Loss Compensation in Multi-Session Recommendation Under Limited Item Availability

Davide Azzalini, Fabio Azzalini, Chiara Criscuolo, Tommaso Dolci, Davide Martinenghi, Sihem Amer-Yahia

Loss Compensation in Multi-Session Recommendation Under Limited Item Availability

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*
Lero, University of Limerick
Limerick, Ireland
tommaso.dolci@lero.ie

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

In many recommendation applications, items may have limited availability thereby causing conflict among users interested in the same items. Over time, this results in unequal user treatment: few users are recommended the limited items and receive preferential treatment, while the rest is left with sub-optimal recommendations, ultimately leading them to leave. In this paper, we formalize the novel problem of compensating users in multi-session recommendations under limited item availability. Our aim is to generate recommendations that not only optimize accuracy, but also compensate users over time for the loss of accuracy incurred in previous iterations. We design compensation strategies that serve users and items in different orders and accommodate various recommendation adoption models. Our algorithms are integrated into SoCRATe (System for Compensating Recommendations with Availability and Time), a framework that enables us to study loss compensation over time. Our experiments on real data demonstrate that to best compensate users for the incurred loss, traditional recommenders need to be revisited to account for item availability. Our experiments on synthetic data explore different parameters of our solution and show that it is much faster than an optimal (brute-force) compensation strategy, while achieving comparable results.

1 INTRODUCTION

A recommender system aims to generate a set of items that are highly relevant to a user. A typical workflow takes historical user data, applies a strategy that produces relevant recommendations, and measures accuracy [10, 12, 13, 19, 22, 23, 25, 26]. In this work, we are interested in a setting where items have limited availability and multiple users are served at each iteration. For instance, when a new book from a popular author comes out, there may not be enough copies for all the interested users. This creates a situation of inequality, where only "privileged" users get access to a limited resource. If the system is not able to compensate the incurred accuracy loss over time, inequality will increase leading unsatisfied users to leave. In this paper, we advocate for an approach that observes users as they consume items, and reduces their loss in accuracy in subsequent iterations. To the best of our knowledge, our work is the first to formalize and tackle recommendations under limited item availability, and compensating users by taking into consideration the order in which users are served, the incurred loss in accuracy, and the recommendation adoption model.

Formalizing loss and compensation in multi-session recommendations raises several challenges. The first challenge **C1** is to model how users adopt recommendations, i.e., their explicit choice of which items to consume out of recommended items. Additionally, several implicit signals must be accounted for, including the order and speed at which users consume recommendations. The second challenge **C2** is to determine at what moment and for which users recommendations are generated, whether at fixed time intervals or based on the users' consumption rate. Under limited availability, some users may not get an item even if it is in their top choice. This yields a loss in accuracy. Hence, the third challenge **C3** is to leverage the observed feedback and the quality of items served to each user to refine recommendations in subsequent iterations in such a way that users are "compensated" for their "loss in accuracy" in previous iterations. This was recently discussed in our vision paper [2] and a prototype demonstration [1].

To address **C1**, we simulate users and recommendation adoption according to existing models [8, 9, 30]. In multi-session recommendation, a user chooses k out of N provided recommendations according to some recommendation adoption model [3, 9, 36]. We recognize three different such models: *random*, where users choose items arbitrarily; *utility-based*, where users choose items in decreasing order of utility, e.g., in crowdsourcing, tasks are chosen based on their fitness to the user profile; *rank-based*, where users follow a ranking of recommended items. Adoption models capture a user's strategic behavior when a *scoring function*, expressed as a linear combination of several dimensions, is used to rank items. In crowdsourcing, ranking dimensions usually include monetary compensation and task requesters.

To address **C2**, we introduce two time granularities: *fixed*, where recommendations of all users are generated at fixed time periods, and *user-group*, where the best moment is determined for a group of users as a function of their behavior, i.e., grouping users with similar consumption rates and producing new recommendations for them at the same time.

To address **C3**, we propose to learn a user's choice of recommended items as a vector, where an entry represents the user's weight for an optimization dimension. At each iteration, a user chooses k out of N recommended items. That choice induces a weight update to reflect the user's preferences. This approach is based on generalizing the work in [20], whose purpose is to

*These authors contributed equally to this research.

quantify workers' motivation on a crowdsourcing platform by observing their task adoption in multiple sessions. Refined weight vectors are used to re-rank recommendations produced for each user to match their preferences.

We formalize the notion of loss using scoring functions since they capture the utility of items to users. We model loss as the difference between proposed and optimal recommendations, i.e., those that would have been returned to a user if no other users were present. This loss is leveraged in the next iteration to compensate users, either individually or together. Building on this, we first formalize the problem of loss compensation and then design algorithms that solve it with different compensation strategies and recommendation adoption models. We consider two loss compensation strategies: *preference-driven*, where users are compensated in decreasing loss order by receiving 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. Our algorithms can be used with any recommendation engine. We integrate them into SoCRATe (System for Compensating Recommendations with Availability and Time), a novel recommendation framework that watches recommendation consumption and decides when to apply a loss compensation strategy to make up for sub-optimal recommendations due to limited item availability[1]. Our solution falls within the category of sequence-aware recommendation because it leverages both short-term knowledge on the users' actions in the last session (given by weights of the scoring function expressing their preferences) and longer-term knowledge on their past behavior given by an item-user matrix (see [21] for a survey on this topic).

We run extensive experiments to compare our solution to traditional recommendation and to examine the optimality of our algorithms with respect to loss compensation. To do so, we consider suitable measures: *i)* average and standard deviation of cumulative loss, and *ii)* response time. To showcase the benefit of our solution, we inject item availabilities in popular datasets, namely *Amazon Digital Music* [17], *Amazon Movies&TV* [17], and a *Task Recommendation* dataset [18]. Our results show that SoCRATe outperforms traditional recommendation by delivering *i)* better average cumulative loss and *ii)* better average standard deviation of cumulative loss over multiple iterations. Furthermore, we test our system with a synthetic dataset and a brute-force strategy that finds the best possible ordering of users at each iteration, i.e., the one that yields optimal loss compensation. We observe that SoCRATe is comparable to the optimal solution in terms of minimizing standard deviation of loss while being much faster.

Our empirical analysis finds that sorting users on decreasing cumulative loss always yields a reduction in standard deviation of loss, thereby decreasing disparities between users. In fact, user loss represents the difference in satisfaction between the optimal recommendation and that of the actual recommended items. Therefore, compensating loss over time allows us to treat users more equally, without running into situations of privilege. Several lessons can be learned. First, under limited item availability, recommender systems should be modified so that items are suggested in *round-robin* of availability rather than in the traditional *preference-driven* fashion. Second, the amount of loss compensation is a function of the distribution of conflicts among users: the higher the conflict, the more SoCRATe is able to compensate users.

The rest of the paper is organized as follows: we present our data model and formalize our problem in Section 2. We describe our solutions in Section 3. The experiments are reported in Section 4. Section 5 summarizes the related work. We conclude in Section 6.

[1]SoCRATe code is available at: https://github.com/TommasoD/SoCRATe_v2.

2 DATA MODEL AND PROBLEM

2.1 Data Model

We are given a set of users U and items I. We are interested in multi-session recommendations, where items are recommended to users in multiple iterations. An item $i \in I$ has an availability, $a(i,t)$, that represents, at each iteration t, the number of copies left of i. In our setting, an ordered list of N recommendations $R_u^t \subseteq I$ is provided to a user $u \in U$, s.t. each item in R_u^t has availability greater than 0 at iteration t. When u receives R_u^t, u may choose to adopt a subset $S_u^t \subseteq R_u^t$ of size $k \leq N$. $OptR_u^t$ is the set of optimal recommendations in terms of accuracy that a user u could get at iteration t. Due to the presence of other users and limited item availability, R_u^t may differ from $OptR_u^t$. This introduces the notion of "loss", i.e., the difference between R_u^t, the recommendations served to a user u at iteration t, and the optimal set $OptR_u^t$. Given a set of computed attributes defined as scalar functions $O = \{O_1, O_2, ..., O_m\}$, the loss of a user u at iteration t is a vector that records one loss per dimension, i.e., $loss_u^t = \langle ||O_1(OptR_u^t) - O_1(R_u^t)||, \ldots, ||O_m(OptR_u^t) - O_m(R_u^t)|| \rangle$, where $O_h(OptR_u^t)$, with $1 \leq h \leq m$, is a vector whose elements are obtained by applying function O_h to the elements of $OptR_u^t$ (and similarly for $O_h(R_u^t)$) and $||\alpha||$ is the ℓ_1-norm of α.

