Detecting Users from Website Sessions: A Simulation Study and Results on Multiple Simulation Scenarios

Corné de Ruijt

Faculty of Science
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
Email: c.a.m.de.ruijt@vu.nl

Sandjai Bhulai

Faculty of Science
Vrije Universiteit Amsterdam
Amsterdam, the Netherlands
Email: s.bhulai@vu.nl

***Abstract*—In this paper, we propose a click simulation model capable of simulating users' interactions with a search engine, in particular in the presence of user censoring. We illustrate the simulation model by applying it to the problem of detecting unique users from the session data of a search engine. In real click datasets, the user initiating the session may be censored, as unique users are often determined by their cookies. Therefore, analyzing this problem using a click simulation model, for which we have an uncensored ground truth, allows for studying the effect of cookie churn itself. Furthermore, it allows for studying how well clustering algorithms perform in detecting clusters of sessions that originate from a single user. To cluster sessions, we present and compare various constrained DBSCAN*-type clustering algorithms. From this comparison, we find that even though the clusters found by the best DBSCAN*-type algorithm did significantly outperform other benchmark clustering methods, it performed considerably worse compared to using the observed cookie clusters. This result remains under different simulation scenarios, though the results do improve when strengthening the user signal. While clustering algorithms may be useful to detect similar users for purposes such as user clustering, cookie tracking remains the preferred method for tracking individual users.**

Keywords–Click models; Session clustering; HDBSCAN*

I. Introduction

This paper is an extension of our previous work on click model simulation and (Internet) session clustering, presented in [1]. In particular, we provide a more detailed description of the click simulation model and (H)DBSCAN* clustering algorithm with a maximum cluster size. Furthermore, we present the performance of the session clustering algorithm on multiple simulation scenarios. The latter was only briefly discussed in our earlier work, presented at the 2020 DATA ANALYTICS conference [1].

The current Internet environment heavily relies on cookies for the enhancement of our Internet browsing experience. These cookies are small pieces of data stored in the browser after being received from a server, along with a requested web page from that server. If the Internet user pushes subsequent requests to the server, the cookie is send along, allowing the server to recognize the user and adjust its response accordingly. Hence, as cookies allow identifying users over multiple requests, they play a crucial role in session management, the personalization of websites and ads, and user tracking.

However, the usage of multiple devices, multiple browsers, and the focus on cookie management has made the problem of identifying single users over multiple sessions more complex. One study reports that so much as 20% of all Internet users delete their cookies at least once a week, whereas this percentage increases to 30% when considering cookie churn on a monthly basis [2]. Not being able to track Internet users may lead to sub-optimal behavior of search engines and online ads, as these have less information about previous search and click behavior to infer the user's preference for certain items from. As cookie churn and the usage of multiple devices censor the underlying user who is generating web traffic, we call this user censoring.

Following the 2015 ICDM and 2016 CIKM machine learning challenges [3, 4], cross-device matching has in recent years received considerable scientific attention. Cross-device matching refers to the problem of identifying individual Internet users from a set of Internet logs, where Internet users may have been using multiple devices, and are therefore tracked as separate users. These studies, however, do have some limitations. Most approaches mentioned in the literature are limited to finding pairwise matches, i.e., pairs of sessions that are likely to originate from the same user. Such inference is insufficient if one is interested in identifying exclusive session clusters consisting of more than two sessions.

Furthermore, there seems to be ambiguity in what exactly is meant by cross-device matching, or by session clustering, and to what extent successful methods applied to one problem will also work well on other problems. The ICDM and CIKM competitions consider the problem from the perspective of an online advertiser, advertising on multiple websites. Other approaches (e.g., [2, 5, 6]) consider the problem from the perspective of a single website. At this point, it is unclear whether approaches that work well on a single website are likely to be successful in the online advertisement case, and vice versa. Apart from this multi versus single website perspective, most datasets studied seem to originate from large advertisers or search engines. This raises the question of how generalizable these approaches are for websites or advertisers with less traffic or less heterogeneous searches.

To allow for sensitivity analysis in session clustering, we consider the single website perspective, and propose a single query click simulation model that allows for cookie censoring. Simulation has two main advantages: 1) by adjusting the simulation parameters, we may study how session clustering algorithms perform on websites with different user browsing

characteristics. 2) It provides a ground truth, which, due to user censoring, is only partially observed in real world datasets. Apart from the ground truth being useful in evaluating clustering algorithms, it also allows for studying the effects of user censoring on typical website statistics, such as the number of unique visitors on a website. Although several models have been proposed in the literature that could be used to create a simulation model, they only capture a specific part of search behavior and/or censoring. To our knowledge, this paper is the first to combine these models into one click simulation model with censoring.

Besides introducing the simulation model, we compare several clustering approaches on multiple simulated datasets, where all clustering methods are based on the DBSCAN* and HDBSCAN* algorithms. To measure their effectiveness, we not only consider the error in terms of typical supervised clustering error measures, such the Adjusted Rand Index, but also in terms of the error in estimating overall web statistics. These include the number of unique users, distribution of the number of sessions per user, and the user conversion distribution.

This paper has the following structure. Section II discusses relevant literature related to session clustering. Section III discusses the simulation model, adaptions of (H)DBSCAN*, and experimental set-up. Section IV discusses the results, whereas Section V discusses the implications and ideas for further research.

II. Related work

A. Click simulation

Simulating click behavior is definitely not a new concept. Chuklin et al. [7, pp. 75-77] suggests using pre-fitted click models for this purpose, where the model is pre-fitted to public click datasets. One risk of using pre-fitted models is an availability bias: can the characteristics of public click datasets, commonly provided by large search engines, easily be generalized over all search engines? Also, these datasets do not always provide the type of information one is interested in, such as the device used to initiate a session.

Fleder and Hosanagar [8] provide a generative approach for modeling user preferences, which we will discuss in more depth in Section III-A. This model can be used as an alternative to model users' preferences for clicking on items. Using pre-fitted or generative models do have a trade-off in terms of accuracy vs interpretability. E.g., the former may have an accurate estimate of users' item preferences, but it provides little understanding of why this preference over different products has a certain shape, whereas for the latter, we expect this to be vice versa.

Several authors have studied how cookie censoring occurs. E.g., [2, 9, 10] consider cookie churn, whereas [11] considers specifically cross-device behavior. Results from these studies can be used to model cookie churn dynamics in a simulation model.

B. Identifying unique users from sessions

Identifying unique users from sessions can be seen as a specific case of the entity/identity resolution problem [6]. Though, what makes this problem special is the nature of the dataset. This often consists of a large number of sessions, of which clicks and web page meta-data (such as the URL) are the main sources of information. Because of these characteristics, entity resolution algorithms that do not account for these characteristics are likely to fail in their objective.

Session matching can be applied from an online advertiser's perspective, as was the case during the 2015 ICDM and 2016 CIKM machine learning challenges [12, 13, 14, 15, 16, 17, 18, 19, 20, 21], or from the perspective of a single website [2, 5, 6]. What remains unclear is whether these two problems can be considered the same. Although in both cases the main motivation for cookie matching may be the same, e.g., increasing the click-through rate by means of personalization, the type of data is bound to be different. When advertising on multiple websites, the data seems to consist for a substantial part out of a large variety of visited URLs. Hence, proposed approaches from the advertisement perspective tend to rely heavily on natural language processing techniques [15, 16, 18, 19, 21, 22]. In case of a single website, the URLs or web pages' meta-data may be less diverse, and the "unique fingerprints" [23] users create while browsing a single website may therefore be less distinctive than on multiple websites.

Most often, both the single and multiple website perspectives are modeled as a binary classification problem. Here, a model is trained to identify whether two feature vectors describing sessions a and b originate from the same user. Striking is the success of tree-boosting methods for this task, which also in both the 2015 ICDM and 2016 CIKM machine learning competitions showed promising results. For a more in-depth discussion of the different methods applied in cross-device matching, modeled as a binary classification problem, we refer to Karakaya et al. [22]. Also worth mentioning is that many methods proposed to both the 2015 ICDM and 2016 CIKM competitions allow for overlapping user clusters. As the objective is to find pairs of sessions likely to originate from the same user, this may result in sessions a, b, c to be classified as $f(a, b) = 1$ and $f(a, c) = 1$, but $f(b, c) = 0$, f being the same user classifier. Such result may be undesirable in some practical applications.

A slight generalization of the cross-device matching problem is the cookie matching problem. Here we are given a set of sessions that are already partially clustered into users via cookies, but only partially due to some form of user censoring. I.e., cross-device matching and cookie matching only seem to differ on whether one assumes that user censoring only occurs because of cross-device usage, or also because of cookie churn. However, many approaches proposed in the literature can be applied to both problems. Hence, in these formulations, this distinction seems irrelevant. Various authors have considered the cookie matching problem, though under different names such as: 'user stitching' [6], 'visitor stitching' [5], or 'automatic identity linkage' [24]. Like in cross-device matching, these studies tend to allow for overlapping clusters.

One approach that does not allow for overlapping clusters is considered by [12], using classical bipartite matching algorithms such as the Hungarian algorithm. However, it is questionable to what extent these approaches are scalable, as the paper works with relatively small datasets. Furthermore, as users might have more than two cookies, bipartite matching will only solve a part of the problem.

