# SoCRATe: A Recommendation System with Limited-Availability Items


Davide Azzalini*
Politecnico di Milano
Milan, Italy
davide.azzalini@polimi.it

Fabio Azzalini*
Politecnico di Milano
Milan, Italy
fabio.azzalini@polimi.it

Chiara Criscuolo*
Politecnico di Milano
Milan, Italy
chiara.criscuolo@polimi.it

Tommaso Dolci*
Politecnico di Milano
Milan, Italy
tommaso.dolci@polimi.it

Davide Martinenghi
Politecnico di Milano
Milan, Italy
davide.martinenghi@polimi.it

Sihem Amer-Yahia
CNRS, Université Grenoble Alpes
Grenoble, France
sihem.amer-yahia@cnrs.fr



## ABSTRACT

We demonstrate SoCRATe, an online system dedicated to providing adaptive recommendations to users when items have limited availability. SoCRATe is relevant to several real-world applications, among which movie and task recommendations. SoCRATe has several appealing features: *(i)* watching users as they consume recommendations and accounting for user feedback in refining recommendations in the next round; *(ii)* implementing loss compensation strategies to make up for sub-optimal recommendations, in terms of accuracy, when items have limited availability; *(iii)* deciding when to re-generate recommendations on a need-based fashion. SoCRATe accommodates real users as well as simulated users to enable testing multiple recommendation choice models. To frame evaluation, SoCRATe introduces a new set of measures that capture recommendation accuracy, user satisfaction and item consumption over time. All these features make SoCRATe unique and able to adapt recommendations to user preferences in a resource-limited setting. A video of SoCRATe is available at https://youtu.be/4wlaScc_rUo.



## CCS CONCEPTS

• **Information systems** → **Content ranking**; **Social recommendation**; **Recommender systems**.

## KEYWORDS

Recommender Systems, Dynamic User Preferences, Compensation Strategies

**ACM Reference Format:**
Davide Azzalini, Fabio Azzalini, Chiara Criscuolo, Tommaso Dolci, Davide Martinenghi, and Sihem Amer-Yahia. 2022. SoCRATe: A Recommendation System with Limited-Availability Items. In *Proceedings of the 31st ACM International Conference on Information and Knowledge Management (CIKM '22), October 17–21, 2022, Atlanta, GA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3511808.3557208


*These authors contributed equally to this research.



*CIKM '22, October 17–21, 2022, Atlanta, GA, USA*



ACM ISBN 978-1-4503-9236-5/22/10…$15.00
https://doi.org/10.1145/3511808.3557208


## 1 INTRODUCTION

The goal of a recommendation system is to produce a set of items that are highly relevant to a user. A typical recommendation workflow takes historical user data, applies a strategy that produces relevant recommendations, and measures accuracy [4–6, 8–12]. The decision of refining recommendations is usually made offline based on how well a given strategy performs, and based on the users' explicit and implicit feedback. In this paper, we advocate for an iterative approach that benefits from observing users as they consume proposed items, thus creating a feedback loop to adapt recommendations to users on the fly in the next iteration. This is particularly important when items have limited availability, such as product recommendation, where only a fixed number of copies of an item is available, and in crowdsourcing, where tasks are to be completed by a fixed number of workers. Our work relates to both sequential [14] and session-based [13] recommendations.

We demonstrate SoCRATe,[1] a system dedicated to providing adaptive recommendations to users. The limited availability of items incurs competition between users for the same items, which in turn results in users receiving sub-optimal recommendations. SoCRATe introduces the notion of item compensation in subsequent iterations, which aims to treat users equally.

