D2U: Data Driven User Emulation for the Enhancement of Cyber Testing, Training, and Data Set Generation

Sean Oesch
Oak Ridge National Laboratory
Oak Ridge, TN, USA
oeschts@ornl.gov

Robert Bridges
Oak Ridge National Laboratory
Oak Ridge, TN, USA
bridgesra@ornl.gov

Miki Verma
Stanford University
Palo Alto, CA, USA
meverma@stanford.edu

Brian Weber
Oak Ridge National Laboratory
Oak Ridge, TN, USA
weberb@ornl.gov

Oumar Diallo
Oak Ridge National Laboratory
Oak Ridge, TN, USA
omdiallo@gmail.com

ABSTRACT

Whether testing intrusion detection systems, conducting training exercises, or creating data sets to be used by the broader cybersecurity community, realistic user behavior is a critical component of a cyber range. Existing methods either rely on network level data or replay recorded user actions to approximate real users in a network. Our work produces generative models trained on actual user data (sequences of application usage) collected from endpoints. Once trained to the user's behavioral data, these models can generate novel sequences of actions from the same distribution as the training data. These sequences of actions are then fed to our custom software via configuration files, which replicate those behaviors on end devices. Notably, our models are platform agnostic and could generate behavior data for any emulation software package. In this paper we present our model generation process, software architecture, and an investigation of the fidelity of our models. Specifically, we consider two different representations of the behavioral sequences, on which three standard generative models for sequential data—Markov Chain, Hidden Markov Model, and Random Surfer—are employed. Additionally, we examine adding a latent variable to faithfully capture time-of-day trends. Best results are observed when sampling a unique next behavior (regardless of the specific sequential model used) and the duration to take the behavior, paired with the temporal latent variable. Our software is currently deployed in a cyber range to help evaluate the efficacy of defensive cyber technologies, and we suggest additional ways that the cyber community as a whole can benefit from more realistic user behavior emulation.

KEYWORDS

data sets, experimental infrastructure, user emulation, data driven

CSET'21, August 09, 2021, Virtual

ACM ISBN 978-1-4503-9065-1/21/08…$15.00
https://doi.org/10.1145/3474718.3475718

ACM Reference Format:
Sean Oesch, Robert Bridges, Miki Verma, Brian Weber, and Oumar Diallo. 2021. D2U: Data Driven User Emulation for the Enhancement of Cyber Testing, Training, and Data Set Generation. In *Cyber Security Experimentation and Test Workshop (CSET '21), August 9, 2021, Virtual, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3474718.3475718

1 INTRODUCTION

In 2015, the National Science Foundation released a report entitled Cybersecurity Experimentation of the Future: Catalyzing a New Generation of Experimental Cybersecurity Research (CEF) [1], which laid out a roadmap of the infrastructure needed to support cyber research. The CEF highlights the need for better techniques to "instrument and observe the behavior of real humans and to automatically extract valid behavior models, and then inject into test environments". While the CEF mentions several existing emulation technologies, including LARIAT [17] and KOALA [3] from MIT Lincoln Laboratory, and the DETER Agents Simulating Humans (DASH) [21] project from USC-ISI, it also emphasizes the need for further research in modeling human behaviors.

In addition, the problem of generating realistic data for use in cyber testing and training is well established in the literature [1, 18]. Many existing datasets cannot be properly validated and are outdated [1, 14, 22]. A critical step towards more realistic cyber data for training and testing is high fidelity user emulation [5].

In this paper we present D2U, a novel user behavior emulation technology that collects, models, and performs realistic user behaviors in cyber testbeds, aligning with the goals of the CEF and providing much needed data for cyber experimentation. D2U does more than replay previously-observed behavior or string together patterns of actions based on heuristics, and it does not rely on agent-based modeling approaches. Instead, D2U provides the ability to generate unlimited, stochastic but realistic sequences of behavior based on data collected on individual users; then D2U orchestrates a plethora of virtual machines to enact these behaviors, thereby emulating a network of users.

In a one-time training phase D2U collects application (app) usage data on a real computer user and creates probabilistic generative models of that user's unique patterns of behaviors. These models enable the production of new sequences that appear similar to the

[1] https://cef.cyberexperimentation.org/

real user's behavior. D2U calls sequence-visualization software, allowing quality analysis to gauge realism of the generated behaviors. Next, user behavior, as generated by our models, is fed into custom software that is able to actuate those behaviors on real or virtual devices—a "digital twin" of the user on that device.

By placing these twins on many devices on the network, D2U helps address a variety of problematic use cases, including: (1) generating realistic host and network data for testing cyber tools (e.g., needed to evaluate emerging cybersecurity technologies, such as User Behavior Analytics (UEBA), Anomaly Detection (AD), and Intrusion Detection System (IDS) technologies), (2) creating cyber deception technologies (e.g., camouflaging real users/traffic with realistic emulated versions), (3) creating training data for machine-learning-based IT and cyber tools, (4) enhancing the realism of cyber exercises (e.g., red-team events for testing and practicing network defense), and (5) generating novel datasets to support both research and industry. D2U is currently running in a cyber testbed with over 300 active emulated users at [redacted] to support large-scale experiments involving defensive cyber technologies.

This preliminary work provides a qualitative evaluation of our models based on 42 days of application sequence data collected on one of the paper authors (Sec. 2), as well as describing the software architecture we use to deploy D2U in our cyber range (Sec. 3). Specifically, we consider two dichotomies in our approach: Firstly the choice of representing sequential data with repeated symbols (STS) or unique symbols paired with their duration (DSSDS—Sec. 2.3.1). Secondly, the structural modeling choice of using the original sequences (Flat) vs. adding a latent layer to model time-of-day trends (Hierarchical—Sec. 2.3.2). As both require a generative model for sequential data, we employ three standard approaches: Markov Chain, Hidden Markov Model, and Random Surfer (Sec. 2.3.3). We find that, regardless of the sequential model used, DSSDS best replicates realistic "spell lengths", the duration one resides in the current behavior, and the Hierarchical model best captures temporal trends.

The application sequence data used to train our model, as well as model output and sample configuration files, are available at https://github.com/oeschsec/D2U-artifacts. While we are not planning to make the modeling code publicly available to keep our options open for future usage, replication of models/results could be attained by the modeling details enclosed.