The purpose of the dimensions in O is to better characterize how the preferences of user u evolve over time. The importance given by user u to each dimension in O is captured as a weight vector W_u^t that is updated at each iteration based on the items user u decided to adopt. In this way, we are able to capture user preferences that depend on single item attributes (e.g., price of a book or a reward for a crowdsourcing task), or on a combination of multiple attributes (e.g., balancing price vs. reviews of a book or balancing reward vs. expected task completion time) or across items (e.g., diversity of recommended items). At each iteration, we refine a user's weight vector by running a regression that considers the k chosen recommendations against the remaining $N-k$ at the previous iteration as well as past adoption history. We use $\mathcal{L}_u^t$ to refer to the cumulative loss for user u up to iteration t:

$$\mathcal{L}_u^t = \sum_{j=1}^{t} W_u^j \cdot loss_u^j \ . \tag{1}$$

Equation (1) represents the sum of all the losses incurred in previous iterations weighted according to the user preferences. The higher the relevance of an aspect for a user, the higher its impact on the computed loss, ultimately leading to an increased cumulative loss. We assume our observations occur for a time horizon of T iterations. Table 1 summarizes our data model.

2.2 Motivating Example

Our goal is to set up a system that compensates users for previously experienced loss. We illustrate this with an example considering 3 users $U = \{u_1, u_2, u_3\}$, 6 items $I = \{i_1, \ldots, i_6\}$ with initial availability $a(i,0) = [1,2,2,2,2,1]$, two dimensions O_1 and O_2 that are *price* and *utility*, and weight vector W_u^t that represents

Table 1: Symbols of SoCRATe data model

Symbol	Description
U	set of users
I	set of items
O	set of optimization dimensions
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
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
$a(i,t)$	number of copies left of item i at time t
$a(i)$	initial availability $a(i,0)$
$SortU^t$	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
$\mathcal{L}_u^t$	cumulative loss for user u up to iteration t
T	number of observed iterations

the importance a user gives to dimensions (they are initialized for all users to 0.5). For conciseness, let us model price as $O_1(i_j) = 1 - 0.1j$ (e.g., i_3's price is 0.7) and assume users have similar preferences (i.e., utility), modeled as $O_2(i_j, u_k) = 1 - 0.1j - 0.1k$ (e.g., i_3's utility for user u_2 is 0.5), so that every user prefers i_1, then i_2, then i_3, etc. Let us also assume $N = 3$ recommended items and $T = 2$ iterations. The initial serving order of users is $[u_1, u_2, u_3]$. At the first iteration, the optimal recommendation is $OptR_u^0 = [i_1, i_2, i_3]$ for all users; however, only u_1 receives it, while u_2 and u_3 receive $R_{u_2}^0 = [i_2, i_3, i_4]$ and $R_{u_3}^0 = [i_4, i_5, i_6]$, respectively. This entails no loss for u_1, while the loss for u_2 is computed as $loss_{u_2}^0 = \langle ||O_1(OptR_{u_2}^0) - O_1(R_{u_2}^0)||, ||O_2(OptR_{u_2}^0) - O_2(R_{u_2}^0)|| \rangle$, i.e., $\langle ||[0.9, 0.8, 0.7] - [0.8, 0.7, 0.6]||, ||[0.8, 0.7, 0.6] - [0.7, 0.6, 0.5]|| \rangle = \langle 0.3, 0.3 \rangle$, hence $\mathcal{L}_{u_2}^0 = 0.3$; similarly, $\mathcal{L}_{u_3}^0 = 0.9$. Such losses should be used to modify the order users will be served at the next iteration ($[u_3, u_2, u_1]$, in this case) and to update the weights.

2.3 Problem Formulation

Given a time horizon T (number of observed iterations), the problem we formulate aims to produce a set of recommendations for a user at each iteration in such a way that the standard deviation of the cumulative loss is minimized. To this end, let us denote with

$$\overline{\mathcal{L}^t} = \frac{1}{|U|} \sum_{u \in U} \mathcal{L}_u^t \tag{2}$$

the mean cumulative loss of all users at a given iteration t, and with

$$\mathcal{S}^t = \sqrt{\frac{1}{|U|} \cdot \sum_{u \in U} \left(\mathcal{L}_u^t - \overline{\mathcal{L}^t} \right)^2} \tag{3}$$

the standard deviation of cumulative loss of all users. With this, our problem is to find at each iteration $t \in T$ and for each user u, a set of N recommendations R_u^t such that:

$$R_u^t = \text{argmin } \mathcal{S}^t. \tag{4}$$

In other words, for any given iteration, our purpose is to find, for each user, the best recommendations that will minimize the standard deviation of cumulative loss across all users. To avoid inequality among users, we enforce equal treatment over time. In fact, minimizing the standard deviation guarantees the minimum cumulative loss distance between users for each iteration. In turn, equal user treatment increases recommendation accuracy and, consequently, favors user retention.

Our problem is a variant of the Knapsack Problem [4]. Each item has a value that corresponds to its contribution to the scoring function and a weight (in our case the weight vector W_u^{t-1} of the user), and we want to find N items for each user that optimize an aggregation of values under a capacity constraint. This means the value of recommending an item to a user evolves over time. The challenge in solving this problem is to keep track and account for this dynamic aspect. Despite the hardness of this problem, we will see in Section 3 how we can satisfactorily address it with solutions based on heuristic considerations.

3 OUR SOLUTION

We first provide an overview of the architecture of SoCRATe and then we describe our algorithms.

3.1 Architecture of SoCRATe

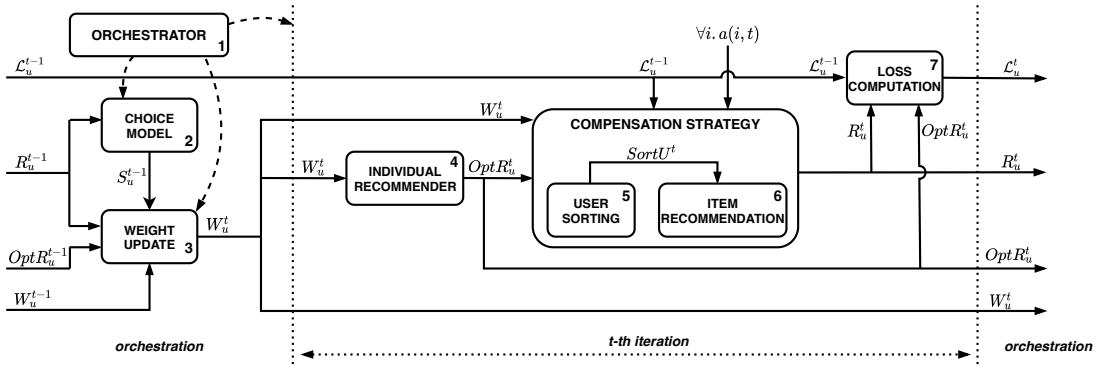
The architecture of SoCRATe is provided in Figure 1. The Orchestrator (1) is in charge of determining the moment at which recommendations are computed. It monitors when users are done consuming recommendations during iteration $t-1$ and uses that to determine when to produce new recommendations and start iteration t. To better model this scenario, we implement two time granularity splits: *fixed* (all users consume items at the same time) and *user-group* (different sets of users consume their recommendations at different time). The Orchestrator makes use of the recommendations consumed by users in S_u^{t-1}, according to different adoption models described in Section 3.2 (2), and triggers the Weight Update (3) module to obtain new weights. Weight computation receives S_u^{t-1}, R_u^{t-1}, $OptR_u^{t-1}$ and returns the updated weights W_u^t. Each entry of W_u^t represents the preference of a user for an optimization dimension. To refine weight vectors, we run a regression. Refined weight vectors are used to re-rank the recommendations produced for each user to match their feedback.