Dasgupta et al. [2] also move beyond pairwise clustering. The authors consider a combination of several similarity measures to determine whether two cookies originate from the same user, and apply a greedy graph coloring algorithm to cluster a session graph into user clusters. However, since multi-device usage as we observe on websites now was not that much the case when the paper was published in 2012, the algorithm strongly relies on the assumption that only one device is used at a time. This allowed the authors to only consider non-overlapping cookies in terms of time as candidates for cookie matching, whereas in the multi-device case, such a constraint would not be able to identify unique users simultaneously using multiple devices.

In this paper, we will use the term session clustering to relate to the problem of identifying unique users from session data. We prefer this term, as our methods do not per se require having partial session clusters from cookies, something that would be the case in cookie matching. Furthermore, we seek non-overlapping clusters, whereas 'matching' relates to training a classifier to predict whether two sessions originate from the same user. However, still many of the methods discussed so far are applicable to this formulation of the problem.

We take a similar approach as [19] towards session clustering. This approach first trains a classifier that predicts whether sessions a and b originate from the same user (that is, share the same cookie in the data). Next, each session forms pairs with its K nearest neighbor (K-NN) sessions, after which each nearest neighbor is re-evaluated using the classifier on whether the session and neighbor indeed originate from the same user. All sessions included in the remaining pairs are subsequently clustered using a greedy clustering algorithm, from which all sessions in a cluster are also added to the set of session pairs.

This method shows some similarity with DBSCAN [25], where also K-NN is used to quickly identify similar data points. However, DBSCAN computes a (possibly approximate) minimum spanning tree (MST), from which a quick approximation can be made of the distances between points. Compared to the greedy clustering approach by [19], this leads to a considerable speed up. On the other hand, as DBSCAN misses a constraint on the maximum cluster size, we will turn to two of DBSCAN's descendants: DBSCAN* and HDBSCAN* [26, 27], which can quite easily be adjusted to incorporate a maximum cluster constraint.

III. Methods

A. Simulating click data with cookie-churn

We consider a simulation model that models how users behave when interacting with a search engine. We choose to simulate behavior on a search engine, and not behavior on other types of websites, as there is extensive literature on what type of parametric models are accurate for modeling user behavior on search engines [7]. Furthermore, apart from dedicated search engines, a search tool is also a common feature on websites having other purposes [28]. Hence, we believe it is likely that this behavior is also found elsewhere.

To avoid overcomplexifying the simulation model, we only consider the case in which users push one or multiple homogeneous queries to the search engine. I.e., the query itself is the same over all users, and one user may repeat this query a number of times. Users do have different item preferences for the items the search engine may return. Furthermore, the item order may be different in each Search Engine Result Page (SERP). The simulation model consists of three parts. The first part models how users navigate through the SERP, the second part models how users' utility function is determined, while the third part models how the session generating user is censored due to cookie churn or the usage of multiple devices. For reference, Table VI provides an overview of the most important variables in the simulation model.

1) Simulating SERP interactions: Two types of interactions between a user and the search engine are considered. First, users may push the (homogeneous) query to the server, and receive the SERP in response. Second, users may click on results in the SERP. At each interaction, the server checks whether the user has an active cookie. If not, a new cookie is send along with the server's response (that is, either the SERP, or the content page of a particular item in the SERP), and stored in the user's browser. We will discuss how cookie churn is modeled in Section III-A2.

All interactions are stored by the server, which provides a label for the cookie, device and query-session. This query-session is defined in terms of a set of interactions with one SERP. Hence, where in practice a browser session is typically defined by some period of interaction, we deliberately choose to model a session as a set of interactions with one SERP, irrespective of the time between two interactions with this SERP.

To simulate clicks on a search engine, we employ the Simplified Chapelle-Zhang Model (SCZM) [29]. Although this model is known in the literature as the Simplified Dynamic Bayesian Network model (SDBN), we renamed the model as it is only a specific case of a Dynamic Bayesian Network. We choose to use SCZM for two reasons: 1) the model, though simple, seems to perform reasonably well in comparison with other parametric click models when predicting clicks [7]. 2) SCZM captures the ordering effect of items in the SERP. I.e., users may not always reflect their preferences correctly in their clicks, as their behavior is also determined by how items are ordered. Including this 'cascade effect' provides more realistic results.

To describe the simulation model, the following notation will be used. Let $i \in \{1, \ldots, n\}$ be a query-session, which produces a SERP of unique items $\mathcal{S}_i \subseteq \mathcal{V}$, with $\mathcal{V} = \{1, \ldots, V\}$ the set of all items, indexed by v. We assume all SERPs $1, \ldots, n$ to have the same number of items T. Let $u_i \in \mathcal{U}$ denote the user initiating query-session i, with $\mathcal{U} = \{1, \ldots, U\}$ the set of all users. The user index u is used instead of u_i in case the precise query-session i is irrelevant. $r_i(t)$ denotes the item at position t in query-session i. Likewise, $r_i^{-1}(v)$ gives the position of item v in query-session i, and $r_i^{\max}$ denotes the largest position of a clicked item in $\mathcal{S}_i$, where $r_i^{\max} = 0$ if no items were clicked during query-session i.

SCZM considers three latent variables: $R_v^{(i)}$ denotes whether user u_i is attracted to item v during query-session i. This variable is also known as the relevance of item v for the user initiating session i. The probability of item v being relevant to user u in session i is given by $\phi_{u,v}^{(R)}$. $S_v^{(i)}$ denotes whether user u_i is satisfied with item v after having clicked the item, which happens with probability $\phi_{u,v}^{(S)}$, and $E_t^{(i)}$ denotes

whether user u_i will evaluate the item in position t during query-session i. Whether the item at position t in SERP i is clicked is denoted by the binary variable $y_t^{(i)}$.

The model follows the cascade hypothesis, that is, it assumes a user always evaluates the first item ($E_1^{(i)} = 1$ for all $i = 1, \ldots, n$), after which the user decides to evaluate subsequent items in the list according to the perceived attraction and satisfaction of the previous evaluated items in the list, according to

$$E_1^{(i)} = 1; \tag{1}$$

$$\mathbb{P}(R_v^{(i)} = 1) = \begin{cases} \phi_{u_i,v}^{(R)} & \textit{if } v \in \mathcal{S}_i \\ 0 & \textit{otherwise} \end{cases}; \tag{2}$$

$$\mathbb{P}\left(S_v^{(i)} = 1 \mid y_{r_i^{-1}(v)}^{(i)} = 1\right) = \begin{cases} \phi_{u_i,v}^{(S)} & \textit{if } v \in \mathcal{S}_i \\ 0 & \textit{otherwise} \end{cases}; \tag{3}$$

$$y_t^{(i)} = 0 \Rightarrow S_{r_i(t)}^{(i)} = 0; \tag{4}$$

$$E_{t-1}^{(i)} = 1, S_{r_i(t-1)}^{(i)} = 0 \iff E_t^{(i)} = 1, \quad t > 1; \tag{5}$$

$$y_t^{(i)} = 1 \iff E_t^{(i)} = 1, R_{r_i(t)}^{(i)} = 1. \tag{6}$$

To come up with reasonable values for $\phi_{u,v}^{(R)}$ and $\phi_{u,v}^{(S)}$, we used the same approach as in [8]. That is, users are represented by the vectors $\boldsymbol{\eta}_u = \left(\eta_1^{(u)}, \eta_2^{(u)}\right)$, $u \in \mathcal{U}$, where $\eta_1^{(u)}$ and $\eta_2^{(u)}$ are drawn from two independent standard normal distributions. Likewise, all items can be represented by the vectors $\boldsymbol{\psi}_v = \left(\psi_1^{(v)}, \psi_2^{(v)}\right)$, where again $\psi_1^{(v)}$ and $\psi_2^{(v)}$ are drawn from independent standard normal distributions. The probabilities $\phi_{u,v}^{(R)}$, and $\phi_{u,v}^{(S)}$ are then determined by the multinomial logits

$$\phi_{u,v}^{(R)} = \frac{e^{\omega_{u,v} + \nu^{(A)}}}{\sum_{v' \in \mathcal{V} \setminus \{v\}} e^{\omega_{u,v'}} + e^{\omega_{u,v} + \nu^{(A)}}}, \tag{7}$$

$$\phi_{u,v}^{(S)} = \frac{e^{\omega_{u,v} + \nu^{(S)}}}{\sum_{v' \in \mathcal{V} \setminus \{v\}} e^{\omega_{u,v'}} + e^{\omega_{u,v} + \nu^{(S)}}}, \tag{8}$$

with

$$\omega_{u,v} = -q \log \delta(\boldsymbol{\eta}_u, \boldsymbol{\psi}_v). \tag{9}$$

Here δ is some distance function, in our case Euclidean distance. $q \in \mathbb{R}^+$ is some constant value that models the users' preferences towards nearby items, and $\nu^{(A)}$, $\nu^{(S)}$ are salience parameters for attraction and satisfaction respectively.

The order in which items are presented is determined as follows. First, during a warm-up phase, we simulate clicks for $U_{\text{warm-up}}$ users, while randomly ordering the items such that all have equal probability of being positioned at positions $t = 1, \ldots, T$. Next, we estimate the overall probability of each item being found attractive, and we use these probabilities as weights to determine the item order for subsequent query-sessions. More specifically, for each query-session i, we draw items $\mathcal{S}_i$ from a multinomial distribution with parameters $\hat{\phi}_v / \sum_{v \in \mathcal{V}} \hat{\phi}_v$, $v = 1, \ldots, V$; without replacement. The estimate of overall attraction is given by [7, p. 26],

$$\hat{\phi}_v = \frac{1}{|\mathcal{I}_v|} \sum_{i \in \mathcal{I}_v} y_{r_i^{-1}(v)}^{(i)}, \tag{10}$$

with

$$\mathcal{I}_u = \left\{ \mathcal{S}_i : v \in \mathcal{S}_i, r_i^{-1}(v) \leq r_i^{\max} \right\}. \tag{11}$$

To avoid $\hat{\phi}_v$ to be (close to) zero, we impose a minimum probability of 10^{-5} for all $v \in \mathcal{V}$.