2 MODELING APP USAGE SEQUENCES

To create a data-driven model of a user's behavior, data recording the user's activity is collected from the user's host for an ample period of time (many days to weeks). The data is used to train a generative model, that is, an algorithm that, once trained, can be used to produce novel sequences of activities that mimic the input data's qualities. This section details the data collection and preprocessing, then introduces the generative models tested alongside sequential qualities of the data from previous research that drive evaluation and model selection.

2.1 Data Collection & Preprocessing

A collection script was written which records a timestamped observation each 0.5s of the user's "active application", i.e., the frontmost application in the operating system's (OS's) user interface (UI). This script was deployed on the work computer of two of the paper authors, one of whom provided the data used in our analysis. For this participant, we collected 42 days of sequential data over the course of $\sim$ 2 months. Figure 1 provides a summary of the applications used by the participant during a ten day span within the collection period.

Our collection script, written in Python, leverages the AppKit[2] package for Mac OS data collection, which is used in this paper, although we created analogous collection scripts for Linux and Windows OSes from a variety of Python modules. The program was designed to collect data continuously for eight active hours (not including sleep periods where logging was automatically paused). Notably, the program recorded `loginwindow` as the app name before the computer would sleep/pause logging, which allows our models to generate realistic pauses in the user's day. When a collection period was complete, participants were instructed to restart the data collection program.

Each run of the collection program provides a time series of active app observations: $(s'(t_0), \ldots, s'(t_L))$, where $t_{k+1} = t_k + 0.5$s for most k, and $L = 8\text{hrs} \times 60^2\text{s/hr} \times 2 = 57.6\text{K}$ observations. Note, we use s' to denote the raw sequence reserving the simple notation s for it's analogous sequence of apps after preprocessing. Once we have obtained a sufficient number of observation sequences from a user, we normalize our data to have the following properties:

- A single sequence per day, filtering out days in which there was insufficient active data (e.g. user was logged out). When denoting the sequence for a particular day is needed, we let s^i denote the sequence for the i-th day.
- A uniform start and end time (this depends on the user), which we define as simply the minimum start time ($\min_{s'} t_0$) and maximum end time ($\max_{s'} t_L$) observed.
- Uniform and uniformly spaced timestamps in each day's sequence (i.e., all s are sequences with the same time stamps), coarsened to five second intervals. To do this we simply set $s(t)$ to be the closest previous observation—set $s(t) := s'(t_k)$ where $t \in (t_k, t_{k+1})$—provided we have observations s' near time t. In the alternative case, where there is a large gap in a day's data collection (e.g., from the collection script ending but not being restarted quickly) we copy subsequences of s' occurring before and after the gap to fill in the unknown portion, then define $s(t)$ as just previously mentioned.
- We replace seldomly used apps—defined as apps that occur in only a single collection period or that make up less than 1% of the data—with a distinguished symbol, `RARE`, effectively binning all infrequent observations.

Now armed with uniform sequences of user behavior for many days, we consider the sequential properties and models that may faithfully reproduce those properties.

Before proceeding we note that Figures 1, 2, 3, and in Table 3 leverage the R package of Gabadinho et al. [6].

2.2 Sequential Concepts and Modeling

We seek models that are "high-fidelity", where we define the fidelity of a user model in terms of the similarity of the activity

[2]https://developer.apple.com/documentation/appkit

sequences the model generates to that of the true user sequences; consequently, gauging how "good" a model is depends solely on defining similarity of two sequences. Yet, finding an adequate set of similarity metrics for capturing the characteristics in real data is difficult to do. Sequential data appears in a wide variety of domains. Hence, a wide variety of similarity measures for sequences exist; e.g., string metrics such as Hamming for binary data [9], edit distances (e.g., Levenshtein) often used for text applications [8], or Kendall-Tau distance often used for ranking comparisons [10], to name a few. Studer & Ritschard [19] provide a thorough survey.

Considering Figure 1, which visualizes ten days (sequences) of a user's application usage, we observe wide variance in behaviors that characterize real user data. Different app distributions appear at different times. Occasionally long spells of a particular app occur, while short spells are very common. Indeed, toggling apps is observed, where a "core" app dominates a time period but is interrupted frequently with short visit into one or two other apps. Finally, there is a general time window of work hours in the day for this user. Our goal is to create/discover a model that, when trained on data such as that shown in Figure 1, will produce data that has similar qualities. While the general workflow for creating such a model is to identify metrics that measure the desired qualities, and then leverage these metrics to design and evaluate models, it is unknown what combination of sequential measures capture the qualities of real user's behavior—and this is a non-trivial problem!

Although working in social sciences, Studer & Ritschard [19] encountered this very problem and have laid a foundation of sequential concepts that frames our approach. We consider similarity of sequential activity data with regard to four sequence characteristics as itemized by Studer & Ritschard:

- *Sequencing* - the ordering of distinct applications (i.e., order in which a user jumps from one application to another);
- *Duration* - the number of consecutive observations of the same application (equivalently, uninterrupted time spent in each application);
- *Timing* - the time(s) of day at which each application is used;
- *Distribution* - the total time spent in each distinct application.

All of these characteristics are ideally similar between the real user data and the desired synthetic user data. Further, some element of stochasticity or randomness appears in the user data that we seek. Hence, we test probabilistic generative models in a framework we iteratively designed with the above characteristics in mind to approach an accurate method for synthesizing realistic user data.

Continuing, we introduce necessary terminology and notation, following previous works ([6, 19]) but adapted for our needs.

- The terms *state* and *app* and *symbol* are used interchangeably to refer to the user's active application.
- A *Symbol Time Sequence (STS)* is a sequence $s(t)$ where each element of the sequence $s(t_i)$ denotes the symbol at time t_i.
- A *spell* is a consecutive subsequence of the same symbol, or equivalently, an uninterrupted period spent in the same application; it follows that spell length, the number of consecutive same symbols, is *duration* as defined above.