### 1.1 Challenges

The first challenge **C1** is to model user behavior, i.e., how users consume recommendations, and capture both explicit and implicit feedback. Explicit feedback represents the actions taken by the users when provided with recommendations. Examples of actions include rating movies, and completing a task. When users are simulated, different assumptions can be made to reflect how they choose to adopt their recommendations. When users are real, their explicit choice of items out of recommended ones needs to be captured. Additionally, several implicit signals can be captured, including the order and speed at which users consume recommendations. The second challenge **C2** is to leverage the observed feedback to refine recommendations in the next iteration. This challenge is coupled with the need to be adaptive regardless of the recommendation

[1] System for Compensating Recommendations with Availability and Time

strategy. It also raises the question of the order in which users are compensated for the loss incurred from competing on limited-availability items. Indeed, when the number of copies of an item is limited (referred to as *availability*), some users may not get an item even if it is in their top choices. This may happen in book recommendation or in crowdsourcing, where the number of workers who complete a task is limited. Appropriate loss compensation strategies must hence be devised in subsequent steps. The third challenge **C3** is to determine at what moment and for which users recommendations are re-optimized based on their feedback. Indeed, users consume recommendations at different speeds resulting in different priority groups.

## 1.2 Contributions

SoCRATe watches recommendation consumption and decides to apply a loss compensation strategy to make up for sub-optimal recommendations due to limited item availability.

To address **C1**, we introduce an objective function in the form of a linear combination of several dimensions that capture users' preferences when choosing recommendations. At each iteration, SoCRATe assumes that users choose $k$ out of $N$ provided recommendations. For instance, when presented with a list of movie recommendations, users may choose highly relevant items (utility-based) while also favoring affordable items (cost-based). In the case of crowdsourcing, users may pick tasks that are relevant to their profile (utility-based) while also looking for tasks posted by different requesters (diversity) and with high compensation (payment). To capture that, we devise a generic objective function where each dimension receives a weight according to the user's choice. When users are simulated, we implement three choice models [2, 3, 15]: *random*, where users are assumed to choose items randomly; *utility-based*, where users are assumed to choose items in decreasing order of their utility with respect to their profile; and *rank-based*, where users are assumed to choose their items in the order in which they are ranked by the objective function.

To address **C2**, SoCRATe captures each user's feedback as a weight vector, where each entry represents the preference of a user for an optimization dimension. This representation is generic enough to capture a variety of applications. To refine weight vectors, SoCRATe runs a regression that compares the $k$ chosen recommendations against the remaining $N - k$. Refined weight vectors are used to re-rank the recommendations produced for each user to match their feedback. Due to limited item availabilities in some applications, users may compete for some items and not receive their optimal recommendations. Hence, we model the loss for a user as the difference between the proposed recommendations and the optimal recommendations that would have been returned to the user if other users were not considered. This loss is leveraged in the next iteration to compensate users, either individually or in groups according to the temporal granularity. We develop two loss compensation strategies: *preference-driven*, where users are compensated in decreasing loss order by receiving all their recommendations in preference order before the user with the next smaller loss is served, and *round-robin*, where users are considered in decreasing loss order but items are recommended to them in round-robin with respect to item availability.

| | |
|---|---|
| $U$ | set of users |
| $I$ | set of items |
| $R_u^t$ | list of recommended items for user $u$ at time $t$ |
| $N$ | number of recommended items in $R_u^t$ |
| $OptR_u^t$ | optimal list of recommendations for user $u$ at time $t$ |
| $M$ | number of optimal recommendations in $OptR_u^t$ |
| $\hat{OptR}_u^t$ | top $N$ recommendations from $OptR_u^t$ |
| $S_u^t$ | list of adopted items by user $u$ at time $t$, $S_u^t \subseteq R_u^t$ |
| $k_u^t$ | number of adopted items by user $u$ at time $t$ |
| $avail(i,t)$ | number of copies left of item $i$ at time $t$ |
| $SortU^t$ | ranking of $U$ in decreasing loss $loss_u^t$ at time $t$ |
| $W_u^t$ | weight vector for user $u$ at time $t$ based on $S_u^t$ |
| $loss_u^t$ | vector of one loss per dimension for user $u$ at time $t$ |

**Table 1: Symbols of the data model of SoCRATe**

To address **C3**, we develop two temporal granularities: *fixed*, where recommendations of all users are re-optimized at fixed time periods, and *user-group*, where the best moment is determined for a group of users as a function of when they are all ready. In that case, groups will be determined based on similarity in behavior, e.g. how fast users are to consume their recommendations.