2) Cookie censoring: Cookie censoring is incorporated in the simulation model in two ways: by incorporating time and letting cookies churn after some random time $\mathcal{T}$, and by switching from device d to some other device d'. First, we consider the cookie lifetime $\mathcal{T}_{u,o,d}^{\text{cookie}}$ for the o-th cookie of user u on device d, and the user lifetime $\mathcal{T}_u^{\text{user}}$. Whenever the cookie lifetime of cookie o ends, but the current user lifetime is strictly smaller than $\mathcal{T}_u^{\text{user}}$, a new cookie o' is created, which lifetime is drawn from the cookie lifetime distribution F^{cookie}. For a period of $\mathcal{T}_{u,o',d}^{\text{cookie}}$, all click behavior of user u on device d will now be registered under cookie o'.

Second, after each query-session a user may switch from device d to d', which happens according to transition matrix P. Whenever a user switches devices, we consider whether the user has used this device before. If not, a new cookie o' is created, and we draw a new cookie lifetime from F^{cookie}. However, the cookie lifetime $\mathcal{T}_{u,o,d}^{\text{cookie}}$ does not end prematurely when the user switches from device d to d'. If later on the user switches back to device d while the cookie lifetime $\mathcal{T}_{u,o,d}^{\text{cookie}}$ has not ended, the behavior of user u is again tracked via cookie o until another device switch occurs or cookie o churns.

Putting this censoring into practise requires us to provide five distributions: 1) a distribution F^{abs} for the time between query-sessions, which following [10] we will refer to as the *absence time*, 2) a distribution for the cookie lifetime (F^{cookie}), 3) a distribution for the user lifetime (F^{user}), 4) the device transition matrix P, and 5) the initial device probability F^{device}.

For the absence time distribution, we use some results from [10]. Although Dupret and Lalmas [10] fitted a Cox survival model to user absence data in order to estimate user lifetimes, we refitted the data mentioned in the paper with a different model for two reasons. First, there is ambiguity in the method used to model absence time. The authors fit a Cox survival model with one covariate. As the (log-)likelihood of a Cox survival model omits the estimation of the base hazard, the method for estimating this base hazard should be provided (e.g., the Breslow estimator). However, the paper does not report which method was used to fit the baseline hazard. Second, results from Dasgupta et al. [2] on cookie churn suggests that, when taking into account longer periods than 7 days, absence time has a fat-tailed distribution. We found that a Pareto-I with scale parameter $m = 1$ and shape $\alpha = 0.11$ seems to fit the data from [10] approximately well. This distribution was therefore used to model F^{abs}. To allow for absence times smaller than 1 (but still positive), we subtracted one from all drawn lifetimes.

To model the cookie lifetime, we used the results from [2], who find that a hyper-exponential distribution with one over the rate being equal to 50 seconds (with probability .06), 25 minutes (with probability .07), 14 hours (with probability .07), 15 days (with probability .18), and 337 days (with probability .62), fits reasonably well. Here, cookie lifetime is defined as the time difference between the first and last observed action from a single cookie. We consider time at a minute scale,

and therefore rounded up the first phase (50 seconds) to one minute.

The user lifetime is obtained by sampling from N cookie lifetime distributions, where N itself is drawn from a geometric distribution with parameter ρ. As the cookie lifetime distribution is modelled as a hyper-exponential, we will refer to this distribution as a repeated hyper-exponential distribution. Although we sample from the cookie life time distribution, the user lifetime is independent from the cookie lifetimes: they only share the underlying hyper-exponential distribution, not the realizations of that distribution.

To model device transition matrix P, we use the results from [11], who study device transitions between four devices: a PC, tablet, smartphone and game console. We adopted the transition probabilities found in this paper, where we dropped the game console as the found transition probabilities from and to this device were negligible. After dropping the game console, the probabilities were normalized to obtain transition matrix P. The initial device probability distribution F^{device} is also obtained using the results from [11], and is modeled as a multinomial distribution with parameter $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3)$; π_1, π_2, π_3 being the probability of the PC (Dev. 1), tablet (Dev. 2), and smartphone (Dev. 3) being the first device respectively. The normalized initial and transition probabilities from [11] are given by Table I.

TABLE I
INITIAL DEVICE AND DEVICE TRANSITION PROBABILITIES ADOPTED FROM [11]

	$\boldsymbol{\pi}$	Dev. 1	Dev. 2	Dev 3
Dev. 1	.64	.9874	.0042	.0084
Dev. 2	.11	.00256	.9697	.0046
Dev. 3	.25	.029	.0018	.9773

3) Summary of the simulation procedure: The entire simulation procedure is given in Algorithms 1 and 2 (see Appendix). The former describes how user preferences are obtained and how the overall popularity is determined, whereas the latter describes how clicks and cookie churn are simulated over a set of users.

For convenience, we have written the set of warm-up users as $\mathcal{U}_{\text{warm-up}}$, $\hat{\boldsymbol{\phi}} = (\hat{\phi}_1, \ldots, \hat{\phi}_V)$, and $\mathbf{y}_i = (y_1^{(i)}, \ldots, y_T^{(i)})$. The location and scale parameter of the Pareto-I distribution are written as m and α, whereas the rate and rate probability of the hyper-exponential distribution are given by the vectors $\boldsymbol{\lambda}$ and $\mathbf{p}$. Last, let I_d be a 3×3 matrix where the d-th column contains all ones, whereas the rest of the matrix contains all zeros.

The simulation iterates over all users, where for each user new query-sessions are simulated until the user lifetime has elapsed. For each user, first the initial device is drawn, along with a cookie lifetime for that user on that device, and the total user lifetime. Next, query-sessions are simulated for each user in four steps. First, $\mathcal{S}_i$ is (iteratively) drawn using the overall item popularity $\hat{\boldsymbol{\phi}}$, and we simulate clicks using the SCZM model described in Section III-A1, which are stored in dataset $\mathcal{D}$. Second, we simulate the time until the next session. Third, the device of the next session is determined. Fourth, we check whether the last cookie on the new device has churned.

Algorithm 1: User simulation procedure

1 Draw $\eta_1^{(u)}, \eta_2^{(u)}, \psi_1^{(v)}, \psi_2^{(v)}$ i.i.d. from a standard normal distribution for all $v \in \mathcal{V}$ and $u \in \mathcal{U}$;
2 Compute similarities $\omega_{u,v}$ according to (9);
3 Compute the probability of attraction and satisfaction, using (7);
4 Set $\hat{\phi}_v \leftarrow 1$ for all $v \in \mathcal{V}$;
5 $\mathcal{D}_{\text{warm-up}} \leftarrow$ SIMULATE_CLICKS($\mathcal{U}_{\text{warm-up}}$);
6 Recompute $\hat{\boldsymbol{\phi}}$ according to (10);
7 $\mathcal{D} \leftarrow$ SIMULATE_CLICKS($\mathcal{U} \setminus \mathcal{U}_{\text{warm-up}}$);
8 **return** $\mathcal{D}$;

If so, a new cookie is created with a corresponding new cookie lifetime.

Although Algorithm 2 assumes all users arrive at $t = 0$, we shift all times after the simulation to obtain click behavior spread out over time. Here we assume a Poisson arrival process with rate γ. I.e., the first query-session of user u starts some exponentially distributed time after the initial query-session of user $u-1$. Note that these inter-first session times only depend on the time of the first session of the previous user, not on any other subsequent behavior of that user.

B. Session clustering

1) (H)DBSCAN:*

a) Hierarchical clustering using Minimum Spanning Trees (MST): Before we discuss the adjustment made to the HDBSCAN* and DBSCAN* algorithms, we will first briefly describe the two algorithms. We first discuss the overlapping part in both algorithms, after which we discuss their differences. Following the terminology by [26, 27] and [30], let $X = \{X_1, \ldots, X_n\}$ be a set of data points, let $\kappa_k(X_i)$ be the distance from point X_i to its k-th nearest neighbor (for some given value of $k \in \mathbb{N}$), and let $\delta(X_i, X_{i'})$ be some distance measure between points X_i and $X_{i'}$. Based on this original distance measure, DBSCAN* considers an alternative distance measure, which is named the *mutual reachability distance*, and is defined as follows:

$$\delta_k^{\text{mreach}}(X_i, X_{i'}) = \begin{cases} \max\{\kappa_k(X_i), \kappa_k(X_{i'}), \delta(X_i, X_{i'})\} & X_i \neq X_{i'} \\ 0 & X_i = X_{i'} \end{cases}. \quad (12)$$

Although DBSCAN* does not specify the exact distance measure δ, we will (like in Section III-A1) assume this is Euclidean distance. The main motivation for introducing this mutual reachability distance is to better identify different clusters with high density of arbitrary shape, as the measure tends to push different high density clusters further apart.