Following the weight update, the Individual Recommender module (4) is called to produce new recommendations based on the weights of each user. SoCRATe is compatible with any single-user recommender; in our implementation we use KNN [15]. This module returns $OptR_u^t$, which is passed to the Compensation Strategy module. The User Sorting module (5) sorts users according to their loss (see Section 3.2 for different sortings) 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 that are fed to the Loss Computation module (7) to compute each user's incurred loss. SoCRATe then moves to the next iteration by calling the Orchestrator.

3.2 Algorithms in SoCRATe

The generic algorithmic pattern that contains the main steps of our solution is shown in Algorithm 1. We describe the main steps of our solution, and illustrate it for a *fixed* time granularity in Algorithm 2. The process takes as input t, the moment where a new iteration starts according to the Orchestrator, and for each user u, $\mathcal{L}_u^{t-1}$, R_u^{t-1}, $OptR_u^{t-1}$, and W_u^{t-1}. It outputs, for each user, $\mathcal{L}_u^t$, R_u^t, $OptR_u^t$, and W_u^t. The systems can be instantiated according to different user sortings, compensation strategies, and recommendation adoption models. All these system parameters allow us to describe a variety of scenarios.

Figure 1: Architecture of SoCRATe: internal structure of the framework, components and interactions.



Algorithm 1 SoCRATe Algorithm

Input: $\forall u, \mathcal{L}_u^{t-1}, R_u^{t-1}, OptR_u^{t-1}, W_u^{t-1}$
Output: $\forall u, \mathcal{L}_u^{t}, R_u^{t}, OptR_u^{t}, W_u^{t}$

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

User Sorting. Different policies can, in principle, be adopted for sorting users. However, SoCRATe commits to a user sorting based on *loss*, which, as described in Section 4, is the most suitable for compensating users. Yet, in order to compare this choice to other possibilities, in the experiments we run alternative algorithms that adopt other policies, namely: *no*, where the position of users never changes; *shuffle*, where the position of users is set at random at each iteration; *forced*, which considers all possible orderings of users. We observe that sorting users with respect to loss is a way to intuitively reduce disparity among users, targeted by our Problem Statement, since those who incurred the highest loss are given higher priority in the next round of recommendations.

Time Granularity. This determines the moment at which recommendations are re-optimized based on user feedback, and for which user. We consider a *fixed* granularity, where recommendations of all users are re-optimized at the same time, and *user-group*, where the best time is determined for a group of users, divided according to their consumption rate (low, medium, high). Our system also accommodates the case of *user-group* time granularity by simply considering the users in the same group instead of all users.

Adoption Model. There are multiple ways to simulate a recommendation adoption model [3, 9, 36], i.e., the model that reflects which set of items is chosen by a user out of all the recommendations R_u^t. We consider three models: *rank*, where users are assumed to choose their items in the order in which they are ranked by the scoring function; *utility*, where users are assumed to choose items in decreasing order of their utility with respect to their profile; *random*, where users are assumed to choose items randomly.

Algorithm 2 Preference-Based Compensation Strategy

Input: $SortU^t, OptR_u^t, \forall i.\ a(i,t)$

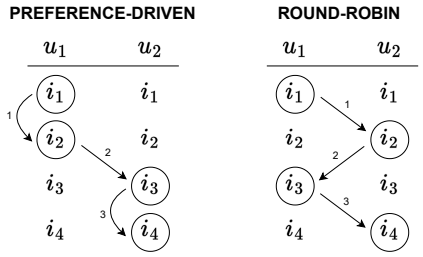
1: $\forall i.\ a(i,t+1) \leftarrow a(i,t)$
2: **for all** $u \in SortU^t$ **do**
3: **for** $n \leftarrow 1$ **to** N **do**
4: **while** $OptR_u^t$ not empty and $a(i,t+1) > 0$ **do**
5: $i \leftarrow \text{pop}(OptR_u^t)$
6: $R_u^t \leftarrow R_u^t \cup i$
7: $a(i,t+1) \leftarrow a(i,t+1) - 1$
8: **end while**
9: **end for**
10: **end for**
11: **return** $\forall u.R_u^t$

Compensation Strategy. This is the strategy to compensate users, either individually or in groups according to the time granularity. We consider two strategies:

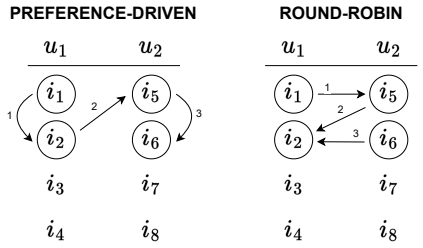
(1) *preference-driven*, where users are compensated in decreasing loss order by receiving all their N recommendations at once before the user with the next smallest loss is served.
(2) *round-robin*, where users are considered in decreasing loss order and items are recommended to them in *round-robin* order of item availability.

When the *preference-driven* compensation is adopted (see Algorithm 2), N items are recommended to the first user before moving to the next user in the sorted list of users. The *round-robin* compensation strategy (see Algorithm 2 inverting lines 2 and 3) recommends one item at a time to each user (according to the order dictated by the sorted list of users), until N items are recommended to each user. *Preference-driven* is expected to maximize the satisfaction of the first users in the sorted list. As a result, this strategy is preferred when the system detects that some specific users are unsatisfied and thus may be prone to abandon the system, rapidly compensating for their incurred loss. On the other hand, *round-robin* is preferred when consumption is the main goal, since it is likely that all users will get items that they like, which will affect their consumption rate and the overall user retention.

Choosing the right compensation strategy is crucial when there is a limited set of items that are very popular among all users and when the availability of these items is limited and not sufficient to satisfy all users. The example in Figure 2 illustrates the intuitive difference between our compensation strategies. In Figure 2a, users have similar recommendations (i.e., similar item rankings) and item availability is low (= 1). In Figure 2b, their



(a) Same preferences



(b) Different preferences

Figure 2: Illustration of compensation strategies. The set of optimal recommendations of each user is shown in decreasing relevance. The numbers on the arrows reflect the order in which items will be recommended. On the top, the two users compete as they have similar recommendations. On the bottom, the two users do not compete.

recommendations differ. When the two users compete, we can see that *preference-driven* favors u_1, while *round-robin* treats u_1 and u_2 equally. Section 4 will examine the benefits and drawbacks of each compensation strategy. We observe that adopting a compensation strategy also goes in the direction of reducing disparity among users, thus addressing our Problem Statement.

3.3 Complexity of the Algorithms

The single-user recommender system computes recommendations for each user just once at the beginning of the first iteration. Each user is associated with a list of items ordered by utility: at each iteration the system iterates over the list as new items are recommended to her/him. SoCRATe internally adapts recommendations at each iteration by updating user weights, which represent the behaviour of the user with respect to the objective functions.

Regarding complexity, if we assume that N, K and $|O|$ in Table 1 are constant (and indeed they correspond to small integer values in our implementation), the worst-case asymptotic complexities of the various procedures invoked in Algorithm 1 are as follows: `ChoiceModel`, `WeightUpdate` and `LossComputation` are in $O(1)$; `IndividualRecommender` is in $O(|I| \log |I|)$; `ItemRecommendation` is in $O(|I|)$. All of these are included in a loop iterating through the set of all users U, thereby incurring a $O(|U||I| \log |I|)$ complexity. `UserSorting` can actually be executed outside the aforementioned loop for all users at once, with a complexity of $O(|U| \log |U|)$. Overall, the complexity of Algorithm 1 is $O(|U||I| \log |I| + |U| \log |U|)$.

4 EXPERIMENTS AND RESULTS

The purpose of our experiments is twofold: *(i)* verify, on real-world datasets, that SoCRATe outperforms traditional recommendation by delivering (1) a better average cumulative loss and (2) a better average standard deviation of the cumulative loss over multiple iterations, and *(ii)* use synthetic data to compare SoCRATe to a brute-force optimal solution and verify that the performance is comparable in terms of minimizing the standard deviation of the cumulative loss while being much faster.