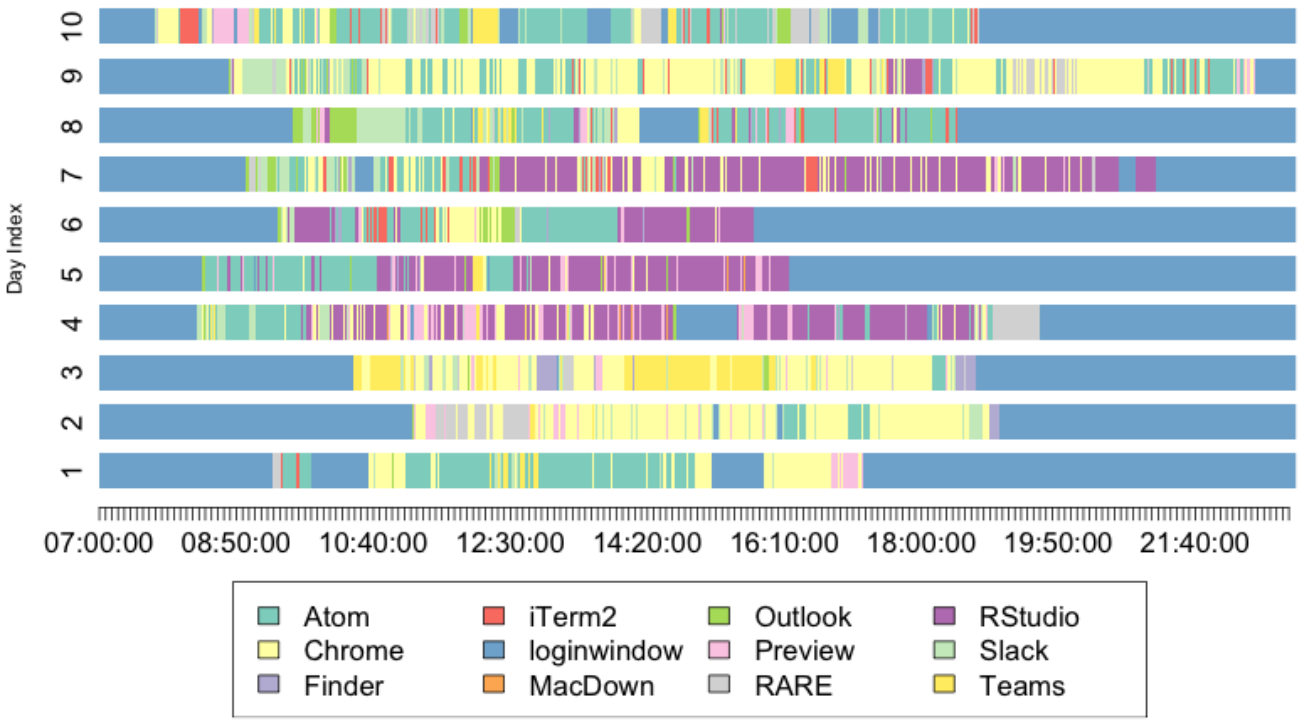


Figure 1: Ground Truth Data Example: Ten days (two work weeks) of data (after preprocessing) for a user.

- The *Distinct Successive States (DSS)* is the sequence of distinct symbols observed, where all spells are treated as length 1. We denote a DSS using $u = (u(1), \ldots, u(m))$.
- A *Distinct Successive State Duration Sequence (DSSDS)* is a sequence of (symbol, spell duration) tuples for each spell in the DSS.

Example: For the toy STS, $s = (a, b, b, b, a, c, c)$ there are three states or symbols, namely, a, b and c; symbol a has two spells of length 1, b has a spell of length 3, and c has a spell of length 2; the DSS is $u = (a, b, a, c)$; the corresponding DSSDS is $((a, 1), (b, 3), (a, 1), (c, 2))$.

The DSSDS is a redundant but alternative specification of the original STS. As we shall see, choosing whether to consider a user activity sequences as an STS or a DSSDS has an impact on model choice and accuracy that we explore via initial results.

2.3 Modeling

We develop a flexible framework for modeling user activity sequences where a particular model can be specified by three decisions:

(1) *Sequence Representation*: STS or DSSDS (Sec. 2.3.1);
(2) *Temporal Structure*: Flat or Hierarchical (Sec. 2.3.2).
(3) *Sequential Model Type & Hyperparameters*: Markov Chain (MC), Hidden Markov Model (HMM), or Random Surfer (RS); hyperparameters vary per model type (Sec. 2.3.3);

The four possible graphical model structures are depicted in Table 1 with rows/columns corresponding to the pair of structural dichotomies ((1) and (2)). Sequential model types (3) are shown in Table 2 along with the model-specific hyperparameters we tested. Each of the four structures (choices for (1) and (2)) are used with one of the three model types, yielding a total of $4 \times 3 = 12$ model combinations ($4 \times 6 = 24$ when considering hyperparameters we tested). To motivate our framework, we first note that all models are Markovian, that is, they are built to preserve some sequential characteristics. We test three increasingly complex models culminating in the RS model, which is an intuitive choice for how one navigates between apps. The STS vs. DSSDS flexibility was added as we were uncertain how such representations affect the model, while the second temporal structure choice was added to accommodate time-of-day characteristics.

2.3.1 Sequence Representation Structure: STS vs. DSSDS. Refer to Section 2.2 for definitions and examples of STS, DSSDS, and related concepts/terminology. The STS structural choice simply means that the sequence models (MC, HMM, Random Surfer) will regard and

Table 1: Graphical Models for the four general model frameworks (Sec. 2.3). Here s_t denotes the STS, i.e., s_t is the app in use at timestamp t; (u_i, d_i) denotes the DSSDS, i.e., u_i is the i-th app used for duration d_i; and c_w denotes the cluster for time window w (Sec. 2.3.2). Dotted lines indicate dependence, as dictated by the specific model types (see Table 2). Beginning in the top left, the Flat STS model simply samples s_t (the app at time t) with a model that depends on the previous times' apps (conditioned on s_j for $j < t$). Moving to the right, the Flat DSSDS samples the next app u_i conditioned on the previously used apps (u_j for $j < i$), and samples the duration d_i conditioned on u_i. The solid arrow indicates that d_i depends on u_i and is computed simply by counting and dividing frequencies seen in the training data. Moving to the Hierarchical models (bottom row), for each 1-hour time window w, a cluster c_w is sampled, depending on the previous time windows' clusters. As indicated by the rectangles, each cluster (c_w) dons its own Flat model (either STS or DSSDS), which is used to sample the app usage within that 1-hour time window. Note that for some models, dotted dependencies exist that are not shown; e.g., if the Flat STS model chosen is a Markov chain of order 2, s_t is dependent on s_{t-2} as well as s_{t-1}.