Given the mutual reachability distance, (H)DBSCAN* represents each data point as a node in a complete weighted graph G, where the weights are simply the mutual reachability distances between data pairs. Using G, the algorithm first computes a minimum spanning tree (MST), which allows for fast identification of clusters. The MST is also used to approximate distances: the distance between two non-adjacent points X_i and $X_{i'}$ in the MST can be approximated by the path length $X_i \rightarrow X_{i'}$ in the MST. At the same time, this

distance is a lower bound on the actual distance: otherwise, $X_i \to X_{i'}$ would be adjacent in the MST.

From this MST, one can build a dendogram of the data points in an agglomerative manner. First, (H)DBSCAN* assigns each data point $X_1, \ldots, X_n$ to separate clusters $\mathcal{B}_1^0, \ldots, \mathcal{B}_n^0$. Here the superscript is used to indicate the hierarchy level of the cluster, which at this stage is zero. Second, it iterates through the edges in G, increasing in terms of their weights. For some edge $(i, i^{'})$ having the smallest edge weight, it finds the two clusters with the highest hierarchy levels $h_i^{\max}$ and $h_{i'}^{\max}$, to which i and $i^{'}$ are assigned to respectively. Next, it and creates a new cluster $\mathcal{B}_j^{\max\{h_i^{\max}, h_{i'}^{\max}\}+1}$, which includes all data points included in the highest hierarchy clusters to which X_i and $X_i^{'}$ were previously assigned to. If this process is repeated for all edges in G, the last edge will create a cluster containing all data, which occurs at level H.

b) DBSCAN:* The construction of the dendogram occurs both in DBSCAN* and HDBSCAN* in the same manner. However, as both methods wish two find non-overlapping clusters, the two methods split ways from there. In DBSCAN*, one would take some value $\epsilon \in \mathbb{R}^+$, and remove all cluster merges in the dendogram that were merged with a weight strictly greater than the chosen maximum distance ϵ. This would lead to a set of disconnected binary trees $\mathcal{T}$, and a set of singleton points $\mathcal{N}$. The singleton points are points for which their k-th nearest neighbor is already at a further distance than ϵ, and these points are consequently labeled as noise. All data points in one tree $\tau \in \mathcal{T}$ are labeled as one cluster.

c) HDBSCAN:* The underlying assumption of cutting the dendogram at level ϵ, is that all clusters have (approximately) the same density. This density is in HDBSCAN* approximated by $\theta = 1/\epsilon$, i.e., close points imply high density. HDBSCAN* allows for different cut-off levels of ϵ, or similarly of θ, where the optimal cut-off level for some cluster is determined via the notion of *relative excess of mass*, which we will introduce in a moment.

More precisely, let M be some given minimum cluster size. To somewhat simplify notation, we let index j refer to any cluster, irrespectively of hierarchy h, such that h can be dropped. HDBSCAN* first creates a *condensed tree* from the dendogram in the following way. It starts at the root of the dendogram, having label j_0, and considers its children. These were merged at some density $\theta_{j,j'}$, merging two clusters with labels j and $j^{'}$. It then considers three options: 1) if both children have less than M points, all points in $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ "fall-out" of the cluster at density $\theta_{j,j'}$, implying that for densities greater than $\theta_{j,j'}$ all points in $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ are labeled as noise. 2) If only one cluster $\mathcal{B}_j$ has less than M points, all points in $\mathcal{B}_j$ fall-out at density $\theta_{j,j'}$, while the parent cluster label (j_0) is now continued for all observations in $\mathcal{B}_{j'}$. I.e., we replace label $j^{'}$ by j_0, and as a result the exact cluster j_0 now refers to depends on whether we pick a density larger or smaller than $\theta_{j,j'}$. 3) If both children have more than M observations, clusters $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ keep their labels j and $j^{'}$. I.e., label j_0 is not continued, and clusters $\mathcal{B}_j$ and $\mathcal{B}_{j'}$ are considered separate clusters for densities larger than $\theta_{j,j'}$. After both children have been relabeled, this process is repeated using these new labels until all nodes have been relabeled.

The resulting condensed tree is essentially still the same as the original dendogram, but with different labels. I.e., by continuing the parent (option 2), some labels now may refer to different clusters, dependent on density θ. Let $\{1, \ldots, m\}$ be the resulting set of labels from relabeling. For each label $j \in \{1, \ldots, m\}$, let $\mathcal{B}_j$ be the set of observations labeled j at the minimum density for which j exists. Furthermore, let $\theta_j^{\max}(X_i)$ and $\theta_j^{\min}(X_i)$ be the densities at which observation X_i falls off cluster j and the density at which X_i first occurs in cluster j respectively. Note that $\theta_j^{\min}(X_i)$ is either zero (when j is the label continued from the root node), or the density at which cluster j splits off from its parent, hence it has the same value for all $X_i \in \mathcal{B}_j$.

Clusters $\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ may still be overlapping. To find non-overlapping clusters, HDBSCAN* introduces the *relative excess of mass* of cluster j as $\sigma(j)$, which is defined by:

$$\sigma(j) = \sum_{X_i \in \mathcal{B}_j} \left[\theta_j^{\max}(X_i) - \theta_j^{\min}(X_i)\right]. \tag{13}$$

The relative excess of mass has an intuitive argument for clustering. Large values for $\sigma(j)$ imply that when increasing the density, the cluster remains more or less intact (apart from some noise points splitting off at higher densities). As a result $\theta_j^{\max}(X_i) - \theta_j^{\min}(X_i)$ becomes large. I.e., the relative excess of mass can be used as a measure of cluster quality. Hence, HDBSCAN* optimizes the sum of relative excess of mass over a subset of clusters $\{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ such that this subset is non-overlapping.

2) Introducing maximum cluster sizes to HDBSCAN and DBSCAN*:* To return to the problem at hand: identifying small session clusters from the set of all sessions that may be originating from the same user, HDBSCAN* and DBSCAN* can obviously be used for this purpose. Apart from the earlier discussed benefit of speed by clustering via MST, incorporating noise points would also intuitively make sense in identifying potential users from sessions: we would expect that quite a large (though unknown) percentage of all sessions might still be from users only initiating a single session.

By tweaking parameters k, (the k-th nearest neighbor in nearest neighbor distance κ_k), ϵ (dendogram cut-off point in case of DBSCAN*), and M (minimum number of points before a cluster is considered noise in HDBSCAN*) one can obtain session clusters that obey a maximum cluster size $\beta \in \mathbb{N}$. However, some early experiments with DBSCAN* and HDBSCAN* showed that the resulting clusters tended to either very large clusters, or labeled (almost) every point as noise. For that reason, we chose to adjust both algorithms, in order to obtain more small clusters having a size smaller than β.

To impose the clusters to be more fine grained, we impose a restriction on the maximum cluster size of the clusters found by (H)DBSCAN*. We do so in three different ways: max-size DBSCAN* (MS-DBSCAN*) imposes this restriction on DBSCAN*, whereas MS-HDBSCAN*$^-$ and MS-HDBSCAN*$^+$ are two ways to impose the restriction on HDBSCAN*.

First we consider MS-DBSCAN*. This algorithm is only a slight adaptation to the DBSCAN* algorithm described in Section III-B1. Given the binary trees $\mathcal{T}$, obtained by removing all nodes and edges in the dendogram above distance ϵ, we further remove all cluster nodes j for which $|\mathcal{B}_j| > \beta$. Doing so results in two new sets: $\widetilde{\mathcal{N}}$ and $\widetilde{\mathcal{T}}$, again representing singleton

points that we assume to be noise, and all points in a tree $\tau \in \widetilde{\mathcal{T}}$ receive the same cluster.

Second are the adaptations of HDBSCAN*. The first steps of these two adaptations are the same. First, all clusters $\mathcal{B}_j \in \{\mathcal{B}_1, \ldots, \mathcal{B}_m\}$ having $|\mathcal{B}_j| > \beta$ are removed from the dendogram. This, like in DBSCAN*, gives two sets: noise points $\mathcal{N}$ and trees $\mathcal{T}$. Second, for each sub-tree $\tau \in \mathcal{T}$, we again optimize the the total relative excess of mass subject to non-overlapping clusters. The difference between MS-HDBSCAN*$^-$ and MS-HDBSCAN*$^+$ arises when a leaf node of the condensed tree (that is, a label that does not split at some larger density into two new labels, though noise points may split off) of some condensed sub-tree τ is in the set of optimal non-overlapping clusters. In case of MS-HDBSCAN*$^-$, all observations in $\mathcal{B}_j$ are given the same label, whereas in case of MS-HDBSCAN*$^+$, these are considered noise.

3) Session cluster re-evaluation: As one might have noticed, so far we have not used any information from the cookies. I.e., knowing which sessions have the same cookie could provide valuable information about the underlying user. In particular, we wish to train a model that can function as an alternative to standard distance measures δ, such as Euclidean or Manhattan distance, which we then again can plug into the adapted (H)DBSCAN* algorithms described in Section III-B2.

Obtaining session clusters with re-evaluation is done as follows. Assume we have a trained classifier $\hat{f}(X_i, X_{i'})$, which returns the probability of X_i and $X_{i'}$ originating from the same user. First, like in [19], we find for each point X_i the K nearest neighbors, which gives us a set $\mathcal{X}$ of all nearest neighbor session pairs. Second, we compute $-\log(\hat{f}(X_i, X_{i'}))$ for all $(X_i, X_{i'}) \in \mathcal{X}$, and fill this into a (sparse) $n \times n$ distance matrix W. For all pairs $(X_i, X_{i'}) \notin \mathcal{X}$, we assume the distance is some large value $\delta_{\max}$, which allows us to store W efficiently, and greatly speeds-up computations compared to evaluating all pairwise same user probabilities. Distance matrix W can subsequently be used as distance measure δ in the algorithms discussed in Section III-B to obtain new session clusters.