4.1 Settings

4.1.1 Measures.

Average Cumulative Loss. Since our overall goal is to minimize user loss, we first measure for each user the total cumulative loss at iteration t, denoted by $\mathcal{L}_u^t$ (Equation (1)), and then we compute the mean cumulative loss of all users at a given iteration t, denoted by $\overline{\mathcal{L}^t}$ (Equation (2)). With this, we obtain an indication of the overall loss incurred throughout the experiment by computing the average loss across all iterations for all users:

$$\overline{\mathcal{L}} = \frac{1}{T} \sum_{t=1}^{T} \overline{\mathcal{L}^t} \ . \tag{5}$$

Standard Deviation of Cumulative Loss. To capture disparity of compensation across users, we also measure the standard deviation of cumulative loss of all users, denoted by $\mathcal{S}^t$ (Equation (3)) and, similarly to Equation (5), we obtain an overall measure of the disparity incurred throughout an experiment by computing the average of $\mathcal{S}^t$ across all iterations:

$$\overline{\mathcal{S}} = \frac{1}{T} \sum_{t=1}^{T} \mathcal{S}^t \ . \tag{6}$$

4.1.2 Algorithmic Variants. Table 2 summarizes our configuration variants. To mimic traditional recommenders, we designed two baselines for the real-world experiments: Trad and Shuffle_Trad. Hence, in both baselines, items are assigned to users according to *preference-driven* compensation, just like in traditional recommendation. The two baselines differ in the order users are considered. In Trad, users are always served in the same order at each new iteration. This means that the same users are always served first at each iteration. If these users consume items that are also preferred by users who are served after them, this will inevitably induce a loss for those "unlucky users", as they will receive less optimal items. In Shuffle_Trad, users are served in a random order at each new iteration. Other strategies with different user sortings are instances of SoCRATe. For the experiments on real data, we considered two versions of SoCRATe: SoPD and SoRR that adopt *preference-driven* and *round-robin* compensations, respectively.

Due to lack of direct competitors, we also designed BruteForce, which considers all possible orderings of users and finds the best ordering at each iteration. BruteForce has three variants: BruPD implements a *preference-driven* compensation with fixed time granularity, BruRR implements a *round-robin* compensation with fixed time granularity, and BruGroup implements a *round-robin* compensation with *user-group* time granularity. These variants are compared, respectively to three variants of SoCRATe: SoPD that implements the *preference-driven* compensation, SoRR that implements the *round-robin* compensation, and finally SoGroup that implements the *round-robin* compensation with *user-group* time granularity. The variants using the *fixed*

Table 2: We consider 6 variants for the synthetic and real-world data experiments. Variants differ according to their configuration for two parameters: the loss compensation strategy and the user sorting. The user sorting has 4 different values: *no, shuffle, forced*, and *loss* as described in Section 3.2. In the experiment on real datasets, each variant implements only one loss compensation strategy. Instead, for the brute-force validation both variants of loss compensation are considered.

Type of Experiment	Variant	Loss Compensation		User Sorting				Granularity		Adoption Model
		pref-driven	*round-robin*	*no*	*shuffle*	*forced*	*loss*	*fixed*	*group*	*rank, utility, rand*
real-world datasets	Trad	✓		✓				✓		✓
	Shuffle_Trad	✓			✓			✓		✓
	SoPD	✓					✓	✓		✓
	SoRR		✓				✓	✓		✓
brute-force validation	BruPD	✓				✓		✓		✓
	BruRR		✓			✓		✓		✓
	BruGroup		✓			✓			✓	only rank
	SoPD	✓					✓	✓		✓
	SoRR		✓				✓	✓		✓
	SoGroup		✓				✓		✓	only rank

Table 3: Parameters of real-world datasets.

Parameters	AMZ_MUS	AMZ_MOV	TASK_REC
Number of users $\|U\|$	5541	3074	110
Number of items $\|I\|$	3568	6423	1806
Number of iterations T	15	15	15
Recommendations per user N	5	5	5
Adopted items per user k	3	3	3
Mean item availability	150	30	5

Table 4: Parameters of SYN_DS.

Parameters	Values
Number of users $\|U\|$	4 or 8
Number of items $\|I\|$	$(2 \cdot \|U\| \cdot (k \cdot (\|I\| - 1) + N))/a(i)$
Number of iterations T	6 or 15
Recommendations per user N	5 per iteration
Adopted items per user k	3 per iteration
Item availability $a(i)$	1 or $\frac{1}{2} \cdot \|U\|$ $\forall i \in I$
User conflict	CONFLICT or NO_CONFLICT

time granularity are tested against all three adoption models, and our results indicate that the adoption model has little or no impact on compensation; therefore, for the two variants adopting the *user-group* time granularity - BruGroup and SoGroup - we only show results regarding the *rank* adoption model. Each one of the BruteForce variants reorders users at every iteration, resulting in $(|U|!)^T$ different runs, i.e., the number of possible permutations of users for each iteration, where T is the total number of iterations considered in an experiment. A run represents an execution with a given ordering of users at each iteration. Since testing $(|U|!)^T$ runs would be unfeasible already for moderate values of $|U|$ and T, we choose to limit our experiments to 1 million runs sampled uniformly at random from the set of all possible runs of BruteForce. The choice of parameter values is such that it is possible to sample from a set of at least 1M runs.

For the BruteForce variants, we will additionally study how SoCRATe compares to the different 1M runs when these fit a suitable distribution for our data, i.e., to get an indication of how well SoCRATe performs with respect to all other possible runs.

4.1.3 Datasets. We test SoCRATe over three real-world datasets: *Amazon Digital Music* (AMZ_MUS) [17], *Amazon Movies&TV* (AMZ_MOV) [17], *Task Recommendation* (TASK_REC) [18]. All datasets are pre-processed as follows: we keep only users and items with at least 5 reviews each in AMZ_MUS, and at least 20 reviews in AMZ_MOV. This is to avoid cold start problems. In TASK_REC, users (workers) complete and rate task sessions. The dataset consists of 200 workers and 20k tasks. For the experiments, we consider a subset of 1806 tasks. Each task belongs to one of 10 types, such as tweet classification, image transcription, or sentiment analysis, and has a reward value between $0.01 and $0.05. Table 3 contains additional details for the real datasets.

We also created a family of synthetic datasets SYN_DS. Table 4 describes the parameters involved in the configurations of SYN_DS. We define two main variants of SYN_DS which differ in the scheme of user conflicts, namely CONFLICT and NO_CONFLICT. The goal of CONFLICT is to maximize conflict between users, thus in this configuration, all users like the same items. Under the NO_CONFLICT variant, overlaps between user recommendations follow a more realistic approach, where few items are popular among all users, few items are mildly popular, and the rest of the items are not very popular. We consider a number of items equal to $|I| = 2 \cdot \frac{|U| \cdot (T-1) \cdot |S_u^t| + |U| \cdot N}{a(i)}$, which corresponds to twice as many items as the minimum number of items that would be needed to perform T iterations with $|U|$ users.

Item Availability. As item availability is typically not made public in real data, we propose to inject it in all our datasets (including synthetic data) following a normal distribution. This allows us to study the benefits of our solution for different availability values. The total number T of iterations is set to 15, as we noticed that, in real datasets, after a rather small number of iterations, cumulative loss seems to stabilize, intuitively because after the most interesting items for a user have been consumed, there is little difference between all remaining items.

4.2 Summary of Findings

Our experiments helped us make three important observations:

O1. Sorting users on decreasing cumulative loss always yields a reduction in the standard deviation of cumulative loss ($\overline{S}$), which results in decreasing disparities between users.

O2. Satisfying **O1** does not impact negatively the average cumulative loss ($\overline{\mathcal{L}}$).
O3. Observation **O1** is satisfied regardless of the compensation strategy. Yet, recommending items to users in *round-robin* of item availability yields the best results.

Lessons Learned. First, under limited item availability, recommender systems should be modified so that items are suggested in *round-robin* rather than in *preference-driven* compensation strategy. Second, under limited item availability, the amount of compensation SoCRATe is able to provide depends on the distribution of conflict among users: the higher the conflict, the more SoCRATe compensates users.

Impact of Recommendation Adoption Models. We found that *rank* and *utility* adoptions always outperform adopting items at random among those recommended. In that case, users end up consuming items that are less optimal. Moreover, with *random* adoption, the worsening of $\overline{\mathcal{S}}$ is even more pronounced (up to 5× for SoRR on AMZ_MOV) as it partially disrupts compensation.

