMA and the Stationary model over the 10^5 users. We observe that our ARIMA model generates a smaller absolute error overall. Breaking it down by user, Figure 8(b) shows that ARIMA gives better error for 78.6% users, while the Stationary model performs better for the remaining 21.4%). Figure 8(c) shows the cumulative absolute errors, verifying the advantages of ARIMA-based prediction.

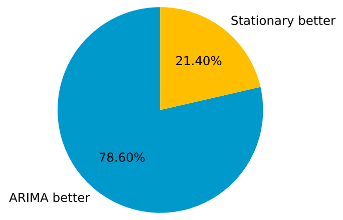
4.3 Understanding Short-term Dependencies

In this subsection, we briefly present the evaluation results for simulated events provided by the short-term point process model we introduced before. First, we describe the performance of single repository simulation results. Second, we enlarge the scope of comparison and evaluate aggregate performance on a set of popular repositories.

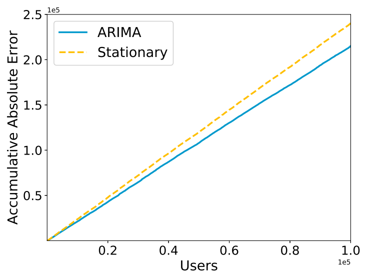
In the first experiment, we randomly choose one repository out of the January 2015 event set and estimate its point process parameters. Then we simulate this model in February 2015 and compare it with ground truth. The estimated triggering kernels,



(a) Average Absolute Error

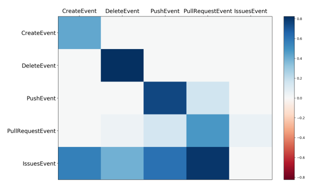


(b) Ratio of Smaller Absolute Errors

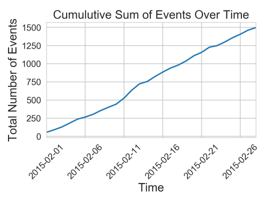


(c) Accumulative Avsolute Error

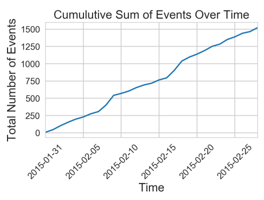
Fig. 8 Comparison between ARIMA and stationary



(a) Triggering Kernels



(b) Groundtruth Events



(c) Simulation Events

Fig. 9 Estimation and simulation results on repository ”JwcbE-owDa9ymJm3D4xhsw”.

and the cumulative distributions of number of events over time are shown in Figure 9. From the triggering kernel map, we can see that the different event types on this repository are relatively independent because the triggering coefficients are higher in diagonal elements except for the IssuesEvent. As presented in this figure, our point process model maintains the daily activity level (i.e., slope of curve) and the burstiness (i.e., smoothness of curve) well in the simulation.

In addition, we compare other statistical metrics in the Table 2. In this table, the diffusion delay is defined as the time difference between the time of first event on this repository and the time of each following event. It represents the average offset of all the events in this repository. Our simulation is within 10% error of the groundtruth.

The simulation also performs well on the average inter-event time and total number of events because it is designed to model the triggering effects on inter-event times.

Table 2 Comparison on single repository.

Metric	Groundtruth	Simulation
Average Diffusion Delay	325.599 h	348.077 h
Average Inter-event Time	1575.643 s	1579.644 s
Total Number of Events	1497	1532

Next, to test the performance of our model on more data, we choose the 1000 most popular repositories in the first half year of 2015 to perform simulation. In this experiment, we only show the WatchEvent and ForkEvent, which reflect the popularity of these repositories. The training period is January 2015 to June 2015, and the simulation is conducted in July 2015. We first independently estimate the parameters for each repository, then perform separate simulation on them, and finally combine the simulated events together to compare with the groundtruth events on these repositories. The total training time of the analysis library for the 1000 repositories is about 6 hours on a server with 48 Intel Xeon CPUs@2.50GHz and 256 GB memory, where the total number of events on these repos is beyond 1.4 million. However, the simulation of these repositories will only last for $250 \sim 300$ seconds to generate one month of results, running on a single core. The comparison results of accuracy are given in Table 3.

Table 3 Comparison on top 1000 repositories.