To train classifier $\hat{f}$, we first cluster a training set according to one of the models discussed in Section III-B, using Euclidean distance for δ. Second, for each cluster we add all unique session pairs into some training set $\mathcal{X}_{\text{clust}}$. Next, we start using the observed cookies: we treat each cookie as a cluster and determine all session pairs in these cookie clusters, where this set of pairs is denoted by $\mathcal{X}_{\text{cookie}}$. To determine for each session pair $(X_i, X_{i'})$ the correct label, we use the information from the observed cookie. If X_i and $X_{i'}$ have the same observed cookie, we set the target variable to one, whereas it equals zero otherwise. The final training set $\mathcal{X}_{train}$ is obtained by undersampling from $\mathcal{X}_{\text{clust}} \cup \mathcal{X}_{\text{cookie}}$.

Note that obtaining negative labeled training pairs from a point its K nearest neighbors follows the assumption that these are indeed more likely to be negatives than positives. If this assumption holds, sampling negatives from the nearest neighbors would intuitively help the classifier to learn more subtle patterns. I.e., the K nearest neighbors are close in terms of the common distance measure, but not according to the classifier.

4) DBSCAN with random clusters:* To benchmark the clustering approaches just discussed, we consider the following benchmark. We first cluster the sessions using the ordinary DBSCAN* algorithm, in which way we obtain initial clusters $\mathcal{B}_1^0, \ldots, \mathcal{B}_m^0$. Next, for each cluster $\mathcal{B}_j^h$ ($h \in \mathbb{N}$, with initially $h = 0$), if $|\mathcal{B}_j^h| > \beta$, we iteratively select $\min\{s_{j,h}, |\mathcal{B}_j^h|, \beta\}$ points uniformly at random from $\mathcal{B}_j$ to form a new cluster $\widetilde{\mathcal{B}}$, and update $\mathcal{B}_j^{h+1} \leftarrow \mathcal{B}_j^h \setminus \widetilde{\mathcal{B}}$. Here, $s_{j,h}$ is drawn from a geometric distribution with $p = 0.5$. This process continues until for all $j \in \{1, \ldots, m\}$: $|\mathcal{B}_j^h| \leq \beta$ for some h, at which the remaining points in $\mathcal{B}_j^h$ are labeled as one cluster.

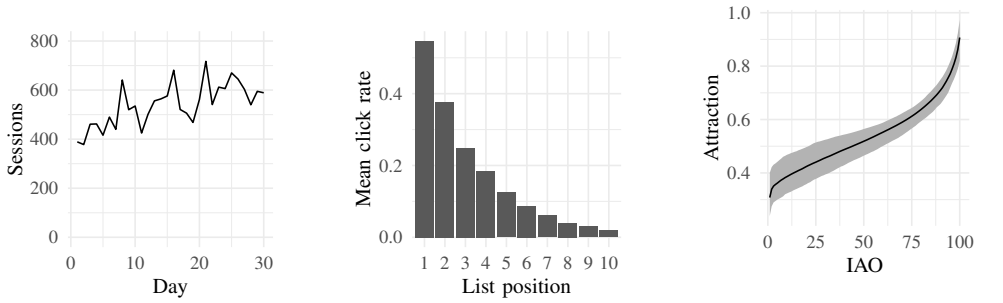
Intuitively, we selected this benchmark as it captures the higher level hierarchy clustering of DBSCAN*, but not the low level clusters (as these clusters are picked at random). Therefore, comparing the previous methods with this random clustering approach allows us to assess whether the smaller size clusters reveal more information than the larger ones.

C. Experimental setup

1) Simulation parameters: Our experimental design consist of two steps. First, we consider a simulation base case on which we evaluate the clustering approaches discussed in Section III-B. In this base case, users' first query arrival follows a Poisson process with rate $\gamma = 0.2$ (minutes), after which subsequent behavior over time of a particular user is modeled according to F^{abs}, F^{cookie}, F^{user}, F^{device}, P, and $\boldsymbol{\pi}$, of which the parameters were already given in Section III-A2. We used $U = 20,000$ users with $U_{\text{warm-up}} = 2,000$ (10%). Furthermore, we removed the first 250 sessions (not part of the first $U_{\text{warm-up}}$ users, who were only used to estimate the overall item popularity), as these would likely all be first sessions from new arriving users, and therefore including them may lead to a bias in the data. Likewise, we removed all observations after $43,200$ minutes (30 days) to avoid the opposite bias: not having any new users. Users could pick from $V = 100$ items, and we choose as maximum list size $T = 10$.

For parameters that could not be adopted from the literature, we tried several parameter values and looked at three characteristics. First, we considered whether the click probability is decreasing in list position. Second, whether the attraction/satisfaction is centered around 0.5, with a standard deviation of approximately 0.1 to 0.2. Third, whether all sessions are somewhat spread out over time. This lead us to choosing users' preference for nearby items $q = 1$, user lifetime phases geometric parameter $\rho = 0.5$, and salience parameters $\nu^{(A)} = \nu^{(S)} = 5$. Figure 1 shows the three base case characteristics for the resulting simulated base case used in further inference. In the second step of the experimental design, we made adjustments to the latter parameters, that is, those not adapted from the literature. These adjustments will be discussed in Section IV-B.

2) Features and MS-(H)DBSCAN hyper-parameter settings:* The simulated dataset was split into a training and test set according to a 70/30 split over the users. I.e., users always are entirely in the training set, or entirely in the test set. For each session, we used the session's start time, observed session count (as observed by the cookie), number of clicks, and whether the session's SERP has at least one click as features. Furthermore, to obtain a vector representation of the items and interactions with the SERP, we first computed a bin-count table. This table contains per item the total number of clicks, skips (no click), and the log-odds ratio between clicks and



(a) Sessions per day.

(b) Mean click rate per list position.

(c) Mean attraction and the area between 0.05 and 0.95 quantile over the users' Item Attraction Order (IAO)

Figure 1. Summary of base case simulation.

skips over 30 percent of all sessions, which combined were used as item vector representations.

Next, for each session i, we concatenated all item vectors $\boldsymbol{\psi}_v$, $v \in \mathcal{S}_i$, in order of their position, resulting in some vector $\boldsymbol{a}_i$ with $3T$ elements. Additionally, we created four more session vectors. The first of these vectors is obtained by multiplying $\boldsymbol{a}_i$ with a vector containing ones at those positions where a click occurred, whereas for the second vector, $\boldsymbol{a}_i$ is multiplied with a vector containing ones at positions where the item was skipped (=not clicked). The third vector is obtained by multiplying $\boldsymbol{a}_i$ with a vector containing ones at the last clicked position. To obtain the fourth vector, $\boldsymbol{a}_i$ is multiplied with a vector of list positions for each item. In all cases, the vector multiplication is element-wise. Next, all five session vectors were concatenated to obtain one session vector representation.

The resulting concatenated session vector was further treated by computing all second order polynomial features, after which we normalized and applied the Yeo-Johnson [31] power scaler to make the distribution of each feature more Gaussian-like. We reduced the vector's dimension using a principle component analysis using seven principle components, the latter was chosen using the elbow method.

For each method, we experimented with $k \in \{1, 3, 5\}$ (here k as in κ_k, the distance to the k-th nearest neighbor). For DBSCAN*-like algorithms, we tried

$$\epsilon \in \left\{ \left(q_{\max} \left(q_{\min} / q_{\max} \right)^{\ell / N} \right) \middle| \ell \in \{1, \ldots, N\} \right\}, \qquad (14)$$

with $N = 9$ and $q_{\min}$, $q_{\max}$ the minimum and maximum Euclidean distance, obtained by computing all pair-wise distances over 1,000 sampled session vectors. For HDBSCAN*-type algorithms, we set minimum cluster size $M = 2$.

For re-evaluation models, we took the approach already explained in Section III-B3. To train classifier $\hat{f}(\mathbf{a}_i, \mathbf{a}_{i'})$, we first run MS-DBSCAN* with the best found values for k and ϵ from earlier validation of MS-DBSCAN* on the training set to, together with the cookie clusters, obtain $\mathcal{X}_{\text{train}}$. Next, we computed the Manhattan, Euclidean, and infinity norm between $\mathbf{a}_i$ and $\mathbf{a}_{i'}$, $(i, i') \in \mathcal{X}_{\text{train}}$ that were used as feature vector to train a logistic regression model. Although also other classifiers could be used, we considered that using a logistic regression model on a compressed input (the three distance measures) would be a proper trade-off between model complexity and accuracy.

We selected for each point the $K = 1,000$ nearest neighbors to evaluate classifier $\hat{f}$ on. All non-evaluated pairs received distance $\delta_{\max} = -\log\left(10^{-6}\right)$. Next, the MS-(H)DBSCAN algorithms were evaluated using the new distance matrix W, where we experimented again with $k \in \{1, 3, 5\}$, and

$$\epsilon \in \left\{ q_{\min} + \frac{\ell (q_{\max} - q_{\min})}{N_{\text{re-eval}}} \middle| \ell \in \{1, \ldots, N_{\text{re-eval}}\} \right\}, \qquad (15)$$

where $N_{\text{re-eval}} = 5$.