Impact of Time Granularity. Although this feature does not directly entail an improvement in disparity among users, it helped us model more realistic scenarios, where users vary in consumption rates. Moreover, even with smaller groups of users, SoCRATe still achieves loss improvements.

Impact of User Conflict. SoCRATe outperforms most runs of BruteForce, and is very close to the optimal, especially when users like the same items (CONFLICT). We also notice that the compensation performance is independent of the other parameters, showing that our solution works well with both compensation strategies and with various combinations of number of users, iterations, and item availability. When considering a more realistic scheme for the distribution of items that users like (NO_CONFLICT), SoCRATe's performance slightly worsens, but still remaining considerably better than the average BruteForce run. These results are in line with our intuition that SoCRATe is designed for cases of limited availability with many conflicts among users. On the contrary, when conflict among users is low (i.e., their recommended items largely differ), the performance of our solution moves slightly away from the optimal, obtaining better results than BruteForce in 90% of the experiments.

Real-World vs Synthetic Data. Experiments on real-world datasets confirm the choices we made for SoCRATe. Specifically, when considering the average cumulative loss and its standard deviation, SoCRATe outperforms traditional recommendation for both *preference-driven* and *round-robin* variants. SoCRATe with *preference-driven* compensation, SoPD, decreases the standard deviation by 15.6% with respect to traditional recommendation. SoCRATe with *round-robin*, SoRR, decreases the standard deviation by 57.0% with respect to traditional recommendation.

4.3 Real Datasets Experiments

We first provide a step-by-step description of SoCRATe applied to AMZ_MOV for examining compensation, and then present the results for AMZ_MUS and TASK_REC for examining cumulative loss.

4.3.1 Examining Compensation. In Figure 3 we report the results of SoCRATe and Trad on AMZ_MOV. Figure 3d shows how Trad treats five users: u_0, u_{768}, u_{1536}, u_{2305} and u_{3073}. The names of the five different users indicate their relative position in the user sorting of the baseline option, i.e., we consider the first user and those at 1/4, 2/4, 3/4, and last of the ranking. The loss experienced by each user is different at each iteration; furthermore, the maximum cumulative loss is higher than 3.0 for the baseline and 1.75 for SoCRATe (see Figure 3b). To understand the reason behind this disparity, the reader has to get inside SoCRATe's logic and specifically the *preference-driven* compensation chosen for the implementation of traditional recommendation baselines.

In Table 5, we selected five different iterations: the first three (0, 1, 2) and the last two (13, 14). For each of them, we report both the loss at the current iteration $L_u^t = W_u^t \cdot loss_u^t$ and cumulative loss $\mathcal{L}_u^t$ experienced by each user. The order in which each user is served is important. Indeed, u_0 never experiences any loss, while the last user u_{3073} starts with a 0.25 loss, and reaches a final cumulative loss equal to 3.24.

The lower part of Table 5 reports losses and user ranks for SoCRATe. The main difference with the baseline is that, at each iteration, users are re-sorted according to the loss they accumulated until that point. The other difference is the use of a *round-robin* compensation, instead of *preference-driven*. In Table 5, we see how the compensation works in SoCRATe. For example, if we consider iterations 13 and 14, we can see how user u_{768}, who is experiencing the highest cumulative loss at iteration 13, is served before the other four users at iteration 14 (global ranking position 35). Conversely, user u_{1536}, who has the lowest cumulative loss, is served globally as 2327th. As shown in Table 5, in the first iteration, users are sorted in the same order as in the baseline, then, in the following iterations, users are re-sorted to minimize loss. As can be seen, both re-sorting users according to the loss and the *round-robin* compensation strategy is effective in reducing disparate treatment among users. In fact, the maximum cumulative loss is almost halved (from 3.24 to 1.75), and also the standard deviation among users is greatly decreased.

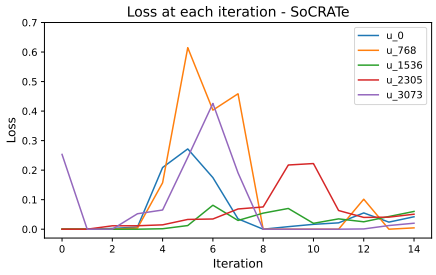
4.3.2 Examining Cumulative Loss. Our experiments on the three real-world datasets are shown in Table 6. For each dataset we select *fixed* time granularity and three adoption models (*rank, utility* and *random*). The rows report two different metrics, computed for both the baseline algorithms and SoCRATe: the average cumulative loss $\overline{\mathcal{L}}$ and the standard deviation of the users cumulative loss $\overline{\mathcal{S}}$. We compare four systems: two baselines (Trad and Shuffle_Trad) and two versions of SoCRATe: SoPD and SoRR that adopt *preference-driven* and *round-robin* compensations, respectively.

Both versions of SoCRATe always outperform the two baselines, both for what regards the average cumulative loss $\overline{\mathcal{L}}$ and average standard deviation of the cumulative loss $\overline{\mathcal{S}}$ on all three datasets.

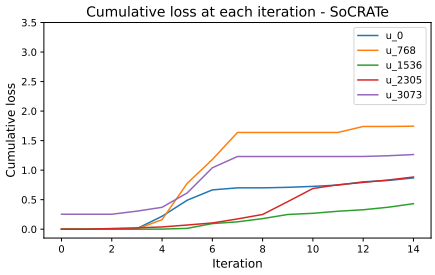
The only exceptions are on AMZ_MUS, in which, for the *random* and *rank* adoption models, the two baselines provide a slightly lower average cumulative loss compared to SoPD. Note however that SoPD provides a lower standard deviation $\overline{\mathcal{S}}$ and hence a better compensation. Both system configurations decrease $\overline{\mathcal{S}}$ with respect to the baselines: specifically, SoPD decreases the standard deviation by 15.6%, and SoRR decreases the standard deviation by 57% with respect to both baselines. Overall, SoRR achieves a lower loss and a better compensation than SoPD.

4.4 Comparison with Optimal Brute Force

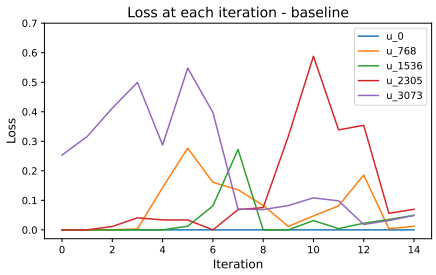
To test the optimality of our solution, we designed a BruteForce strategy that considers all possible orderings of users at each iteration so as to find the best one, i.e., the one that yields the



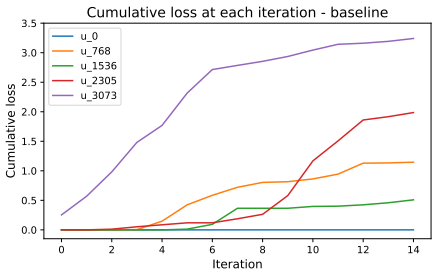
(a) SoRR: Loss



(b) SoRR: Cumulative Loss



(c) Trad: Loss



(d) Trad: Cumulative Loss

Figure 3: Loss per iteration and cumulative loss per iteration from experiments on AMZ_MOV for Trad and SoCRATe.

Table 5: Comparison between Trad baseline and SoRR with *round-robin* compensation on AMZ_MOV: we selected five different iterations and, for each one, we report the position of the user in the sorting Pos, the loss L_u^t, and cumulative loss $\mathcal{L}_u^t$ experienced by each user. Due to space limitations, we only report the first three and the last two iterations.