Metric	Groundtruth	Simulation
Average Diffusion Delay	372.401191 h	373.485502 h
Average Inter-event Time	83815.010660 s	80263.662305 s
Total Number of Events	242247	275133

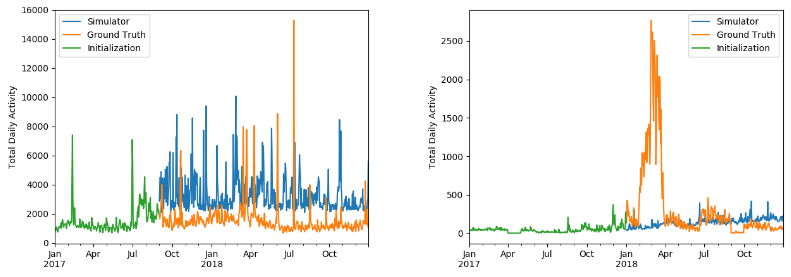
In Table 3, since we include more repositories in the comparison, the diffusion delay distribution is more balanced, resulting the average delay very close to the middle point of a month. We still obtain good accuracy on average inter-event time (relative error $< 5\%$) and total number of events (relative error $\simeq 12\%$), which proves the stability of the point process model.

4.4 Trend Aggregation

Here, we evaluate both the efficiency and accuracy of clustering by temporal behavior patterns. In the simulation, the offline analysis library is fed a data set with a date range of 2016-03-01 to 2016-04-01, comprising 27803145 individual events. We compute regional predictions of temporal activity patterns and use them to eliminate agents that are (predicted to be) asleep from simulation at every time step. The results are detailed in Table 4.

Table 4 Evaluation results for temporal preference clustering

	With Activity Level Prediction	Full Simulation
Runtime (sec)	42.5	203.3
RMSE	305.4	273.1



(a) Dryrun (b) Challenge

Fig. 10 The total activity simulated on the dryrun data and the challenge data.

The table shows that eliminating users predicted to be inactive based on time zone saves 79.1% of simulation time, while the percent difference in simulation error (RMSE) in predicting hourly PushEvents for the simulated month is only 11.2%. Better clustering remains an open challenge and an active area of development of analysis libraries for our simulator. Regardless, the decrease in overhead is substantial and the speed-up scales well with the number of agents in the training data.

4.5 Total Activity Prediction for the Decaying Poisson Point Process Model with Agent Creation

As part of the simulation challenge, our models were tested on two different data sets. The first, referred to as the dryrun data, had 8 months of initialization data with a 16 month simulation period. The second, referred to as the challenge data, had 12 months of initialization data with a simulation period of 12 months. Our models were run on the Twitter and Telegram social media platforms. The total activity trends are visualized in Figure 10. The model does not capture the large spike in activity that occurs in the challenge dataset at the beginning of the simulation period. However, in both cases the model performs well at capturing the baseline activity.

The numbers of agents simulated for the two datasets are shown in Tables 5 and 6. We see that the vast majority of agents in either data set generate their first event after the simulation begins. Incorporating agent destruction and creation into agent-based models is therefore crucial.

Metric	Simulator	Ground Truth
Total Agents	400,381	331,355
New Agents	377,308	306,527

Table 5 Simulation of new agents on the dryrun data.

Metric	Simulator	Ground Truth
Total Agents	17,585	12,005
New Agents	16,806	11,131

Table 6 Simulation of new agents on the challenge data.

4.6 Persistent Group Measurements with Cooccurrence

The effect of the coocurrence model cannot be seen in the overall activity trends since it does not affect either the total number of posts or the total number of users. Instead, it affects the relationship between users. We then simulated the Poisson point process model both with and without the application of cooccurence. The results on the two datasets are shown in Tables 7 and 8. These results are collected by constructing a weighted network between agents who act in the same time period. The simulated results are measured against the ground truth data. The reported results show the absolute percentage errors and the Jensen-Shannon (JS) divergences, which is a quantity for measuring similarity between probability distributions. It is 0 if the two probability distributions are identical and has an upper bound of 1 [15]. While the introduction of the cooccurrence module does not always improve every metric, it does generally lead to a greater agreement with the social structure observed in the ground truth.

Metric	Without Cooccurrence	With Cooccurrence
No. of groups (percentage error)	266.67	233.33
Out-group to in-group interaction ratio (% error)	0	0
Distribution of content discussion (JS divergence)	1.0	0.75828
Seed posts to total events (JS divergence)	0.42956	0.39582

Table 7 Group metrics on the dryrun data.

Metric	Without Cooccurrence	With Cooccurrence
No. of groups (percentage error)	114.29	157.14
Out-group to in-group interaction ratio (% error)	220.67	100
Distribution of content discussion (JS divergence)	1.0	1.0
Seed posts to total events (JS divergence)	0.7238	0.69048

Table 8 Group metrics on the challenge data.