All algorithms excluding HDBSCAN* (i.e., including DBSCAN*) were trained using the `sklearn` package in Python [32] (version 0.22.1). `sklearn` was also used to compute error scores (see Section III-C3). We used the `hdbscan` package [33] (version 0.8.26) to obtain the dendogram and condensed tree, based on which we could impose the maximum cluster size in the way described in Section III-B2. For both packages, the default parameters were used unless indicated otherwise.

3) Error metrics: We considered error metrics from two perspectives. First, we consider error measures with respect to overall website performance. More precisely, given some final clustering $\{\mathcal{B}_1^{\text{final}}, \ldots, \mathcal{B}_m^{\text{final}}\}$, the following error measures are computed. 1) We compute the APE (absolute percentage error) between the real and estimated number of unique users (the latter being equal to m), 2) the Kullback-Leibler divergence (KL-divergence) between the real and estimated user session count distribution (the latter being equal to the cluster size distribution), and 3) the KL-divergence between the real and estimated user conversion distribution. Here, user conversion is defined as the fraction of items clicked per user over all shown (but not necessarily evaluated) items.

The second perspective is on the level of the clusters themselves, where we consider two error measures. To determine the quality of the clusters, we computed the adjusted Rand index (ARI) [34] between computed and real session clusters. Besides ARI, we also measure how well the model distinguishes whether each new session originates from an existing or already observed user, which is measured using the accuracy score.

Since ARI measures the overlap between the computed and real session clusters, we consider ARI to be our main error

measure, using the other error measures to study possible side-effects when optimizing for ARI.

IV. Results

A. Results on base simulation case

Table II shows how the different models perform in terms of several error measures on both the training and test set. For each method, the shown results are the best results obtained under the different hyper parameters tried for that method under that dataset. I.e., in theory the hyper parameters might be slightly different between training and test, though in practice we found this was rarely the case.

The *OBS* model in the table are the scores one would obtain if the observed cookies would be used as clusters. Models using the classifier as distance measure are indicated using subscript p. What immediately becomes apparent is that compared to these observed cookie clusters, all methods perform considerably worse. Hence, in the scenario we consider: a single query where the true location $\left(\eta_1^{(u)}, \eta_2^{(u)}\right)$ is only revealed by clicked and skipped item locations, our approaches do not come near what one would obtain if one would simply take the observed cookies.

However, the scores do reveal some interesting patterns. First, approaches using a probabilistic distance measure seem to overfit the data: they perform relatively well (compared to the other approaches) on various measures on the training set, but on the test set these results are mitigated. Here, MS-DBSCAN* seems to work best when considering multiple error measures. Looking at the results from different hyper-parameter settings for MS-DBSCAN* (Table III), we observe that selecting $k = 1$ performed best. Furthermore, due to our maximum size constraint the clusters did not alter for $\ell \geq 4$ ($\epsilon \geq 6.33$).

Furthermore, methods without a probabilistic distance measure do outperform the DBSCAN*-RAND method on most measures. I.e., they perform better at picking sessions originating from the same user from a given cluster $\mathcal{B}_j$ produced by DBSCAN*, than if we would pick session pairs at random. Although it is difficult to draw a firm conclusion, these findings might be an indication that the same user signal we try to infer from the click data is somewhat weak: if our methods would not pick up a signal at all, we would expect them to have the same result as the DBSCAN*-RAND method.

B. Results on multiple simulation scenarios

In order to judge the sensitivity of our findings on the parameter settings of the simulation model, we permuted the simulation settings to see if this would alter our results. In particular, we considered user distance sensitivity $q \in \{1, 2, 5, 10, 25, 50\}$ (denoted by USER_DIST_SENS_[q]), number of items $V \in \{10, 100\}$ (denoted by ITEM_COUNT_[V]), lifetime phases $\rho \in \{.15, .29, .43, .5, .57, .71, .85\}$ (denoted by LIFETIME_PHASES_[ρ]), and salience $(\phi, \phi') \in \{1, 2, 5, 10\}^2$ (denoted by SALIENCE_[ϕ]_[ϕ']). Whenever one parameter was permuted, the rest of the parameters was left at its value in the base case.

As re-running all models on all simulation settings would be computationally rather expensive, we only re-evaluated the best performing models on the simulation cases. Since in our base case we found that the parameters $k = 1$, $\epsilon = \left(q_{\max}\left(q_{\min}/q_{\max}\right)^{2/3}\right)$ worked reasonably well, these parameters were used for MS-DBSCAN* and DBSCAN*-RAND. The maximum cluster size remained the same as in the base case.

Figure 2 shows how the models perform over the different simulation settings in terms of ARI, which is our main response variable of interest. The figure suggests that all cluster models do stochastically dominate DBSCAN*-RAND. Furthermore, MS-DBSCAN* seems to outperform the other clustering methods in terms of ARI. As assumptions like homogeneity of variance or normality do not hold in this case, we used a Kruskall-Wallis test, which rejects in this case that all median ARI scores over the different methods are the same (using significance level $\alpha = .01$, $p < 10^{-4}$). Pairwise (between MS-DBSCAN* and all other methods) one-sided pairwise Wilcoxon signed rank tests also indicate MS-DBSCAN* performed significantly better than the other methods (all p-values are smaller than 10^{-4}).

Table IV shows how MS-DBSCAN* performs on the various simulation cases. The rows in boldface have ARI ≥ 0.0025. The results suggest that when strengthening the signal, that is increasing click probabilities, leads to some improvement in ARI. The most obvious way to do so is by decreasing the number of items (which, as we use bin counting, ensures each item has sufficient data for bin counting). However, these improvements remain small.

Table V shows how the different error measures correlate, using the error scores from all clustering algorithms on the various simulation cases. ARI seems to be weakly correlated with most other error measures, with the sign being in the desired direction (i.e., decrease in KL-divergence for both session count and conversion, but an increase in the new user accuracy). However, both ARI and the new user accuracy show a positive correlation with the percentage error in the number of unique users.

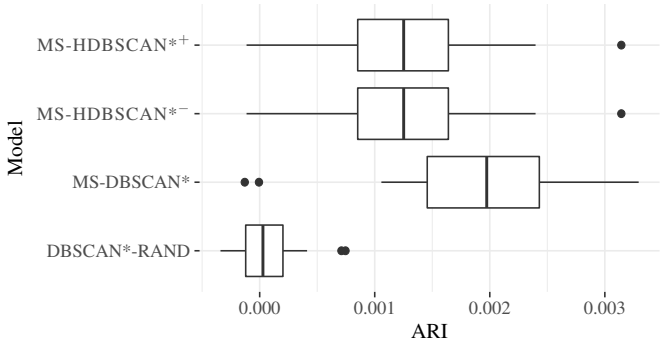


Figure 2. Scores over all simulations.

V. Conclusion and Discussion

In this paper, we presented a homogeneous query click simulation model, and illustrated its usage to the problem of uncovering users from their web sessions. The simulation model is composed of several models from which previous literature suggests that these models work well in explaining

TABLE II
RESULTS ON THE BASE CASE.

Model	Dataset	ARI	KL-div. session count	KL-div conversion	APE unique user	New user accuracy
MS-DBSCAN*	train	0.0012	**0.55**	0.13	15	**0.56**
MS-DBSCAN*$_p$	train	**0.14**	0.74	**0.092**	77	0.5
DBSCAN*-RAND	train	0.0002	1	0.096	**0.011**	0.42
MS-HDBSCAN*$^+$	train	0.0007	0.75	0.15	10	0.52
MS-HDBSCAN*$^-$	train	0.0007	0.75	0.15	10	0.52
MS-HDBSCAN*^+_p	train	0.092	0.9	0.11	**0.011**	0.46
MS-HDBSCAN*^-_p	train	0.1	0.9	0.11	**0.011**	0.46
OBS	*train*	*0.91*	*0.017*	*0.0032*	*15*	*0.95*
MS-DBSCAN*	test	**0.0022**	**0.11**	**0.0026**	60	**0.56**
MS-DBSCAN*$_p$	test	0.0015	1.4	0.13	**6.8**	0.4
DBSCAN*-RAND	test	0.0004	0.32	0.015	40	0.5
MS-HDBSCAN*$^+$	test	0.002	0.16	0.0042	53	0.55
MS-HDBSCAN*$^-$	test	0.002	0.16	0.0042	53	0.55
MS-HDBSCAN*^+_p	test	0.0015	1.4	0.13	7.2	0.4
MS-HDBSCAN*^-_p	test	0.0015	1.4	0.13	7.2	0.4
OBS	*test*	*0.91*	*0.1*	*0.0076*	*51*	*0.95*

TABLE III
ARI OF MS-DBSCAN* ON THE TRAINING SET OF THE BASE CASE.

		k		
ℓ	ϵ	1	3	5
1	0.013	$< 10^{-4}$	$< 10^{-4}$	$< 10^{-4}$
2	3.44	0.0008	0.0004	0.0001
3	4.84	0.0011	0.0005	0.0004
4	6.33	0.0013	0.0006	0.0004
5	8.20	0.0013	0.0006	0.0004
6	10.76	0.0013	0.0006	0.0004
7	14.57	0.0013	0.0006	0.0004
8	20.94	0.0013	0.0006	0.0004
9	30.45	0.0013	0.0006	0.0004

typical patterns observed in click data, while remaining relatively simple. Such patterns include the position bias, cookie censoring, and users' utility over multiple products. Furthermore, we illustrated the simulation model on the problem of (partially observed) session clustering, that is, identifying unique users from their query-sessions. To solve the latter problem, we tested several mutations of (H)DBSCAN*, where these mutations differ from HDBSCAN*, or DBSCAN*, as they allow for incorporating a maximum cluster size. Furthermore, we consider both a Euclidean and probabilistic distance measure to determine whether a pair of sessions originated from the same user. The probabilistic distance measure was obtained using a pre-trained classification model.