# 2 THE SOCRATE SYSTEM

## 2.1 Architecture of SoCRATe

The architecture of SoCRATe is provided in Figure 1. Table 1 summarizes the symbols of the data model. The Orchestrator (1) is in charge of determining the moment at which recommendations are re-generated for individuals or for user groups. It monitors when users are done consuming recommendations from the previous iteration and uses this to determine when to re-generate recommendations and start the next iteration. It implements our two time granularity splits: *fixed* and *user-group*. It makes use of the recommendations adopted by users $S_u^t$ according to the Choice Model in use (2), and triggers the Weight Update module (3) to obtain the new weights. Weight computation leverages $S_u^t$ along with the previous recommendations $R_u^{t-1}$ and their optimal counterpart $\hat{OptR}_u^{t-1}$ to return the updated weights $W_u^t$.

Following that, the Individual Recommender module (4) is called to produce new recommendations based on the weights associated with each user. This module returns $OptR_u^t$, which is fed to the Compensation Strategy module. Here the User Sorting module (5) sorts users according to their loss and calls the Item Recommendation module (6). This module implements two strategies: *preference-driven* and *round-robin*. The output is a set of new recommendations, $R_u^t$, that are fed to the Loss Computation module (7) to compute each user's incurred loss, $loss_u^t$, by comparison with $\hat{OptR}_u^t$, i.e., the top $N$ optimal recommendations from $OptR_u^t$. SoCRATe then iterates by calling the Orchestrator.

## 2.2 UI and Algorithms of SoCRATe

Figure 2 is a screenshot of the UI of SoCRATe. Zone A is the Control Panel, where the user selects the dataset to analyze along with compensation parameters, user choice model, and time granularity. The UI provides the following features:

*Loss Management.* Here the user can see the evolution of different aggregations of the loss incurred by a selected subset of five users of SoCRATe. In particular, Zone B displays the loss at each

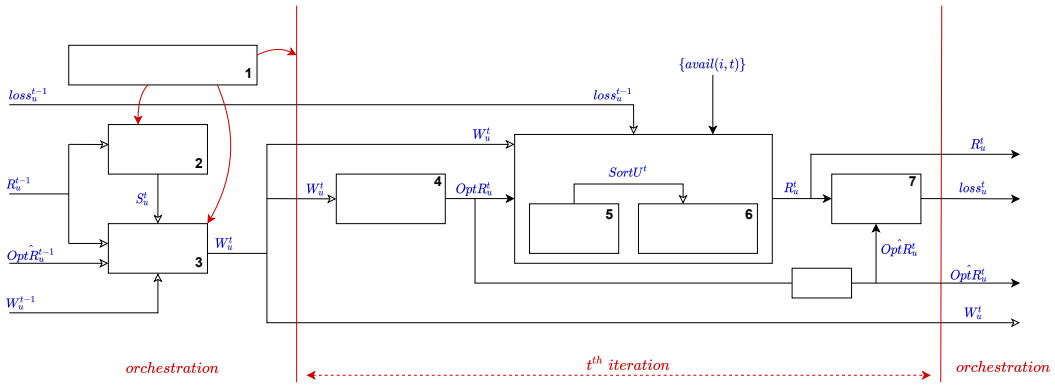


**Figure 1: Architecture of SoCRATe**

iteration for *u_max*, i.e., the user with the highest total loss after the simulation has ended, *u_min*, i.e., the user with the lowest total loss, *u_25%*, *u_50%* (median user) and *u_75%*, i.e., the users with a total loss closest to the $25^{th}$, $50^{th}$ and $75^{th}$ percentiles, respectively. Such information is displayed both for the specific instance of SoCRATe based on the choices made in Zone A (top) and for a baseline where no compensation is applied (bottom). Zone C contains similar information as Zone B, except for the fact that cumulated loss is displayed instead of that incurred at the current iteration. Zone D sums up in a single scalar value the performance of different compensation strategies by means of gauge charts. Specifically, mean and standard deviation of the cumulated losses computed among all users at the last iteration are displayed for both the specific instance of SoCRATe (top) and for a baseline (bottom).

*Item Management.* Zone E is dedicated to the evolution of item consumption in terms of availability and popularity. Item distribution according to four availability clusters is displayed by means of a series of histograms (left), one for each iteration. The distribution of popularity among currently available items is also shown through a percentage stacked area chart (right) according to three popularity groups. Finally, Zone F reports overall statistics.