5 RELATED WORK

A key direction in simulation literature is to empower diverse application domains with agent-based simulators for in-depth understanding of real-world complexities, and forecasting quantities of interest. Railsback et al. proposed an agent-based simulation platform for development recommendations [16]. Drogoul *et al.* design a multi-agent-based simulator for distributed artificial intelligence research [17]. Bunn *et al.* use agent-based simulation for electricity trading arrangements of England and Wales [18]. Raberto *et al.* apply agent-based simulation for understanding the financial markets [19].

Recently, simulations for online media have attracted a growing interest from the community. Abundant studies have been made for understanding the influence of social networks on human daily lives and decision making [20,21], validating online viral marketing [22,23], and modelling rumor propagation [24,25]. One key link missing from the picture, however, is a flexible and self-configuring simulation framework that integrates various online media with a modular interface designed for extensibility of data analytics and prediction functions.

With the proliferation of interconnected users, many studies were made on modelling user behaviour. Wang *et al.* jointly model user and object interactions with latent credible status and user dependencies [26,27]. Yao *et al.* incorporate time information and compute a fundamental error bound [28,29]. Shao *et al.* formulate a nonlinear integer programming problem to select optimal sources to solicit in order to minimize the expected fusion error [30]. Zhao *et al.* focus on modeling the evolution of popularity on online media by mining big information cascades [10]. Hoang *et al.* study the online propagation behaviors of micro-blogging content [31]. From the prospective of online media simulation, it is not easy to design plug-in modules that accommodate such diverse considerations. To the best of our knowledge, this paper develops a novel modular framework for integrating models at different scales to address difference micro/macro concerns while allowing self-configuration and easy retargeting.

The paper describes a novel highly retargetable simulator of online media. With systemically designed self-configuration modules, we make automatic learning and self-configuration of online media simulators accessible.

6 CONCLUSIONS

In this paper, we described a new simulator of social media streams. The simulator is novel in that it does not require manual configuration. Rather, the simulated medium is automatically instantiated from analysis of past data traces from the target medium. Prediction algorithms allow extrapolating from the past trace to a predicted future. Preliminary evaluation results show that the predictions are accurate and scalable. The simulator is extensible. As better prediction and clustering algorithms are developed over time, they can be integrated as modules in the analysis library and as agent behavior models. Similarly, as new social media are considered, they can be converted into an agent/object/action representation by new thin pre-processing modules, allowing existing analysis algorithms to extract parameters of agents and objects for the new medium. Thus, analysis and simulation code can be reused across different online media. Future work of the authors will address performance optimization of the simulator, extensions of analysis libraries and agent behavior models, and support for additional input media types.

ACKNOWLEDGMENTS

This work was sponsored in part by DARPA under contract W911NF-17-C-0099.

References

1. C. Song, Z. Qu, N. Blumm, and A.-L. Barabási, “Limits of predictability in human mobility,” *Science*, vol. 327, no. 5968, pp. 1018–1021, 2010. [Online]. Available: https://science.sciencemag.org/content/327/5968/1018
2. J. D. Hamilton, *Time series analysis*. Princeton university press Princeton, 1994, vol. 2.
3. T. Anderson, “Estimation for autoregressive moving average models in the time and frequency domains,” *The Annals of Statistics*, pp. 842–865, 1977.
4. P. Shaman, “An approximate inverse for the covariance matrix of moving average and autoregressive processes,” *The Annals of Statistics*, pp. 532–538, 1975.
5. G. M. Ljung and G. E. Box, “The likelihood function of stationary autoregressive-moving average models,” *Biometrika*, vol. 66, no. 2, pp. 265–270, 1979.
6. A. G. Hawkes, “Spectra of some self-exciting and mutually exciting point processes,” *Biometrika*, vol. 58, no. 1, pp. 83–90, 1971.
7. M.-A. Rizoiu, Y. Lee, S. Mishra, and L. Xie, “A tutorial on hawkes processes for events in social media,” *arXiv preprint arXiv:1708.06401*, 2017.
8. P. Embrechts, T. Liniger, and L. Lin, “Multivariate hawkes processes: an application to financial data,” *Journal of Applied Probability*, vol. 48, no. A, pp. 367–378, 2011.
9. Y. Ogata, “Statistical models for earthquake occurrences and residual analysis for point processes,” *Journal of the American Statistical association*, vol. 83, no. 401, pp. 9–27, 1988.
10. Q. Zhao, M. A. Erdogdu, H. Y. He, A. Rajaraman, and J. Leskovec, “Seismic: A self-exciting point process model for predicting tweet popularity,” in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1513–1522.
11. D. C. Liu and J. Nocedal, “On the limited memory bfgs method for large scale optimization,” *Mathematical programming*, vol. 45, no. 1-3, pp. 503–528, 1989.
12. T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
13. J. Liu, S. Wu, and J. V. Zidek, “On segmented multivariate regression,” *Statistica Sinica*, vol. 7, no. 2, pp. 497–525, 1997. [Online]. Available: http://www.jstor.org/stable/24306090
14. M. T. Bastos, R. L. G. Raimundo, and R. Travitzki, “Gatekeeping twitter: message diffusion in political hashtags,” *Media, Culture & Society*, vol. 35, no. 2, pp. 260–270, 2013. [Online]. Available: https://doi.org/10.1177/0163443712467594
15. J. Lin, “Divergence measures based on the shannon entropy,” *IEEE Transactions on Information Theory*, vol. 37, no. 1, pp. 145–151, Jan 1991.
16. S. F. Railsback, S. L. Lytinen, and S. K. Jackson, “Agent-based simulation platforms: Review and development recommendations,” *Simulation*, vol. 82, no. 9, pp. 609–623, 2006.
17. A. Drogoul, D. Vanbergue, and T. Meurisse, “Multi-agent based simulation: Where are the agents?” in *International Workshop on Multi-Agent Systems and Agent-Based Simulation*. Springer, 2002, pp. 1–15.
18. D. W. Bunn and F. S. Oliveira, “Agent-based simulation-an application to the new electricity trading arrangements of england and wales,” *IEEE transactions on Evolutionary Computation*, vol. 5, no. 5, pp. 493–503, 2001.