System	User	It #0			It #1			It #2			It #13			It #14		
		Pos	L_u^t	$\mathcal{L}_u^t$	Pos	L_u^t	$\mathcal{L}_u^t$	Pos	L_u^t	$\mathcal{L}_u^t$	Pos	L_u^t	$\mathcal{L}_u^t$	Pos	L_u^t	$\mathcal{L}_u^t$
Trad	u_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	u_{768}	768	0	0	768	0	0	768	0	0	768	0	1.13	768	0.01	1.15
	u_{1536}	1536	0	0	1536	0	0	1536	0	0	1536	0.04	0.46	1536	0.05	0.51
	u_{2305}	2305	0	0	2305	0	0	2305	0.1	0.1	2305	0.06	1.92	2305	0.07	1.98
	u_{3073}	3073	0.25	0.25	3073	0.32	0.57	3073	0.41	0.98	3073	0.03	3.19	3073	0.05	3.24
SoRR	u_0	0	0	0	218	0	0	1031	0	0	1121	0.02	0.83	1373	0.04	0.87
	u_{768}	768	0	0	986	0	0	1582	0	0	3	0	1.74	35	0	1.74
	u_{1536}	1536	0	0	1735	0	0	2100	0	0	2327	0.05	0.37	2282	0.06	0.43
	u_{2305}	2305	0	0	2436	0	0	2601	0.01	0.01	1216	0.04	0.83	1274	0.05	0.88
	u_{3073}	3073	0.25	0.25	376	0	0.25	332	0	0.25	451	0.01	1.24	496	0.02	1.26

Table 6: Experiments comparing the two baselines (Trad and Shuffle_Trad) with two versions of SoCRATe (SoPD and SoRR) on the three real-world datasets, using two compensation strategies and three adoption models.

System	Metrics	AMZ_MOV			AMZ_MUS			TASK_REC		
		Rank	*Utility*	*Random*	*Rank*	*Utility*	*Random*	*Rank*	*Utility*	*Random*
Trad	$\overline{\mathcal{L}}$	0.4953	0.5266	0.6962	0.4862	0.4782	0.5384	0.0990	0.0990	0.1235
	$\overline{\mathcal{S}}$	0.6621	0.7204	0.9330	0.5440	0.6242	0.7801	0.1961	0.1961	0.2484
Shuffle_Trad	$\overline{\mathcal{L}}$	0.4175	0.4104	0.6958	0.4101	0.3906	0.5217	0.0640	0.0663	0.0747
	$\overline{\mathcal{S}}$	0.5038	0.4843	0.8311	0.4070	0.4285	0.6390	0.1636	0.1749	0.2180
SoPD	$\overline{\mathcal{L}}$	0.3884	0.3780	0.6750	0.4162	0.3804	0.5439	0.0446	0.0446	0.0527
	$\overline{\mathcal{S}}$	0.3111	0.2964	0.5864	0.2963	0.2928	0.5598	0.0851	0.0851	**0.0920**
SoRR	$\overline{\mathcal{L}}$	**0.0697**	**0.0940**	**0.5623**	**0.1268**	**0.1845**	**0.4345**	**0.0116**	**0.0116**	**0.0463**
	$\overline{\mathcal{S}}$	**0.0837**	**0.1049**	**0.5568**	**0.1315**	**0.1736**	**0.4490**	**0.0485**	**0.0485**	0.0980

highest loss compensation. BruteForce is expected to be costly in terms of execution time, since it executes a high number of runs, while its best run is expected to be optimal or close to optimal in terms of loss compensation. The starting order of users at

the first iteration is the same for both SoCRATe and BruteForce. Additionally, we set item availability to a low amount for the following two reasons. First, we want to simulate a low-availability scenario, where the order of users becomes critical when minimizing loss. Second, we seek to validate the hypothesis that SoCRATe is able to compensate even in situations of extreme unequal treatment among users. In fact, if user preferences are similar and few copies of items are available, disparity will be maximized at least in the first iterations.

4.4.1 User Conflict Configurations. To better validate SoCRATe, we test our system against two kinds of synthetic datasets: `CONFLICT` and `NO_CONFLICT`. Moreover, we study SoCRATe with different adoption models and time granularities.

The Case of Conflicting Users. We run BruteForce on `CONFLICT` for 6 and 15 iterations so as to observe SoCRATe's loss compensation both in the short and long terms. We run BruteForce for both compensation strategies: *round-robin* (BruRR) and *preference-driven* (BruPD). We report results in a variety of scenarios: with 4 and 8 users, over 6 and 15 iterations, and considering 1 and half the number of users as possible item availabilities for the *round-robin* case. We run additional tests with *round-robin*, because SoRR is the variant of SoCRATe that performs best, as shown in Section 4.3.

The Case of Non-Conflicting users. Under `NO_CONFLICT`, 1/6 items are popular among all users, with a utility between 0.8 and 1; 1/6 of the items are mildly popular, with a utility between 0.6 and 0.8; the remaining 2/3 are not popular, with a utility between 0 (the minimum) and 0.6. Preferences are assigned at random to users according to these ranges. We test with *round-robin* compensation, with combinations of 6 and 15 iterations for 4 and 8 users.

Adoption and Time Granularity. Finally, we compare SoRR to BruRR with the different adoption models *rank*, *utility* and *random*, and validate the *user-group* time granularity comparing SoGroup with BruGroup. We set the other parameters to a standard configuration consisting of 15 iterations, 4 users, availability of 1 per item, and `CONFLICT`. For the *user-group* variant, we consider 15 iterations, *round-robin* compensation, `CONFLICT`, and 8 users divided into 3 groups according to their consumption rate: fast, medium, and slow consumers. In our simulations, fast consumers receive new recommendations at each iteration, medium consumers every two iterations, and slow consumers every four iterations. We test for item availability of 1 and 4 per item.

4.4.2 Results. Tables 7 and 8 report our results. Under `CONFLICT`, the standard deviation of the cumulative loss $\overline{\mathcal{S}}$ is minimized in all cases, with both SoRR and SoPD achieving a better performance than most of the runs of BruRR and BruPD respectively; this is true for every configuration, regardless of $a(i)$, $|U|$ and T. Under `NO_CONFLICT`, results are slightly worse than in `CONFLICT`. This allows us to conclude that SoRR is better suited for extreme situations of limited availability and conflicts between users.

Nonetheless, if we try to have an indication of how well SoCRATe scores with respect to the BruteForce runs, we find that our solution consistently obtains performances on a par with the best runs of BruRR even in realistic situations, confirming the empirical results presented in Section 4.3. To this end, we fit a continuous distribution to the values obtained from the 1

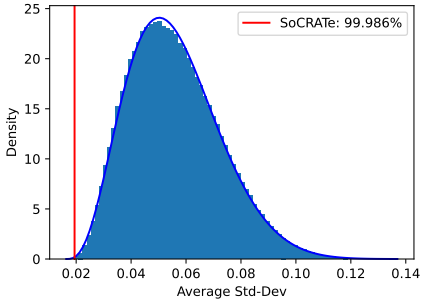


Figure 4: BruRR runs distribution for the following configuration: `CONFLICT`, $|U| = 4$, $T = 15$, $a(i) = 1$.

million runs of the BruteForce variants and we empirically chose a beta distribution as the best-fitting distribution to our experimental data. Then, we considered the probability density function of the distribution of standard deviations of cumulative losses and determined the probability that SoCRATe performs better than BruteForce. In particular, we computed the area $A_\sigma = 1 - (cdf(X))$, where cdf is the cumulative distribution function for the value X of $\overline{\mathcal{S}}_{BF}$ obtained by SoCRATe. The goal is to maximize this probability, i.e. the area located to the right of the value obtained by SoCRATe. Figure 4 shows that SoCRATe maximizes this area and lies very close to the absolute best run.

Additionally, Table 8 shows that SoRR performs well independently of the adoption model, and that its loss compensation allows us to always achieve a balance between users in terms of cumulative loss. With the *user-group* time granularity SoGroup is also close to the optimal regarding $\overline{\mathcal{S}}$, with only a small decrease in the performance with a higher availability.

Finally, Figure 5 reports the performance of SoRR when ranging item availability from 1 to 6 copies of each item under the `NO_CONFLICT` configuration. From our experiments we noticed that increasing the value of item availability the performance of our solution decreases. In fact, when the number of users is equal to 8 and item availability higher than 4 (i.e. half of the number of users), the experimental scenario cannot be considered of limited availability anymore, i.e., the case for which our solution has been specifically designed to cope with.

Our preliminary response time investigation found that both the sorting of the users on loss (as opposed to not sorting them) and the presence of conflicts among users (i.e., `CONFLICT` or `NO_CONFLICT`) have negligible impact on time required to provide recommendations to users (for the numbers of users we considered).