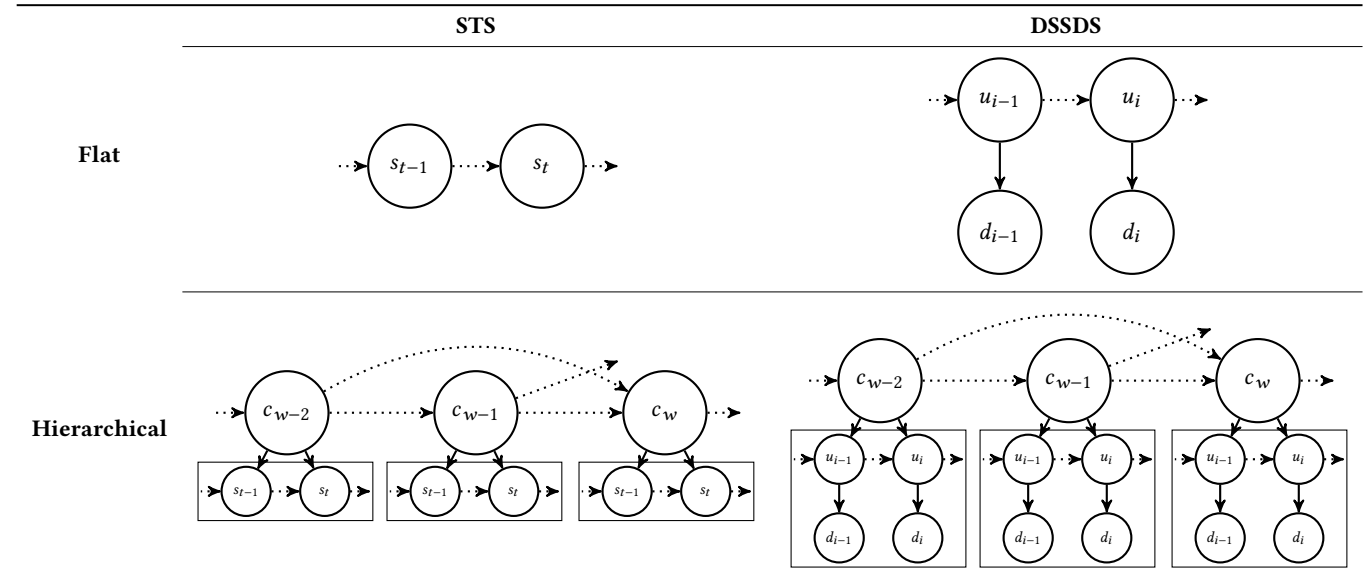


Table 2: Specific Model Types with Tested Hyperparameters

Type	Hyperparameters
Markov Chain (MC)	$m = 1, 2$
Hidden Markov Model (HMM)	$n_h = 3, 5, 7$
Random Surfer (RS)	$\alpha = [20, 20]$, $\beta_{i,j} = \delta_{i,j} = 1.1$

generate the original, full, STS sequence, which includes repeated symbols (constant subsequences) if an app is used for consecutive time intervals. On the contrary, the DSSDS structural choice indicates that the sequence model will be trained on and generate a DSS (Distinct State Sequence, containing no repeated symbols) combined with the number of intervals to remain in each symbol. More specifically, a symbol is drawn from the symbol model; then, the duration for the chosen symbol is sampled from the given symbols' duration probability distribution. The spell duration distributions are learned for each symbol from all training sequences.

2.3.2 Temporal Structure: Flat vs. Hierarchical. The Flat structure choice indicates that the sequence model (choice (3)) will be trained on and generate a whole day's sequence data (based on structural choice (1) for the representation of the day's sequence).

The Hierarchical structure choice incorporates a latent variable c that regards the time of day. For the Hierarchical structure, the input data sequences are split into time window subsequences (1 hour in our case), vectorized, and clustered via $K-$means clustering (see Figure 2). We learn K from each user via the elbow method. Next, let $v_1, \ldots, v_K$ denote the cluster centers, which are simply the expected percent each app appeared in that cluster's constituent data. Since each (1hr) time window, w, in the training data now dons a cluster assignment, $c_w \in \{1, \ldots, K\}$, a Markov distribution of order 2 is learned on the sequence of clusters observed. (This is used to generate a cluster for each hour of the day and is different than the sequence model used for generating the application sequence on 5s intervals.) Finally, a (per cluster) sequence model is trained from the data in each cluster, resulting in app sequence models $M_1, \ldots, M_K$ (based on the other structural and modeling choices (1), (3)). These K user models attempt to capture that users' behavior as observed in the data that forms that particular cluster. The generative model process is then as follows: at each time window (w) a cluster (c_w) is chosen based on the current time window, and the previous two chosen time window clusters; a subsequence for that time window (that hour of the day) is generated from the corresponding cluster's app sequence model (M_{c_w}). A priori, the benefit of this approach is that natural differences in behavior throughout the day are taken into account. As our results show, this hierarchical structure also generates higher variance sets of sequences, which tends to produce more realistic sequence sets.

2.3.3 Sequential Model Types. Markov Chain (MC): This model uses an MC model of order m, the lone hyperparameter; i.e., the probability of a symbol depends only on the previous m symbols. This model is learned from the input data sequences and is implemented using Pomegranate Python package.[3] For details on theory see, e.g., [7]. As MC order $m = 1$ is perhaps the simplest model that