19. M. Raberto, S. Cincotti, S. M. Focardi, and M. Marchesi, "Agent-based simulation of a financial market," *Physica A: Statistical Mechanics and its Applications*, vol. 299, no. 1-2, pp. 319–327, 2001.
20. J. Zhang, L. Tong, P. Lamberson, R. Durazo-Arvizu, A. Luke, and D. Shoham, "Leveraging social influence to address overweight and obesity using agent-based models: the role of adolescent social networks," *Social science & medicine*, vol. 125, pp. 203–213, 2015.
21. M. J. Lanham, G. P. Morgan, and K. M. Carley, "Social network modeling and agent-based simulation in support of crisis de-escalation," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 1, pp. 103–110, 2014.
22. E. Serrano and C. A. Iglesias, "Validating viral marketing strategies in twitter via agent-based social simulation," *Expert Systems with Applications*, vol. 50, pp. 140–150, 2016.
23. A. Sela, D. Goldenberg, E. Shmueli, and I. Ben-Gal, "Scheduled seeding for latent viral marketing," in *Advances in Social Networks Analysis and Mining (ASONAM), 2016 IEEE/ACM International Conference on*. IEEE, 2016, pp. 642–643.
24. E. Serrano, C. Á. Iglesias, and M. Garijo, "A novel agent-based rumor spreading model in twitter," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 811–814.
25. C. Kaligotla, E. Yücesan, and S. E. Chick, "An agent based model of spread of competing rumors through online interactions on social media," in *Winter Simulation Conference (WSC), 2015*. IEEE, 2015, pp. 3985–3996.
26. D. Wang, M. T. A. Amin, S. Li, T. F. Abdelzaher, L. M. Kaplan, S. Gu, C. Pan, H. Liu, C. C. Aggarwal, R. K. Ganti, X. Wang, P. Mohapatra, B. K. Szymanski, and H. K. Le, "Using humans as sensors: An estimation-theoretic perspective," *IPSN-14 Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, pp. 35–46, 2014.
27. S. Wang, L. Su, S. Li, S. Hu, M. T. A. Amin, H. Wang, S. Yao, L. M. Kaplan, and T. F. Abdelzaher, "Scalable social sensing of interdependent phenomena," in *IPSN*, 2015.
28. S. Yao, S. Hu, S. Li, Y. Zhao, L. Su, L. M. Kaplan, A. Yener, and T. F. Abdelzaher, "On source dependency models for reliable social sensing: Algorithms and fundamental error bounds," *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, pp. 467–476, 2016.
29. S. Yao, M. T. Amin, L. Su, S. Hu, S. Li, S. Wang, Y. Zhao, T. Abdelzaher, L. Kaplan, C. Aggarwal *et al.*, "Recursive ground truth estimator for social data streams," in *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*. IEEE Press, 2016, p. 14.
30. H. Shao, S. Wang, S. Li, S. Yao, Y. Zhao, T. Amin, T. Abdelzaher, and L. Kaplan, "Optimizing source selection in social sensing in the presence of influence graphs," in *Distributed Computing Systems (ICDCS), 2017 IEEE 37th International Conference on*. IEEE, 2017, pp. 1157–1167.
31. T.-A. Hoang and E.-P. Lim, "Microblogging content propagation modeling using topic-specific behavioral factors," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 9, pp. 2407–2422, 2016.