5 RELATED WORK

Our work relates to sequence-based recommendations [7, 21, 34]. It also relates to session-based recommendations that aim to capture short-term dynamic user feedback [33]. Sequential recommenders differ from session-based in that the order of individual items matters and is used to predict the next item a user is most likely to consume. *Our work differs in many ways: we introduce dynamic time splitting and compensation strategies to make up for loss; we combine historical user data with dynamic user feedback based on various recommendation adoption models.* To the best of our knowledge, no other work encompasses all features of SoCRATE. Table 9 contains some works that are most related to ours.

Table 7: Results on SYN_DS comparing different variants of SoCRATe and BruteForce for various conflict configurations (CONFLICT and NO_CONFLICT), compensation strategies and values of $a(i)$, $|U|$ and T. We show the average cumulative loss ($\overline{\mathcal{L}}$) and the average standard deviation of the cumulative loss ($\overline{\mathcal{S}}$) for both SoCRATe ($\overline{\mathcal{S}_S}$ and $\overline{\mathcal{L}_S}$) and BruteForce ($\overline{\mathcal{S}_{BF}}$ and $\overline{\mathcal{L}_{BF}}$). Results when $\overline{\mathcal{S}_S}$ is within the first percentile and $\overline{\mathcal{L}_S}$ is better or equal to the mean value of BruteForce are in bold.

Parameters						Metrics						
Algorithms	**Conflicts**	$a(i)$	$\|U\|$	T	$\|I\|$	$\overline{\mathcal{L}}_{BF}^{min}$	$\overline{\mathcal{L}}_{BF}^{mean}$	$\overline{\mathcal{L}}_S$	$\overline{\mathcal{S}}_{BF}^{min}$	$\overline{\mathcal{S}}_{BF}^{mean}$	$\overline{\mathcal{S}}_S$	A_σ
BruRR, SoRR	CONFLICT	1	4	6	160	0.5660	0.5663	0.5669	0.0144	0.0410	*0.0200*	99.06%
			4	15	376	0.8959	0.9004	**0.8983**	0.0164	0.0559	**0.0194**	99.98%
			8	6	320	0.6364	0.6434	**0.6420**	0.0194	0.0477	**0.0222**	99.99%
			8	15	752	0.9701	0.9877	**0.9877**	0.0234	0.0596	**0.0247**	99.99%
		2	4	6	80	0.4284	0.4297	0.4299	0.0068	0.0240	**0.0083**	99.63%
			4	15	188	0.7372	0.7603	0.7609	0.0100	0.0324	**0.0144**	99.84%
	NO_CONFLICT	1	4	6	160	0.2311	0.2504	**0.2459**	0.0186	0.0539	0.0225	98.46%
			4	15	376	0.2288	0.2659	0.2927	0.0231	0.0647	0.0397	97.67%
			8	6	320	0.3284	0.3596	**0.3591**	0.0175	0.0614	0.0361	97.86%
			8	15	752	0.2822	0.3316	**0.3240**	0.0259	0.0755	0.0592	92.61%
BruPD, SoPD	CONFLICT	1	4	6	160	0.7408	0.7422	0.7423	0.0840	0.2402	**0.1156**	99.08%
			4	15	376	1.0021	1.0127	**1.0048**	0.0585	0.2162	**0.0634**	99.99%
			8	6	320	1.0055	1.0105	1.0119	0.1141	0.2525	**0.1222**	99.99%
			8	15	752	1.2731	1.2994	**1.2952**	0.0745	0.1946	**0.0638**	99.99%

Table 8: Results on SYN_DS comparing BruRR against SoRR with different adoption models, and BruGroup against SoGroup.

Parameters			Metrics						
Algorithms	**Adoption**	$a(i)$	$\overline{\mathcal{L}}_{BF}^{min}$	$\overline{\mathcal{L}}_{BF}^{mean}$	$\overline{\mathcal{L}}_S$	$\overline{\mathcal{S}}_{BF}^{min}$	$\overline{\mathcal{S}}_{BF}^{mean}$	$\overline{\mathcal{S}}_S$	A_σ
BruRR, SoRR	*rank*	1	0.8959	0.9004	**0.8983**	0.0164	0.0559	**0.0194**	99.98%
	utility		0.9215	0.9730	0.9848	0.0159	0.0519	**0.0171**	99.96%
	random		0.9715	1.1870	1.1974	0.0172	0.0663	0.0295	98.98%
BruGroup, SoGroup	*rank*	1	0.3121	0.3274	0.3282	0.1410	0.1649	**0.1474**	99.92%
		4	0.1855	0.1949	0.1991	0.0719	0.0895	0.0806	97.41%

Table 9: Comparison overview of the related work.

Paper	Sequence-Aware	Algorithm		Time Granularities		Simulated Adoption	*Loss Compensation*
		MF	SQ	FIXED	GROUP		
SoCRATe	✓	✓	✓	✓	✓	✓	✓
Jannach and Ludewig [11]	✓	✓		✓			
Zheleva et al. [37]	✓		✓	✓			
Wang and Zhang [32]					✓		
Moore et al. [16]			✓		✓		
Moling et al. [15]			✓		✓		
Vasile et al. [31]			✓	✓			
Wu et al. [35]			✓	✓			
Sánchez and Bellogín [24]	✓	✓	✓	✓			
Hazrati and Ricci [9]		✓		✓		✓	
Fleder and Hosanagar [8]		✓		✓		✓	
Szlávik et al. [30]		✓		✓		✓	

Algorithm: MF: matrix-factorization, SQ: sequence learning; **Time granularity**: FIXED or based on user groups.

On Sequence-Aware Recommendations. SoCRATe is sequence-aware as it combines principles of sequence-based and session-based recommendations [21]. Sequence-based treats the user-item interactions as a dynamic sequence and takes the sequential dependencies into account to capture the current and recent preference of a user [5, 34]. Session-based captures multiple user-item iterations that happen together in a continuous period of time (session). It is able to capture both a user's short-term preference from recent sessions and the preference dynamics reflecting the change of preferences from one session to another [33]. In [11] the authors detected in the e-commerce domain that recommending on-sale items and currently trending items can lead to higher adoption rates of the recommendations in this domain; in [37] the scope is to find clusters of users in the music environment, specifically, they adapted the model to capture the user mood in listening sessions; finally, in [24], the authors developed a new

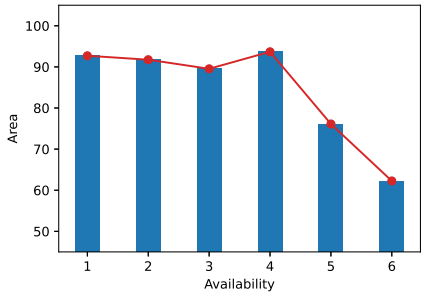


Figure 5: SoRR performance varying the availability of items.

framework tailored to venue recommendation, applied to any dataset containing temporal or sequential information, specifically it uses different item ranking strategies and new evaluation metrics. Finally, in [31, 35] the systems provide consecutive recommendations in a cold-start context. *These systems are different from ours because they are either sequence-based or session-based* as indicated in Table 9.

On Time Granularity and Weight Update. In [32], there is a similarity with SoCRATE regarding the time granularity, specifically the user-group one. This work introduces the concept of recommending an item at a specific time when the user is ready to adopt it. In [15, 16] the system adapts in real-time to the changes in user preferences. *Their approach is similar to our weight update. However, these systems are true RS algorithms, while our system is not dependent on a specific RS algorithm.* Perhaps the closest work to ours is [6], which models the idea of limited availability in an online matching problem (applied in a recommender system). The scoring function combines relevance and diversity, and time granularity is determined by different user arrival rates. *The main difference with our work is that our compensation is among users, instead, their idea of diversity is among items recommended to a single user.*

On Fair Recommendations. [27–29] address the problem of a group of users with potentially different preferences, such as a group trying to decide which movie to watch. The proposed solutions are sequence-aware recommendation systems that take into account users' satisfaction with the recommended items and their disagreement with the rest of the group. In contrast, SoCRATE does not view users as a group to be satisfied; rather, it allows users to receive different items instead of forcing them to receive the same one (individual recommendation). Additionally, our solution focuses on a limited-availability scenario, contrary to the contexts of [27–29]. While the concept of group satisfaction and dissatisfaction is similar to the idea of loss, their computation is specific to group and user recommendations, whereas loss computation calculates the difference between the individual's optimal and actual recommendations.