[3]https://github.com/jmschrei/pomegranate

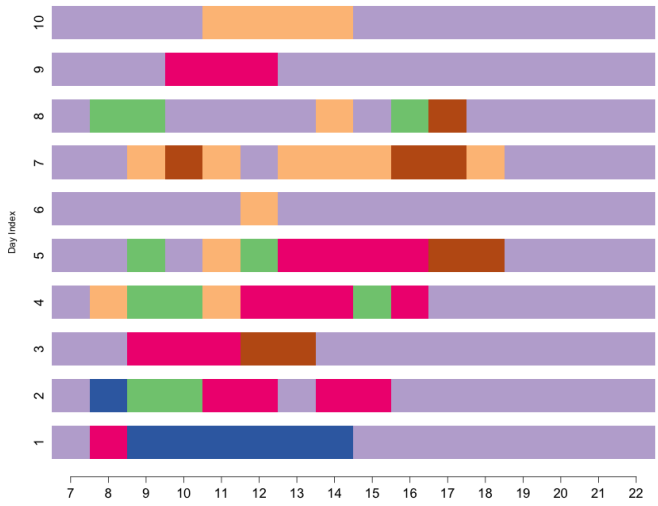


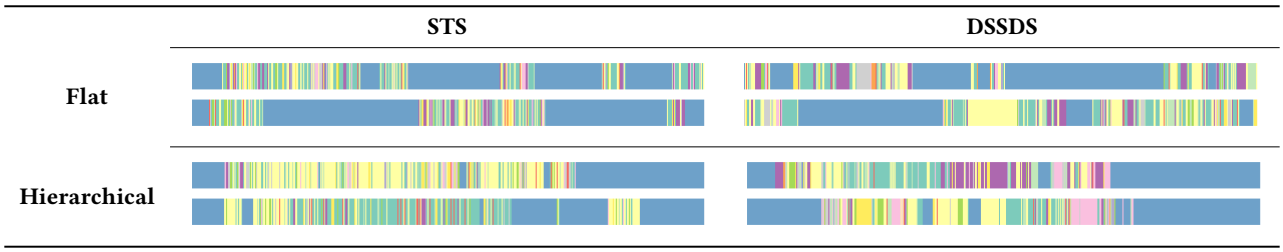
Figure 2: Clusters (c_w, for Hierarchical temporal structure) displayed for each 1-hour time window (w) and found for user data from Figure 1 with $K = 7$.

respects sequential data, we consider it the benchmark sequence model.

Hidden Markov Model (HMM): HMM models are comprised of n_h hidden or latent states (n_h is a hyperparameter), an MC model for transitioning among the hidden states (of order 1 in our case, so a transition matrix), and each hidden state is furnished with an emission probability, which is simply a distribution over the observed symbols; for details on HMM theory, see, e.g., [16]. The generative process uses the transition matrix to sample the latent state, then a symbol is sampled from emission probability for that state. This model is trained from the input data sequences using the Baum-Welch algorithm and implemented using Pomegranate Python package.[3]

Random Surfer: The Random Surfer model is the underlying model in the PageRank algorithm [4]. The "surfer" (user) moves between applications (symbols) as a mixture of a symbol transition matrix T (row-stochastic matrix giving an MC model of order 1) and a "teleportation" distribution p (multinomial distribution) on the n symbols. Specifically, at each step, the user will, with probability π, choose the next symbol based on T and the current symbol, or, with probability $1 - \pi$, sample the next symbol from the multinomial distribution of symbols, p, which is independent of the current symbol. This model was developed to mimic the behavior of a surfer browsing through web pages, choosing at each step to either follow a link on the current page, or jump to a entirely unrelated page. The model is parameterized by π, p, and T, which are learned from data by optimizing the posterior distribution (Maximum a Posteriori estimate), using a gradient ascent algorithm. See Appendix A for more details. The hyperparameters of our implementation define the prior distributions on each parameter: mixing parameter $\pi \sim \text{Beta}(\alpha)$, transition matrix rows (multinomials) $T(i, \cdot) \sim \text{Dirichlet}(\beta_i)$ and multinomial $p \sim \text{Dirichlet}(\delta)$. As shown in Table

Table 3: Structural Choices Pros & Cons: For each of the four structural choices (Flat vs. Hierarchical and STS vs. DSSDS) we present two days (7AM - midnight) of data sampled from the trained Markov chain of order 1 (benchmark model). Compare this with ground truth data and regard key in Figure 1. For both Flat and Hierarchical structure, STS models (first column) have unrealistically short spell length; that is, they jump between apps too often. DSSDS models (second column), which sample a necessarily different symbol and the duration to remain in that symbol, capture much more realistic spells. Now regarding the first row, we note that for both STS and DSSDS models, the Flat structure disregards time of day—notice early and late app usage along with long periods logged out during the middle of the day, neither of which occurred in ground truth data. As designed, the Hierarchical models (bottom row) capture time-of-day trends in the data. Finally, we note that while this data is only two days from the MC order 1 model, the advantages and pitfalls of these structural choices are representative for all models. Hierarchical DSSDS models are most realistic in terms of app duration and time-of-day patterns.



2, we use 20 for both α components, strongly encouraging equal use of T and p, and use 1.1 for all Dirichlet values.

2.4 Qualitative Modeling Results

For the hierarchical models, $K = 7$ clusters were found for this user by the elbow method. A sample of the clusters is depicted in Figure 2. We note that K varies per user in empirical experiments, so we suggest learning this parameter per user.

Our results found trends based on each of the structural decisions (choices (1) and (2)). Regard Table 3, which shows representative samples from the benchmark model, (MC, $m = 1$) for each of the four structural decisions, and compare to the ground-truth data, Figure 1. In short, regardless of sequence model choice, STS representations result in spell lengths that are all very short—i.e., apps are changed too often and too sporadically. To explain this, consider sampling the same apps for an extended period of time. Although longer spells are not infrequent, the probability of staying in the same app, say $P(a|a)$, is small since our data has so many app changes. Thus, the probability of a spell of length k is (equal for MC with $m = 1$ or otherwise close to) $P(a|a)^k$, which tends to 0 quickly as k grows.

Our results also confirm that the Flat temporal structure provides unrealistic data for the time of day (unsurprisingly as they do not regard the time of day), while the Hierarchical temporal structure does capture time-of-day trends. Overall, regardless of the sequential model choice, DSSDS Hierarchical structure is best, giving more realistic spell lengths (app use duration) and respecting trends based on the time of day.

Figure 3 depicts the best performing models under the DSSDS Hierarchical structure. The sequence models MC $m = 2$ and HMM $n_h = 5, 7$ (7 not shown) provide very realistic generated app sequences. The Random Surfer model is a bit more inclined to change apps rapidly; in particular, the large number of logged-out periods midday seem unrealistic. These preliminary results conclude that the DSSDS Hierarchical with MC or HMM are the best combinations. We note that the Hierarchical temporal structure will allow different sequential model types per cluster (e.g., M_1 is MC, while M_2 is Random Surfer), although we did not test this. In light of these results, for clusters with high entropy and overall short spells, we suspect the Random Surfer sequential model to excel.

3 D2U DEPLOYMENT

To deploy D2U inside a cyber range, we developed a custom software architecture. This architecture, shown in Figure 4, is scalable, enabling hundreds or thousands of emulators to run simultaneously, and extensible, allowing additional user actions to be added and expanded with minimal effort using python. A unique configuration file generated by our model is provided to the emulator software running on each device. When testing in our range, we used configuration files generated by the model described in this paper, as well as a model built on data collected from another of the paper authors. The emulator software then performs the specified actions, including web browsing, document creation and editing, email, ssh, ftp, shell commands, and other common behaviors.