Given a simulated dataset, we considered solving the problem of uncovering users from their web sessions by using (H)DSCAN*-type clustering algorithms. Comparing (H)DSCAN*-type algorithms with clusters one would obtain by using cookies, we found the accuracy of using cookies largely outperformed that of not using or partially using cookie data. This considerable difference seems to be due to two reasons. 1) The simulated censored cookies turned out to be rather accurate, implying that, assuming the parameters used for cookie censoring adapted from previous literature are accurate, censoring in cookie data does not impose that much of a problem in accurately measuring the metrics studied in this paper. These metrics being the number of unique users, user sessions count distribution, user conversion distribution, the quality of session clusters (in terms of adjusted Rand index (ARI)), and estimating whether the next session originates from a new or existing user. 2) As we only consider a homogeneous query, the users' preferences are only revealed from the items users clicked, a signal the various (H)DBSCAN*-type algorithms find difficult to detect. Strengthening this signal, e.g., by increasing the number of clicks, leads to a small but significant improvement in ARI.

Other interesting observations include the difference between using Euclidean distance and a probability distance measure in the (H)DBSCAN*-type algorithms, the latter being obtained from training a classifier on detecting whether session pairs originate from the same user. The results show that the probabilistic classifier tends to overfit. Where some methods using probabilistic distance measures performed reasonable on the training set, they were outperformed by methods using Euclidean distance on the test set.

By studying the correlations between the various error metrics considered in this paper, we observe that some error measures show contradictory correlations. In particular, the positive correlation between cluster ARI and average percentage error in the number of unique users (.38), and between the accuracy in estimating whether the next session originates from a new user and the new user average percentage error (.95), indicate that optimizing for one of these error measures may lead to decreased performance in the other.

Although our findings suggest that the practicality of session clustering from single query click data is limited, the usage of the simulation model did allow for studying the sensitivity of the clustering algorithms on different click behavior, something that would not easily have been possible with real click data. It also allowed us to study the effects of user censoring caused by cookie churn or the usage of multiple devices. This showed that if we adopt models for cookie churn behavior found in the literature, this censoring only has a small effect on the accuracy of the website metrics discussed in this paper, with an exception for estimating the number of unique users.

Given our findings, a number of questions remain. First, it would be interesting to extend the simulation model to allow for multiple queries. As the solutions to the (multi-query)

TABLE IV
RESULTS MS-DBSCAN* ON OTHER SIMULATION CASES.

Simulation case	ARI	KL-div. conversion	KL-div. session count	APE unique user	New user accuracy
base_case	0.0021	0.0044	0.095	62	0.53
item_count_10	**0.0025**	**0.0006**	**0.049**	**67**	**0.57**
item_count_100	0.0015	0.0053	0.098	59	0.54
lifetime_phases_.15	0.0014	0.014	0.14	56	0.56
lifetime_phases_.29	0.0016	0.0076	0.1	59	0.55
lifetime_phases_.43	0.0021	0.008	0.11	58	0.56
lifetime_phases_.5	0.0021	0.0044	0.095	62	0.53
lifetime_phases_.57	0.0019	0.0049	0.11	61	0.55
lifetime_phases_.71	0.0019	0.0054	0.084	60	0.56
lifetime_phases_.85	**0.0028**	**0.005**	**0.092**	**61**	**0.53**
salience_1_1	0.0012	0.0018	0.07	64	0.58
salience_1_2	0.0022	0.0019	0.059	65	0.57
salience_1_5	0.0014	0.0015	0.085	62	0.59
salience_1_10	**0.0026**	$\mathbf{< 10^{-4}}$	**0.084**	**62**	**0.58**
salience_2_1	0.0011	0.0026	0.15	53	0.55
salience_2_2	0.002	0.0033	0.19	51	0.55
salience_2_5	**0.0027**	**0.0005**	**0.12**	**56**	**0.57**
salience_2_10	0.0018	$< 10^{-4}$	0.094	60	0.58
salience_5_1	$< 10^{-4}$	0.011	0.25	44	0.52
salience_5_2	$< 10^{-4}$	0.021	0.2	51	0.53
salience_5_5	0.0021	0.0044	0.095	62	0.53
salience_5_10	0.002	$< 10^{-4}$	0.079	62	0.57
salience_10_1	0.0016	0.0049	0.051	64	0.57
salience_10_2	0.0014	0.0034	0.065	66	0.57
salience_10_5	0.0018	0.0045	0.1	60	0.56
salience_10_10	0.0012	0.0001	0.042	71	0.63
user_dist_sense_1	0.0021	0.0044	0.095	62	0.53
user_dist_sens_2	**0.0029**	**0.012**	**0.17**	**49**	**0.54**
user_dist_sens_5	**0.0033**	**0.0092**	**0.15**	**49**	**0.53**
user_dist_sens_10	**0.0026**	**0.002**	**0.12**	**57**	**0.56**
user_dist_sens_25	0.0024	$< 10^{-4}$	0.13	59	0.57
user_dist_sens_50	**0.0032**	**0.0002**	**0.09**	**64**	**0.56**

TABLE V
CORRELATION MATRIX ERROR MEASURES.

	ARI	KL-div. conversion	KL-div session count	APE unique user	New user accuracy
ARI	1.00				
KL-div. conversion	-0.15	1.00			
KL-div. session count	-0.16	0.60	1.00		
APE unique user	**0.38**	-0.52	-0.92	1.00	
New user accuracy	0.39	-0.59	-0.85	**0.95**	1.00

CIKM 2016 and ICDM 2015 cross-device matching competitions were quite successful, a logical hypothesis would be that incorporating multiple queries into the simulation model would improve the results obtained from (H)DBSCAN*-type algorithms. On the other hand, more diversity also causes clicks to be more spread across items that may lead to decreasing clustering performance.

Second, in this study, we only used a logistic regression model to approximate the probability of two sessions originating from the same user. Given the limited success of this approach so far, it would be interesting to consider other approaches. As the limited results seem to be due to overfitting, including regularization or using bagging could lead to better results.

Third, there is still limited knowledge on how cookie censoring occurs. Currently, multiple models exists in the literature, but most models only consider a specific type of censoring (e.g., only censoring by cross-device usage or cookie churn), from which one cannot infer how these different types of censoring interact. Also, as discussed in Section III-A2, literature providing parametric models for cookie churn, user lifetime and absence time (the time between two sessions) seems to be contradictory in terms of tail probabilities. Hence, click simulation models that incorporate cookie censoring would benefit from studies taking a more holistic view on cookie censoring.

REFERENCES

[1] C. de Ruijt and S. Bhulai, "Detecting users from website sessions: A simulation study," in *DATA ANALYTICS 2020, The Ninth International Conference on Data Analytics*, 2020, pp. 35–40.

[2] A. Dasgupta, M. Gurevich, L. Zhang, B. Tseng, and A. O. Thomas, "Overcoming browser cookie churn with clustering," in *Proceedings of the fifth ACM international*

conference on Web search and data mining. ACM, 2012, pp. 83–92.

[3] ICDM, *ICDM 2015: Drawbridge Cross-Device Connections*, 2015, retrieved from: https://www.kaggle.com/c/icdm-2015-drawbridge-cross-device-connections, accessed November 17, 2021.

[4] CIKM, *CIKM Cup 2016 Track 1: Cross-Device Entity Linking Challenge*, 2016, retrieved from: https://competitions.codalab.org/competitions/11171, accessed November 17, 2021.

[5] S. Kim, N. Kini, J. Pujara, E. Koh, and L. Getoor, "Probabilistic visitor stitching on cross-device web logs," in *Proceedings of the 26th International Conference on World Wide Web*. ACM, 2017, pp. 1581–1589.

[6] D. Jin, M. Heimann, R. Rossi, and D. Koutra, "node2bits: Compact time-and attribute-aware node representations," in *ECML/PKDD European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2019.

[7] A. Chuklin, I. Markov, and M. d. Rijke, *Click models for web search*. Morgan & Claypool Publishers, 2015.

[8] D. Fleder and K. Hosanagar, "Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity," *Management science*, vol. 55, no. 5, pp. 697–712, 2009.

[9] D. Coey and M. Bailey, "People and cookies: Imperfect treatment assignment in online experiments," in *Proceedings of the 25th International Conference on World Wide Web*. ACM, 2016, pp. 1103–1111.

[10] G. Dupret and M. Lalmas, "Absence time and user engagement: evaluating ranking functions," in *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 2013, pp. 173–182.

[11] G. D. Montanez, R. W. White, and X. Huang, "Cross-device search," in *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 2014, pp. 1669–1678.

[12] F. M. Naini, J. Unnikrishnan, P. Thiran, and M. Vetterli, "Where you are is who you are: User identification by matching statistics," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 2, pp. 358–372, 2016.

[13] R. Saha Roy, R. Sinha, N. Chhaya, and S. Saini, "Probabilistic deduplication of anonymous web traffic," in *Proceedings of the 24th International Conference on World Wide Web*. ACM, 2015, pp. 103–104.