**Algorithm 1** SoCRATe

**Input:** $\forall u, loss_u^{t-1}, R_u^{t-1}, OptR_u^{t-1}, W_u^{t-1}$
**Output:** $\forall u, loss_u^t, R_u^t, OptR_u^t, W_u^t$

1: **for all** $t \in 1, ..., T$ **do**
2: $S_u^t \leftarrow$ `ChoiceModel`$(R_u^{t-1})$, for all $u \in U$
3: $W_u^t \leftarrow$ `WeightUpdate`$(OptR_u^{t-1}, R_u^{t-1}, W_u^{t-1}, S_u^t)$, for all $u \in U$
4: $OptR_u^t \leftarrow$ `IndividualRecommender`$(W_u^t)$, for all $u \in U$
5: $SortU^t \leftarrow$ `UserSorting` $(\{W_u^t\}, \{loss_u^{t-1}\})$
6: $R_u^t \leftarrow$ `ItemRecommendation` $(SortU^t, OptR_u^t, \{avail(i,t)\})$, for all $u \in U$
7: $loss_u^t \leftarrow$ `LossComputation`$(OptR_u^t, R_u^t)$, for all $u \in U$
8: **end for**

Algorithm 1 shows how SoCRATe operates (further details are provided in [1]). The system is illustrated for *fixed* time, but the same applies to *user-group*. It takes as input $t$, the moment where a new iteration starts according to the Orchestrator, and, for each user $u$, $loss_u^{t-1}$, $R_u^{t-1}$, $OptR_u^{t-1}$, $W_u^{t-1}$. It outputs for each user $loss_u^t$, $R_u^t$, $OptR_u^t$, $W_u^t$. The line numbers in the overall algorithm reflect the module numbers in Figure 1. Lines 5 and 6 correspond to the compensation strategy. Each strategy receives a list of users sorted in decreasing loss, $SortU^t$, the optimal recommendations, in terms of accuracy, for each user, $OptR_u^t$, the availability for each item, $avail(i,t)$, and produces a list of recommended items for each user, $R_u^t$. When *preference-driven* compensation is adopted, $N$ items are recommended to the first user in the sorted list before moving to the next one. The *round-robin* variant recommends one item at a time to each user (according to their order in the list), until $N$ items are recommended to each user.

*Preference-driven* compensation is expected to be more effective at maximizing the satisfaction of the first users in the list. As a result, this strategy should be preferred when the system detects that some users are unsatisfied with the recommendations they get, and thus may be prone to abandon the system, thereby affecting overall user retention. *Round-robin* compensation should perform best when the goal is to consume as many items as possible. Under this strategy, it is very likely that all users get at least a few items they like, which will affect their consumption rate. Figure 3 illustrates the difference between our compensation strategies when item availability is low (e.g., equal to 1). On the left example, users have similar optimal recommendations: in this case, *preference-driven* favors $u_1$, while *round-robin* treats $u_1$ and $u_2$ more equally. On the right example, users do not compete since they have different preferences.

## 3 DEMONSTRATION SCENARIO

We demonstrate SoCRATe over three real-world datasets, namely *Amazon Digital Music*, *Amazon Movies and TV*,[2] *Task Recommendation*. We keep only those users and items with at least 5 reviews each in the Amazon music dataset, and at least 20 reviews in the Amazon movie dataset. In *Task Recommendation* [7], users (workers) complete and rate task sessions. The dataset consists of 200 workers and 20k tasks. Each task belongs to one of 10 types, such as tweet classification, image transcription or sentiment analysis, and has a reward that varies between $0.01 and $0.05. Additionally, we give the opportunity to test the system on a synthetic dataset to control conflict between users and examine its impact on compensation.