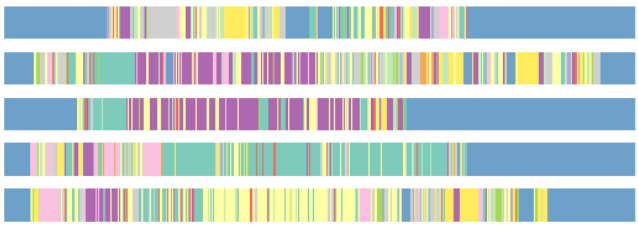
Because we focused on application sequences for this initial work, more granular actions such as mouse clicks and text entry that occur within each application are guaranteed to occur in the correct order, though we sought to add as much realism as possible. For example, the web browsing action draws from a distribution based upon the Alexa Top 100 websites in the US when visiting websites, and for tab management we based the probability of a tab being closed or a new tab being opened on the data we collected from our user. However, the workflow within each application is guaranteed to be consistent and rational. The web browser will be open prior to our code attempting to visit a website, for example.

The emulation code itself is written such that individual action types are plugins that use a common interface. This design allows new actions to be added with minimal effort by creating a python plugin that conforms to this interface. To provide network services inside cyber ranges where they may not already be available, we used Docker containers to host common services.

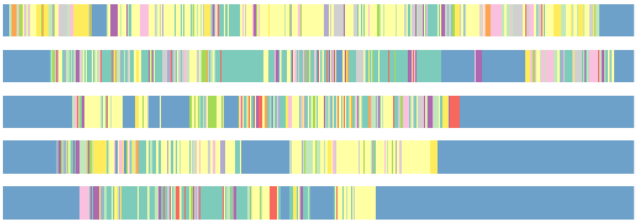
When running, the emulator software logs its actions and status using a Kafka stream. Other applications can subscribe to this

stream in order to log or analyze emulator behaviors. In our environment, there is a management server that runs alongside a website built using the MEAN (MongoDB, Express.js, AngularJS, Node.js) stack. The website displays summary statistics for emulators, such as the total number of each action type that has run, as well as statistics for individual emulators, including current action, time of last heartbeat message received for that emulator, and other relevant information. The website can also be used to send messages to individual emulators through the server, the server acting as an emulation orchestrator. These messages can be sent as interrupts, so that the emulator ceases its current action to perform the one specified, or such that they are simply added to the end of the existing action queue.

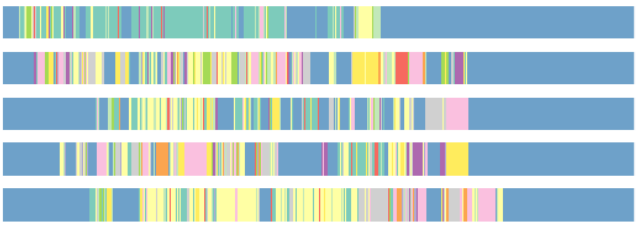
When using this architecture for data generation or testing, it is important to ensure that the emulated user and the management server are communicating out of band from the traffic generated by the emulated user nodes. Generated traffic should be from the emulated user's actions and not from contact with the management server. In cases where only a limited number of emulators are required or a more lightweight solution is desired, D2U can be run headless without connecting to either the frontend or the management server. When running headless, less verbose logging information is stored locally.



(a) Markov Chain Model ($m = 2$)



(b) Hidden Markov Model ($n_h = 5$)



(c) Random Surfer Model

Figure 3: Following Table 3, where Hierarchical DSSDS structure exhibits the most representative behavior, we now vary the model type. Five days sampled from each of MC, HMM, RS models, with Hierarchical (K = 7 clusters, 1-hour window length) DSSDS structure, trained on sequences from user in Figure 1.

4 EXISTING USER EMULATION TECHNOLOGIES

Existing user emulation technologies take one of the following approaches: (1) modeling and generating network traffic directly [12, 13, 17], (2) replaying recorded user behaviors [5, 15], (3) agent-based simulation [21], (4) using human generated configuration files (these may be approximated from real data) [20]. Solutions that fall into category (1) are fundamentally different from D2U because they replay or generate network traffic, whereas D2U emulates user behavior on end devices. In addition to the fact that it is more realistic to generate network traffic by emulating behavior on end devices, simply generating network traffic is also limited in its ability to test tools focused on anomalous user behavior.

Dutta et al. [5] developed user bots to address this shortcoming in traffic generators while testing insider threat detection systems, and was the major work we identified in category (2). These bots can be run in an enterprise system to test live deployments or in a cyber range. To enhance the realism of these bots, they replay user behavior data recorded during a study of West Point Cadets. Megyesi et al. [15] also developed a traffic generation system to create test data for deep pack inspection technologies. Their solution did not aim to emulate a specific user, but rather replayed slices of recorded user data on end devices to produce aggregate traffic with similar characteristics to that seen in operational networks.

The major work in category (3) is the DETER Agents Simulating Humans (DASH) [21] project from USC-ISI. DASH seeks to create user agents that are goal-driven, have incomplete or incorrect views of security, and respond to stress by making less rational decisions. One of the key DASH contributors, Jim Blythe, has done other significant work at the intersection of human behavior and computer security [2, 11].

Finally, there are numerous technologies that fall into category (4), so for the sake of space we include notable representative samples. GHOSTS [20] was developed by Carnegie Mellon in order to build accurate, autonomous non-player characters (NPCs) for cyber warfare exercises. It is written in C# and supports web browsing, terminal commands, email, and editing office documents. Configuration of users is accomplished using JSON files that specify what actions a user will perform and provide sample text for emails and documents. In addition, some privately owned companies, such as Skaion [4] and SimSpace [5], also provide user behavior emulation capabilities.

4.1 Relation to D2U

None of the four aforementioned approaches to user emulation use application sequence data to build generative models of user behavior. To our knowledge, D2U is the first such emulation technology.