6 CONCLUSION

We formalized the problem of generating recommendations that compensate users over time in multi-session recommendations.

SoCRATe is built to be usable with any recommender system under limited item availability. In this scenario, few users are recommended some items and receive preferential treatment, while the rest is left with sub-optimal recommendations, ultimately leading them to leave. For this reason, first we identified two measures to be minimized (*average cumulative loss* $\overline{\mathcal{L}}$ and *standard deviation of cumulative loss* $\overline{\mathcal{S}}$), in order to avoid inequality and favor user retention. Then, we designed two compensation strategies (*preference-driven* and *round-robin*) to enforce equal user treatment over time, considering users and items in different orders and simulating different recommendation adoption models and time granularities. Finally, we introduced SoCRATe, a framework that implements all these algorithms to enable user loss compensation. Experiments on real data showed that we are able to compensate loss under different conditions, suggesting that traditional recommenders should be revisited. Additionally, experiments on synthetic data demonstrated that our approach is way faster than an optimal brute-force compensation strategy, while achieving comparable results.

One limitation of our system is that we compensate users "outside" of the individual recommender. This has the benefit of generality, but by including compensation inside a recommendation strategy we could improve performance in terms of accuracy and fairness. Another limitation is using real datasets of limited size, which, although sufficient to substantiate our claims, may not show all compensation opportunities that a more thorough investigation with larger datasets could offer.

In future work, we shall therefore consider larger datasets, as well as studying the system by varying item availabilities over time, e.g., by injecting new items or increasing their availability. Furthermore, we will complement our system evaluation with traditional metrics of recommendation, such as recall and precision. In fact, available datasets lack the required labelling regarding availability, making this task highly complex. We plan to extend our work to involve, e.g., a user study, in order to enrich our experiments with a further evaluation based on recall and precision.

Additionally, we would like to investigate how the solution compares to incorporating compensation inside a particular recommendation algorithm. To do that, we would like to (i) blend short-term user feedback with long-term user history, and (ii) include compensation inside a recommendation strategy, in our case KNN. We would like to build on the work of [14] that adds temporal dynamics inside collaborative filtering. This would allow us to keep track of the evolution of loss in the recommendation logic itself, which may yield higher accuracy results since a recommender has access to a larger pool of items. Comparing this strategy to SoCRATe would enable us to verify this hypothesis and open new research directions for exploring the extension of other recommenders.

ACKNOWLEDGMENTS

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094_P2.

REFERENCES

[1] 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 & Knowledge Management*. 4793–4797.

[2] 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)*.

[3] 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

[4] Chandra Chekuri and Sanjeev Khanna. 2006. A polynomial time approximation scheme for the multiple knapsack problem. *Society for Industrial and Applied Mathematics Journal on Computing* 38, 3 (2006).

[5] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. 2018. Sequential recommendation with user memory networks. In *Proceedings of the eleventh ACM international conference on web search and data mining*. 108–116.

[6] John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. 2019. Balancing Relevance and Diversity in Online Bipartite Matching via Submodularity. In *The Thirty-Third Conference on Artificial Intelligence, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI Press, 1877–1884.

[7] Hui Fang, Danning Zhang, Yiheng Shu, and Guibing Guo. 2020. Deep Learning for Sequential Recommendation: Algorithms, Influential Factors, and Evaluations. *ACM Trans. Inf. Syst.* 39, 1 (2020), 10:1–10:42.

[8] Daniel Fleder and Kartik Hosanagar. 2009. Blockbuster Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity. *Manage. Sci.* 55, 5 (may 2009), 697–712. https://doi.org/10.1287/mnsc.1080.0974

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

[10] 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.

[11] Dietmar Jannach and Malte Ludewig. 2017. Determining Characteristics of Successful Recommendations from Log Data: A Case Study. In *Proceedings of the Symposium on Applied Computing* (Marrakech, Morocco) *(SAC '17)*. ACM, New York, NY, USA, 1643–1648. https://doi.org/10.1145/3019612.3019757

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

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

[14] Yehuda Koren. 2009. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining*, John F. Elder IV, Françoise Fogelman-Soulié, Peter A. Flach, and Mohammed Javeed Zaki (Eds.). ACM, 447–456.

[15] Omar Moling, Linas Baltrunas, and Francesco Ricci. 2012. Optimal Radio Channel Recommendations with Explicit and Implicit Feedback. In *Proceedings of the Sixth ACM Conference on Recommender Systems* (Dublin, Ireland) *(RecSys '12)*. ACM, New York, NY, USA, 75–82. https://doi.org/10.1145/2365952.2365971

[16] Joshua L Moore, Shuo Chen, Douglas Turnbull, and Thorsten Joachims. 2013. Taste Over Time: The Temporal Dynamics of User Preferences.. In *ISMIR*, Vol. 13. 401–406.

[17] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 188–197.

[18] 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.

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

[20] Julien Pilourdault, Sihem Amer-Yahia, Senjuti Basu Roy, and Dongwon Lee. 2018. Task Relevance and Diversity as Worker Motivation in Crowdsourcing. In *IEEE ICDE*. 365–376.

[21] Massimo Quadrana, Paolo Cremonesi, and Dietmar Jannach. 2018. Sequence-Aware Recommender Systems. *ACM Comput. Surv.* 51, 4 (2018), 66:1–66:36. https://doi.org/10.1145/3190616

[22] 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.

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

[24] Pablo Sánchez and Alejandro Bellogín. 2020. Applying reranking strategies to route recommendation using sequence-aware evaluation. *User Modeling and User-Adapted Interaction* 30, 4 (2020), 659–725.

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

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

[27] Maria Stratigi, Jyrki Nummenmaa, Evaggelia Pitoura, and Kostas Stefanidis. 2020. Fair sequential group recommendations. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 1443–1452.

[28] Maria Stratigi, Evaggelia Pitoura, Jyrki Nummenmaa, and Kostas Stefanidis. 2022. Sequential group recommendations based on satisfaction and disagreement scores. *Journal of Intelligent Information Systems* (2022), 1–28.

[29] Maria Stratigi, Evaggelia Pitoura, and Kostas Stefanidis. 2023. SQUIRREL: A framework for sequential group recommendations through reinforcement learning. *Information Systems* 112 (2023), 1021–1028.

[30] Zoltán Szlávik, Wojtek Kowalczyk, and Martijn Schut. 2011. Diversity measurement of recommender systems under different user choice models. In *Proceedings of the International AAAI Conference on Web and Social Media*, Vol. 5. 369–376.

[31] Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2Vec: Product Embeddings Using Side-Information for Recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems* (Boston, Massachusetts, USA) *(RecSys '16)*. ACM, New York, NY, USA, 225–232. https://doi.org/10.1145/2959100.2959160

[32] Jian Wang and Yi Zhang. 2013. Opportunity model for e-commerce recommendation: right product; right time. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*. 303–312.

[33] Shoujin Wang, Longbing Cao, Yan Wang, Quan Z. Sheng, Mehmet A. Orgun, and Defu Lian. 2022. A Survey on Session-based Recommender Systems. *ACM Comput. Surv.* 54, 7 (2022), 154:1–154:38.

[34] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet A. Orgun. 2020. Sequential Recommender Systems: Challenges, Progress and Prospects. *CoRR* abs/2001.04830 (2020).

[35] Xiang Wu, Qi Liu, Enhong Chen, Liang He, Jingsong Lv, Can Cao, and Guoping Hu. 2013. Personalized Next-Song Recommendation in Online Karaokes. In *Proceedings of the 7th ACM Conference on Recommender Systems* (Hong Kong, China) *(RecSys '13)*. ACM, New York, NY, USA, 137–140. https://doi.org/10.1145/2507157.2507215

[36] 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).

[37] Elena Zheleva, John Guiver, Eduarda Mendes Rodrigues, and Nataša Milić-Frayling. 2010. Statistical Models of Music-Listening Sessions in Social Media. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. ACM, New York, NY, USA, 1019–1028. https://doi.org/10.1145/1772690.1772794