Three possible stakeholders may interact with our framework: system administrators may use SoCRATe to monitor the correct

[2] https://jmcauley.ucsd.edu/data/amazon/

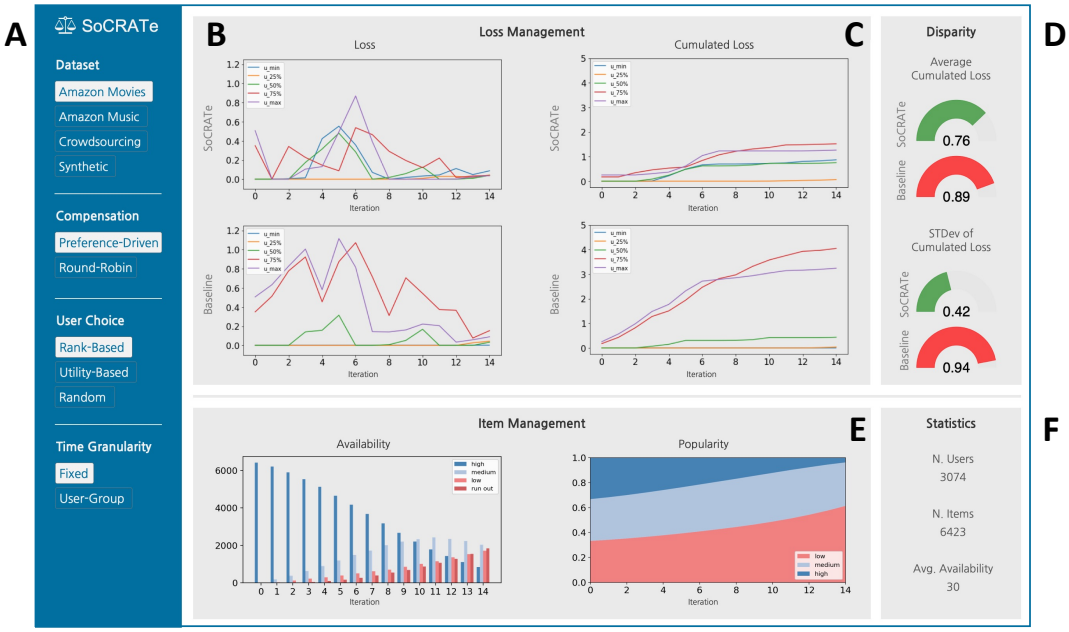


**Figure 2: Screenshot of SoCRATe User Interface, showing real results from a simulation on the *Amazon Movies and TV* dataset.**

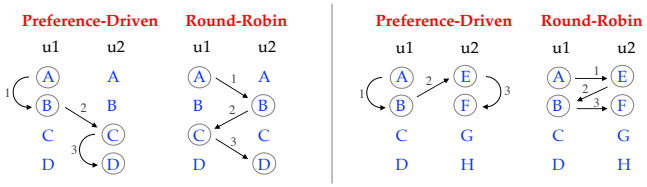


**Figure 3: Illustration of the two compensation strategies. The list of optimal recommendations of each user is shown in decreasing relevance. The numbers on the arrows reflect the order on which items are recommended. On the left, the two users compete as they have similar recommendations. On the right, they do not compete.**

functioning of their system, software developers may use the insight coming from SoCRATe to fine tune the recommender system, finally, end-users will understand if and why the recommendations they received are good or not.

Figure 2 illustrates the use of SoCRATe; let us choose the following parametrization (Zone A): *Amazon Movies* dataset, *preference-driven* compensation, *rank-based* choice model and *fixed* time granularity. The Statistics panel (Zone F) displays general information about the dataset, such as the number of users and items, and the average availability of items. By analyzing the Loss section (Zone B), a system administrator can inspect the loss that five selected users obtain at each iteration and compare it with a baseline where no compensation strategy is applied, therefore users are always considered in the same order by the recommender. The bottom multi-line chart demonstrates how the baseline approach treats the five users very disparately: two of the users (*u_min* and *u_25%*) do not experience any loss, *u_50%* experiences a small amount of loss, while the other two (*u_75%* and *u_max*) experience a significant amount of loss. On the other hand, SoCRATe provides substantially better recommendations, equally distributing the loss among all users. The Cumulated Loss section (Zone C) shows the total loss experienced by our five selected users. From the two charts it is clear how SoCRATe (upper chart) not only treats the users more equally but it also provides lower overall loss; this is confirmed by the first two gauge indicators (Zone D) showing the Average Cumulated Loss of SoCRATe compared to the baseline. Specifically, SoCRATe obtains a value of 0.76, lower than 0.89 obtained by the baseline. The two gauge indicators in the bottom part of Zone D display the standard deviation of the Cumulated Loss, which quantifies the disparity between the users of the system; in fact, SoCRATe scores 0.42 while the baseline obtains 0.94, again confirming a more equal treatment of the users. This is useful for system administrators to improve the equity of her recommender. Also end-users can see whether the recommender is treating them equally.