[4] http://www.skaion.com/
[5] https://simspace.com/

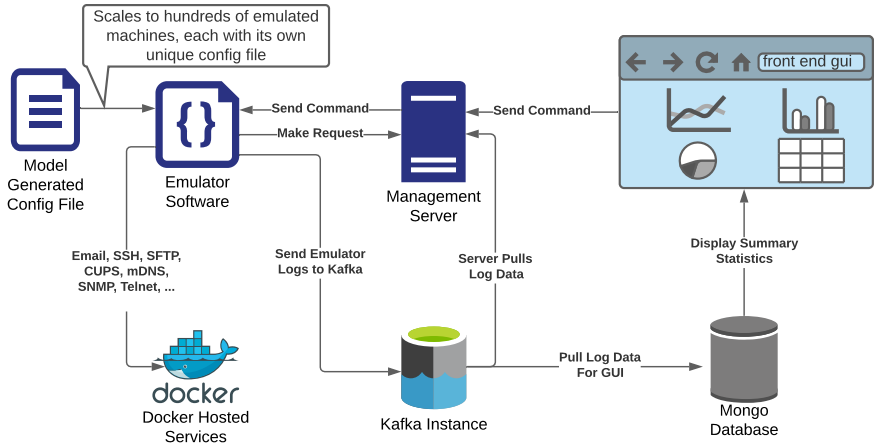


Figure 4: D2U Deployment Architecture

In contrast to approach (1), D2U emulates actual user behaviors rather than traffic patterns. In contrast to approaches (2) and (4), D2U provides limitless, realistic user behavior data that appears to be generated by the user the model was trained on. And in contrast to approach (3), D2U does not rely on accurately representing a user's mental model of security or their environment.

5 CONCLUSION

D2U provides unlimited, novel sequences of user behavior using generative models based on actual user data for use in testing and training. In this paper we described our processes for data collection and model generation, as well as qualitatively demonstrating the performance of our model and describing the software architecture used in deployment. Specifically, we considered different structural modeling decisions—STS vs. DSSDS data representations and a potentially including a temporal latent variable (Flat vs Hierarchical structure)—as well as three sequence models (Markov Chain, Hidden Markov Model, and Random Surfer) used with each structural combination. We found that, regardless of the sequence model used, DSSDS Hierarchical structure best captures spell length (duration in each app) and app use per time of day. When instantiating the DSSDS Hierarchical structure with the Markov Chains and Hidden Markov Models the best results are achieved on our data.

In the future, we plan to conduct further testing on additional users and user types (e.g. researcher, IT staff, administrative assistant, manager) to refine and expand upon our results. In this preliminary work we utilized data from a single researcher to develop and evaluate our models. Other users and user types will demonstrate unique sequential and temporal patterns that will impact model selection and hyperparameters.

We also plan to collect additional forms of user data to use as input to our models. In addition to application sequence data, file access logs, CPU usage data, browser logs, and screenshot image analysis would offer additional insight into user behaviors.

Ulimately, we hope to pioneer the next generation of emulated users: while our approach is an adequate solution for the problem we encountered (cyber range environment), a logical next step conceptually is adding a responsive and "smart" AI component to these emulated users; e.g., embedding NLP components to "read" and realistically respond to chat messages, emails, etc. In short, broadening the task from faithfully generating sequences of actions to building emulated users that seek to pass the Turing test. An example step in this direction would be to use an extra layer in the model where a user is "holding in mind" multiple applications per task and switching between them.

ACKNOWLEDGMENTS

The research is based upon work supported by the Department of Defense (DOD), Naval Information Warfare Systems Command (NAVWAR), via the Department of Energy (DOE) under contract DE-AC05-00OR22725. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of the DOD, NAVWAR, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. This technology is currently under provisional patent 202004661.US.00 as Data Driven User Emulator.

REFERENCES

[1] William H Allen. 2007. Mixing wheat with the chaff: Creating useful test data for ids evaluation. *IEEE Security & Privacy* 5, 4 (2007), 65–67.
[2] A Botello, J Lin, D Mozzacco, JE Sutton, M Spraragen, J Blythe, and M Zyda. 2010. An Agent Architecture for Large-scale Security Simulation. (2010).
[3] Timothy M Braje. 2016. *Advanced tools for cyber ranges.* Technical Report. MIT Lincoln Laboratory Lexington United States.
[4] Sergey Brin and Lawrence Page. 1998. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems* 30, 1-7 (1998), 107–117.
[5] Preetam Dutta, Gabriel Ryan, Aleksander Zieba, and Salvatore Stolfo. 2018. Simulated user bots: Real time testing of insider threat detection systems. In *2018 IEEE Security and Privacy Workshops (SPW)*. IEEE, 228–236.
[6] Alexis Gabadinho, Gilbert Ritschard, Nicolas S. Müller, and Matthias Studer. 2011. Analyzing and Visualizing State Sequences in R with TraMineR. *Journal of Statistical Software* 40, 4 (2011). https://doi.org/10.18637/jss.v040.i04
[7] Paul A Gagniuc. 2017. *Markov chains: from theory to implementation and experimentation.* John Wiley & Sons.

[8] Wael H Gomaa, Aly A Fahmy, et al. 2013. A survey of text similarity approaches. *International Journal of Computer Applications* 68, 13 (2013), 13–18.

[9] Richard W Hamming. 1950. Error detecting and error correcting codes. *The Bell system technical journal* 29, 2 (1950), 147–160.

[10] William R Knight. 1966. A computer method for calculating Kendall's tau with ungrouped data. *J. Amer. Statist. Assoc.* 61, 314 (1966), 436–439.

[11] Vijay Kothari, Jim Blythe, Sean W Smith, and Ross Koppel. 2015. Measuring the security impacts of password policies using cognitive behavioral agent-based modeling. In *Proceedings of the 2015 Symposium and Bootcamp on the Science of Security*. 1–9.

[12] Samir Mammadov, Dhanish Mehta, Evan Stoner, and Marco M Carvalho. 2017. High fidelity adaptive cyber emulation. In *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8.

[13] Frederic Massicotte, Francois Gagnon, Yvan Labiche, Lionel Briand, and Mathieu Couture. 2006. Automatic evaluation of intrusion detection systems. In *2006 22nd Annual Computer Security Applications Conference (ACSAC'06)*. IEEE, 361–370.

[14] John McHugh. 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)* 3, 4 (2000), 262–294.

[15] Péter Megyesi, Géza Szabó, and Sándor Molnár. 2015. User behavior based traffic emulator: A framework for generating test data for DPI tools. *Computer Networks* 92 (2015), 41–54.

[16] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.

[17] Lee M Rossey, Robert K Cunningham, David J Fried, Jesse C Rabek, Richard P Lippmann, Joshua W Haines, and Marc A Zissman. 2002. LARIAT: Lincoln adaptable real-time information assurance testbed. In *Proceedings, IEEE Aerospace Conference*, Vol. 6. IEEE, 6–6.