[14] Q. Wang, "Recombining customer journeys with probabilistic cookie matching: A supervised learning approach," in *Machine learning applications in operations management and digital marketing*, 2019, ch. 6, pp. 127–139.

[15] J. Lian and X. Xie, "Cross-device user matching based on massive browse logs: The runner-up solution for the 2016 CIKM cup," *arXiv preprint arXiv:1610.03928*, 2016.

[16] N. K. Tran, "Classification and learning-to-rank approaches for cross-device matching at CIKM cup 2016," *arXiv preprint arXiv:1612.07117*, 2016.

[17] R. Díaz-Morales, "Cross-device tracking: Matching devices and cookies," in *2015 IEEE International Conference on Data Mining Workshop (ICDMW)*, 2015, pp. 1699–1704.

[18] M. C. Phan, A. Sun, and Y. Tay, "Cross-device user linking: URL, session, visiting time, and device-log embedding," in *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2017, pp. 933–936.

[19] M. C. Phan, Y. Tay, and T.-A. N. Pham, "Cross device matching for online advertising with neural feature ensembles: First place solution at CIKM cup 2016," 2016.

[20] R. Song, S. Chen, B. Deng, and L. Li, "eXtreme gradient boosting for identifying individual users across different digital devices," in *International Conference on Web-Age Information Management*. Springer, 2016, pp. 43–54.

[21] U. Tanielian, A.-M. Tousch, and F. Vasile, "Siamese cookie embedding networks for cross-device user matching," in *Companion Proceedings of the The Web Conference 2018*. ACM, 2018, pp. 85–86.

[22] C. Karakaya, H. Toğuç, R. S. Kuzu, and A. H. Büyüklü, "Survey of cross device matching approaches with a case study on a novel database," in *2018 3rd International Conference on Computer Science and Engineering (UBMK)*, 2018, pp. 139–144.

[23] L. Olejnik, C. Castelluccia, and A. Janc, "On the uniqueness of web browsing history patterns," *annals of telecommunications-annales des télécommunications*, vol. 69, no. 1-2, pp. 63–74, 2014.

[24] L. Jalali, M. Khan, and R. Biswas, "Learning and multi-objective optimization for automatic identity linkage," in *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 2018, pp. 4926–4931.

[25] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[26] R. J. Campello, D. Moulavi, and J. Sander, "Density-based clustering based on hierarchical density estimates," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2013, pp. 160–172.

[27] R. J. Campello, D. Moulavi, A. Zimek, and J. Sander, "Hierarchical density estimates for data clustering, visualization, and outlier detection," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 10, no. 1, pp. 1–51, 2015.

[28] C. Luna-Nevarez and M. R. Hyman, "Common practices in destination website design," *Journal of destination marketing & management*, vol. 1, no. 1-2, pp. 94–106, 2012.

[29] O. Chapelle and Y. Zhang, "A dynamic bayesian network click model for web search ranking," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 1–10.

[30] L. McInnes and J. Healy, "Accelerated hierarchical density clustering," *arXiv preprint arXiv:1705.07321*, 2017.

[31] I.-K. Yeo and R. A. Johnson, "A new family of power transformations to improve normality or symmetry," *Biometrika*, vol. 87, no. 4, pp. 954–959, 2000.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] L. McInnes, J. Healy, and S. Astels, "hdbscan: Hierarchical density based clustering," *Journal of Open Source*

Software, vol. 2, no. 11, p. 205, 2017.

[34] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

APPENDIX

TABLE VI
LIST OF NOTATION.

Variable	Description
$\{1,\dots,n\}$	Set of query-sessions, indexed by i
$\mathcal{S}_i = \{1,\dots,T\}$	Set of items in SERP of session i, indexed by t
$\mathcal{V} = \{1,\dots,V\}$	Set of all items, indexed by v
$\mathcal{U} = \{1,\dots,U\}$	Set of all users, indexed by u
$r_i(t)$	Item $v \in \mathcal{V}$ at position t in the SERP of query-session i
$r_i^{-1}(v)$	Position of item v in the SERP of query-session i, zero if $v \notin \mathcal{S}_i$
$r_i^{\max}$	Largest position of a clicked item $v \in \mathcal{S}_i$, zero if no items were clicked
$R_v^{(i)}$; $\phi_{u,v}^{(R)}$	Attraction of user i for item v, with $\mathbb{P}(R_v^{(i)} = 1) = \phi_{u,v}^{(R)}$ given $v \in \mathcal{S}_i$
$S_v^{(i)}$; $\phi_{u,v}^{(S)}$	Satisfaction of user i for item v, with $\mathbb{P}(S_v^{(i)} = 1) = \phi_{u,v}^{(S)}$ given $v \in \mathcal{S}_i$
$E_t^{(i)}$	Whether item at position t in SERP i was evaluated
$y_t^{(i)}$	Whether item at position t in SERP i was clicked
$\boldsymbol{\eta}_u$	Vector denoting the position of an user u in the user-item space
$\boldsymbol{\psi}_v$	Vector denoting the position of an item v in the user-item space
$\nu^{(A)}, \nu^{(S)}$	Salience parameters for attraction and satisfaction
q	Users' preference for nearby items
$\omega_{u,v}$	Distance between user u and item v in the user-item space
$\hat{\phi}_v$	Overall estimated popularity of item $v \in \mathcal{V}$
F^{cookie}	Cookie lifetime distribution (hyper-exponential) with parameters $\boldsymbol{\lambda}$ and $\mathbf{p}$
$\mathcal{T}_{u,o,d}^{\text{cookie}} \sim F^{\text{cookie}}$	R.v. denoting the cookie lifetime for the o-th cookie of user u on device d
F^{abs}	User absence distribution (Pareto-I) with parameters α (shape) and m (scale)
$\mathcal{T}_{i,u}^{\text{abs}} \sim F^{\text{abs}}$	R.v. denoting the time between the i-th and $i+1$-th session of user u
F^{user}	User lifetime distribution (sum of N_u hyper-exponentials) with parameters $\boldsymbol{\lambda}$, $\mathbf{p}$ and ρ (geometric parameter for N_u)
$\mathcal{T}_u^{\text{user}} \sim F^{\text{user}}$	R.v. denoting the user lifetime of user u
P, $\boldsymbol{\pi}$	Device transition matrix and initial device probabilities

Algorithm 2: SIMULATE_CLICKS

Simulate_clicks $(\mathcal{U})$
  for $u \in \mathcal{U}$ **do**
    /* Draw initial device and cookie lifetime, and draw the user's lifetime */
    $D \leftarrow$ dic(); $i \leftarrow 1$; $o \leftarrow 1$; $t \leftarrow 0$;
    Draw device d from MULTINOM($\boldsymbol{\pi}$); $D[d] \leftarrow o$;
    Draw $\mathcal{T}_{u,o,d}^{\text{cookie}}$ from HYPEREXP($\boldsymbol{\lambda}, \boldsymbol{p}$); $\mathcal{T}_u^{\text{user}}$ from REPHYPEREXP($\rho, \boldsymbol{\lambda}, \boldsymbol{p}$);
    /* Simulate new query-sessions while the user's lifetime has not elapsed */
    while $t \leq \mathcal{T}_u^{user}$ **do**
      /* 1) Simulate clicks */
      Draw $\mathcal{S}_i$ in its respective order by repetitively drawing from MULTINOM($\hat{\phi}_v / \sum_{v' \in \mathcal{V}} \hat{\phi}_{v'}; v \in \mathcal{V} \setminus \mathcal{S}_i$);
      Draw $R_v^{(i)}$, $S_v^{(i)}$ from BERNOULLI($\phi_{u,v}^{(R)}$) and BERNOULLI($\phi_{u,v}^{(S)}$) resp. for all $v \in \mathcal{S}_i$;
      Compute $E_t^{(i)}, y_t^{(i)}$, and recompute $S_v^{(i)}$ according to Equations (1) to (6);
      Append $(i, u, o, \mathcal{S}_i, \mathbf{y}_i)$ to $\mathcal{D}$;
      /* 2) Draw the time until the next session and update t accordingly */
      Draw $\mathcal{T}_{i,u}^{\text{abs}}$ from PARETO-I(m, α);
      $t \leftarrow t + \mathcal{T}_{i,u}^{\text{abs}}$; $i \leftarrow i + 1$;
      /* 3) Update the device for the next session */
      Draw d' from MULTINOM($I_d P$);
      if $d' \neq d$ **then**
        if ***not*** $D.exists(d)$ **then**
          $o \leftarrow o + 1$;
          Draw $\mathcal{T}_{u,o,d}^{\text{cookie}}$ from HYPEREXP($\boldsymbol{\lambda}, \boldsymbol{p}$);
          $\mathcal{T}_{u,o,d}^{\text{cookie}} \leftarrow \mathcal{T}_{u,o,d}^{\text{cookie}} + t$;
        else
          $o \leftarrow D[d']$;
        $d \leftarrow d'$;
      /* 4) Simulate cookie churn */
      if $t > \mathcal{T}_{u,o,d}^{cookie}$ **then**
        $o \leftarrow o + 1$;
        Draw $\mathcal{T}_{u,o,d}^{\text{cookie}}$ from HYPEREXP($\boldsymbol{\lambda}, \boldsymbol{p}$);
        $\mathcal{T}_{u,o,d}^{\text{cookie}} \leftarrow \mathcal{T}_{u,o,d}^{\text{cookie}} + t$;
        $D[d] \leftarrow o$;
  return $\mathcal{D}$