Finally, Zone E contains two charts displaying on the left the availability in the form of histogram, on the right the popularity shown as a percentage stacked area chart. Here stakeholders can inspect item availability and popularity during the different iterations. From the chart on the right, it is clear that highly popular items are consumed very fast, while medium and low popularity items are consumed in later iterations. The histogram on the left displays, for three groups (low, medium and high availability), how many items remain at each consequent iteration. System administrators and software developers can use this information to manage copies of items, both in the case of physical items (to make logistical decisions), and multimedia content (to balance the server load). Moreover, system administrators can monitor item throughput and whether there exist items that are never recommended and consumed, and modify the recommendation strategy accordingly.

We have demonstrated SoCRATe, an online system able to provide adaptive recommendations to users. SoCRATe models user behavior and implements strategies to compensate the loss incurred from competing on items with limited availability.

# REFERENCES


[1] Davide Azzalini, Fabio Azzalini, Chiara Criscuolo, Tommaso Dolci, Davide Martinenghi, Letizia Tanca, and Sihem Amer-Yahia. 2022. SoCRATe: A Framework for Compensating Users Over Time with Limited Availability Recommendations. In *Proceedings of the 30th Italian Symposium on Advanced Database Systems (SEBD)*.

[2] Allison J. B. Chaney. 2021. Recommendation System Simulations: A Discussion of Two Key Challenges. *CoRR* abs/2109.02475 (2021). arXiv:2109.02475 https://arxiv.org/abs/2109.02475

[3] Naieme Hazrati and Francesco Ricci. 2022. Recommender Systems Effect on the Evolution of Users' Choices Distribution. *Information Processing & Management* 59, 1 (2022), 102766.

[4] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the Eighth IEEE International Conference on Data Mining*. IEEE, 263–272.

[5] Choonho Kim and Juntae Kim. 2003. A Recommendation Algorithm Using Multilevel Association Rules. In *Proceedings IEEE/WIC International Conference on Web Intelligence (WI 2003)*. IEEE, 524–527.

[6] Yehuda Koren. 2008. Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model. In *ACM SIGKDD*. ACM, 426–434.

[7] Sepideh Nikookar, Mohammadreza Esfandiari, Ria Mae Borromeo, Paras Sakharkar, Sihem Amer-Yahia, and Senjuti Basu Roy. 2022. Diversifying recommendations on sequences of sets. *The VLDB Journal* (2022), 1–22.

[8] Michael J Pazzani and Daniel Billsus. 2007. Content-based Recommendation Systems. In *The adaptive web*. Springer, 325–341.

[9] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. AUAI Press, 452–461.

[10] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2015. Recommender Systems: Introduction and Challenges. In *Recommender systems handbook*. Springer, 1–34.

[11] Badrul Sarwar, George Karypis, Joseph Konstan, John Riedl, et al. 2000. Analysis of Recommendation Algorithms for E-commerce. In *EC*. 158–167.

[12] J Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. 2007. Collaborative Filtering Recommender Systems. In *The adaptive web*. Springer, 291–324.

[13] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z Sheng, Mehmet A Orgun, and Defu Lian. 2021. A Survey on Session-based Recommender Systems. *ACM Computing Surveys (CSUR)* 54, 7 (2021), 1–38.

[14] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet A. Orgun. 2019. Sequential Recommender Systems: Challenges, Progress and Prospects. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*. 6332–6338.

[15] Sirui Yao, Yoni Halpern, Nithum Thain, Xuezhi Wang, Kang Lee, Flavien Prost, Ed H. Chi, Jilin Chen, and Alex Beutel. 2021. Measuring Recommender System Effects with Simulated Users. *CoRR* abs/2101.04526 (2021).