[18] Iman Sharafaldin, Arash Habibi Lashkari, and Ali A Ghorbani. 2018. Toward generating a new intrusion detection dataset and intrusion traffic characterization.. In *ICISSP*. 108–116.

[19] Matthias Studer and Gilbert Ritschard. 2014. A comparative review of sequence dissimilarity measures. (2014). https://doi.org/10.12682/lives.2296-1658.2014.33

[20] Dustin D Updyke, Geoffrey B Dobson, Thomas G Podnar, Luke J Osterritter, Benjamin L Earl, and Adam D Cerini. 2018. *Ghosts in the Machine: A Framework for Cyber-Warfare Exercise NPC Simulation.* Technical Report. CARNEGIE-MELLON UNIV PITTSBURGH PA PITTSBURGH United States.

[21] John Wroclawski, Terry Benzel, Jim Blythe, Ted Faber, Alefiya Hussain, Jelena Mirkovic, and Stephen Schwab. 2016. DETERLab and the DETER Project. In *The GENI Book*. Springer, 35–62.

[22] Stefano Zanero. 2007. Flaws and frauds in the evaluation of IDS/IPS technologies. In *Proc. of FIRST*. Citeseer.

A RANDOM SURFER (PAGERANK) MODEL IMPLEMENTATION

Let $s = [s_0, \ldots, s_m]$, with $s_t \in \{1, \ldots, n\}$, indicating the application (of n applications) observed at each time interval for $m+1$ consecutive time intervals. From each users we will have ideally multiple (l) observation sequences, and use superscript k, s_t^k to denote the k^{th} sequence of observations from that user.

We follow the PageRank [4] diffusion process to model the user's trace of applications with a mixture model, although our end goal is different—we are not interested in using the stationary vector to rank nodes (nodes represent applications in our case), but instead leverage the mixture model between a markov transition matrix and non-markov "starting" distribution for modeling the application sequence per user. Notationally, let

- $\pi \in [0, 1]$ be a mixing parameter or dampening factor, simply a binomial probability for the mixture model (coin flip) between p and T;
- $p : \{1, \ldots, n\} \to [0, 1]^n$, be the starting distribution ($\sum p(j) = 1$), a multinomial over all n applications representing the likelihood of starting a new task with that application;
- $T \in [0, 1]^{n \times n}$ be a row-stochastic transition matrix, $\forall i \sum_j T(i, j) = 1$, so that $T(i, j) = P[s_t = j | s_{t-1} = i]$, i.e., the probability the user transitions from application i to j.

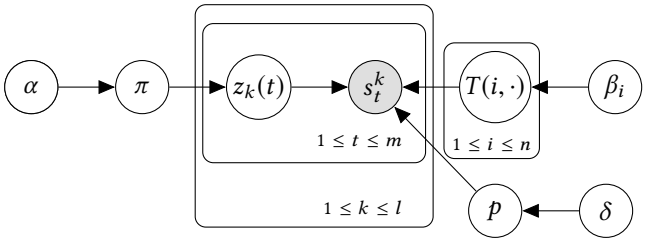


Figure 5: Plate diagram for mixture model. Each s^k is a sequence of observed state changes (applications used) and are assumed independent. We have priors as follows: mixing parameter $\pi \sim$ Beta(α), transition matrix rows (multinomials) $T(i, \cdot) \sim$ Dirichlet(β_i) and multinomial $p \sim$ Dirichlet(δ). While initial state $s_0^k \sim p$, subsequent states are sampled with a mixture from $T(s_{t-1}^k, \cdot)$ with probability π (case $z = 1$) or sampled from p with probability $1 - \pi$ (case $z = 0$).

We interpret T as giving the likelihood of application transitions from the natural workflow, as opposed to starting a new task, as modeled by p.

As depicted in the plate diagram in Figure 5, our generative model uses p as the starting distribution (for sampling s_0), then for each subsequent t, a binomial with mixing parameter π is used to decide between sampling from $T(\cdot, s_t)$ or independently drawing s_t from p. We use the "starting distribution", p, both for the initial application choice (sampling s_0) and as the second distribution in the mixture model, as users will choose an applications sometimes as the start of a new task, while other times based on an inclination from their current application (e.g., clicking a link in an email) modeled by transition matrix T. Finally, multiple sequence observations $\{s_t^k\}_k$ will be considered i.i.d. samples from this generative model. Formally,

$$\begin{aligned} P[\{s^k\}_k | \pi, T, p] &= \prod_k P[s_t^k | \pi, T, p] \\ &= \prod_k P(s_0^k | p) \prod_t P(z_t | \pi) P(s_t^k | s_{t-1}^k, z, T, p) \\ &= \prod_k p(s_0^k) \prod_t [\pi \times T(s_{t-1}^k, s_t^k) + (1 - \pi) \times p(s_t^k)]. \end{aligned} \tag{1}$$

We seek the parameters optimizing the posterior distribution (Maximum A Posteriori or MAP estimate) using conjugate priors as follows: mixing parameter $\pi \sim$ Beta(α), transition matrix rows (multinomials) $T(i, \cdot) \sim$ Dirichlet(β) and multinomial $p \sim$Dirichlet(δ), i.e.,

$$\hat{\pi}, \hat{T}, \hat{p} := \arg\max_{\pi, T, p} P(\pi, T, p | \{s^k\}, \alpha, \beta, \delta) \,. \tag{2}$$

subject to constraints

$$\begin{aligned} \sum_j T(i, j) = 1, \quad 0 \le T(i, j) \le 1 \\ \sum_j p(j) = 1, \qquad 0 \le p(j) \le 1. \end{aligned} \tag{3}$$

Hence this is an optimization over $1 + n \times (n-1) + (n-1) = (n+1)(n-1)+1$ parameters: π, $\{T(i, j) : i = 1, \ldots, n, j = 1, \ldots, n-1\}$, and $\{p(j) : j = 1, \ldots, n-1\}$, since $T(i, n) = 1 - \sum_{j<n} T(i, j)$, and $p(n) = 1 - \sum_{j<n} p(j)$. We use gradient ascent to optimize this. While

our function to be maximized is not in general concave, we plot the function for our data to find it at least appears concave in our case. Informed by this analysis, we simply walk up hill using gradient ascent until convergence, and confirm convergence at the visualized maximum.