Learning User Preferences to Incentivize Exploration in the Sharing Economy

Christoph Hirnschall
ETH Zurich
Zurich, Switzerland
chirnsch@gmail.com

Adish Singla
MPI-SWS
Saarbrücken, Germany
adishs@mpi-sws.org

Sebastian Tschiatschek
Microsoft Research*
Cambridge, United Kingdom
setschia@microsoft.com

Andreas Krause
ETH Zurich
Zurich, Switzerland
krausea@ethz.ch

Abstract

We study platforms in the sharing economy and discuss the need for incentivizing users to explore options that otherwise would not be chosen. For instance, rental platforms such as Airbnb typically rely on customer reviews to provide users with relevant information about different options. Yet, often a large fraction of options does not have any reviews available. Such options are frequently neglected as viable choices, and in turn are unlikely to be evaluated, creating a vicious cycle. Platforms can engage users to deviate from their preferred choice by offering monetary incentives for choosing a different option instead. To efficiently learn the optimal incentives to offer, we consider structural information in user preferences and introduce a novel algorithm - Coordinated Online Learning (CoOL) - for learning with structural information modeled as convex constraints. We provide formal guarantees on the performance of our algorithm and test the viability of our approach in a user study with data of apartments on Airbnb. Our findings suggest that our approach is well-suited to learn appropriate incentives and increase exploration on the investigated platform.

Introduction

In recent years, numerous sharing economy platforms with a variety of goods and services have emerged. These platforms are shaped by users that primarily act in their own interest to maximize their utility. However, such behavior might interfere with the usefulness of the platforms. For example, users of mobility sharing systems typically prefer to drop off rentals at the location in closest proximity, while a more balanced distribution would allow the mobility sharing service to operate more efficiently.

Undesirable user behavior in the sharing economy is in many cases even self-reinforcing. For example, users in the apartment rental marketplace Airbnb are less likely to select infrequently reviewed apartments and are therefore unlikely to provide reviews for these apartments (Fradkin 2014). This is also reflected in the distribution of reviews, where in many cities 20% of apartments account for more than 80% of customer reviews[1].

Such dynamics create a need for platforms in the sharing economy to actively engage users to shape demand and improve efficiency. Several previous papers have proposed the idea of using monetary incentives to encourage desirable behavior in such systems. One example is (Frazier et al. 2014), who studied the problem in a multi-armed bandit setting, where a principal (*e.g.* a marketplace) attempts to maximize utility by incentivizing agents to explore arms other than the myopically preferred one. In their setting, the optimal amount is known to the system, and the main goal is to quantify the required payments to achieve an optimal policy with myopic agents. The idea of shaping demand through monetary incentives in the sharing economy has also been tested in practice. For example, (Singla et al. 2015) use monetary incentives to encourage users of bike sharing systems to return bikes at beneficial locations, making automatic offers through the bike sharing app.

In this context, an important question is what amounts a platform should offer to maximize its utility. (Singla et al. 2015) introduce a simple protocol for learning optimal incentives in the bike sharing system to make users switch from the preferred station to a more beneficial one, ignoring information about specific switches and additional context. Extending on these ideas, we explore a general online learning protocol for efficiently learning optimal incentives.

Our Contributions

We provide the following main contributions in this paper:

- **Structural information:** We consider structural information in user preferences to speed up learning of incentives, and provide a general framework to model structure across tasks via convex constraints. Our algorithm, Coordinated Online Learning (CoOL) is also of interest for related multi-task learning problems.
- **Computational efficiency:** We introduce two novel ideas of *sporadic* and *approximate* projections to increase the computational efficiency of our algorithm. We derive formal guarantees on the performance of the CoOL algorithm and achieve no-regret bounds in this setting.
- **User study on Airbnb:** We collect a unique data set through a user study with apartments on Airbnb and test the viability and benefit of the CoOL algorithm on this dataset.

*Work performed while at ETH Zurich.

[1] Data from `insideairbnb.com`.

Preliminaries

In the following, we introduce the general problem setting of this paper.

Platform. We investigate a general platform in the sharing economy, such as the apartment rental marketplace Airbnb. On this platform, users can choose from n goods and services, denoted as items. A user that arrives at time t chooses an item $i^t \in [n]$. If the user chooses to buy item i^t, the platform gains utility u_i^t.

Incentivizing exploration. The initial choice, item i^t, might not maximize the platform's utility, and the platform might be interested in offering a different item j^t with utility $u_j^t > u_i^t$ instead. For example, j could represent an infrequently reviewed item that the platform wants to explore. To motivate the user to select item j^t instead, the platform can offer an incentive p^t, for example in the form of a monetary discount on that item. The user can either accept or reject the offer p^t depending on the private cost c^t, where the user accepts the offer if $p^t \geq c^t$ and rejects the offer otherwise. If the user accepts the offer, the utility gain of the platform is $u_j^t - u_i^t - p^t$.

Objective. In this setting, two tasks need to be optimized to achieve a high utility gain: finding good switches $i \to j$, and finding good incentives p^t. Good switches $i \to j$ are those, in which the achievable utility gain is positive, i.e. $u_j^t - u_i^t - c^t > 0$. To realize a positive utility gain, the offer p^t needs to be greater or equal to c^t, since otherwise the offer would be rejected.

In this paper, we focus on learning optimal incentives p^t over time, while the platform chooses relevant switches $i \to j$ independently.

Methodology

In this section, we present our methology for learning optimal incentives $p_{i,j}^t$ and start with a single pair of items (i, j). We allow for natural constraints on $p_{i,j}^t$, such that $p_{i,j}^t \in S_{i,j}$, where $S_{i,j}$ is convex and non-empty. For example, $S_{i,j}$ might be lower-bounded by 0 and upper-bounded by the maximum discount that the platform is willing to offer.

Single Pair of Items

We consider the popular algorithmic framework of online convex programming (OCP) (Zinkevich 2003) to learn optimal incentives $p_{i,j}^t$ for a single pair of items. The OCP algorithm is a gradient-descent style algorithm that updates with an adaptive learning rate and performs a projection after every gradient step to maintain feasibility within the constraints $S_{i,j}$. We use $\tau_{i,j}^t$ to denote the number of times a pair of items i, j has been observed and η to denote the learning rate. To measure the performance of the algorithm, we use the loss $l^t(p_{i,j}^t)$, which is the difference between the optimal prediction and the prediction provided by the algorithm, such that $l^t(p^t) = \mathbb{1}_{\{p^t \geq c^t\}} \cdot (p^t - c^t) + \mathbb{1}_{\{p^t < c^t\}} \cdot (u - c^t)$ for $u \geq c^t$, and $l^t(p^t) = 0$ for $u < c^t$[2].

[2]Note that this loss function is non-convex. A convex version is presented in the case study.

Algorithm 1: OL – Online Learning

Input:
- Learning rate constant: $\eta > 0$

Initialize: $p_{i,j}^0 \in S$, $\tau_{i,j}^0 = 0$

for $t = 1, 2, \dots, T$ **do**
- Suffer loss $l^t(p_{i,j}^t)$
- Calculate gradient $g_{i,j}^t$
- Set $\tau_{i,j}^t = \tau_{i,j}^{t-1} + 1$
- Update $p_{i,j}^{t+1} = p_{i,j}^t - \frac{\eta}{\sqrt{\tau_{i,j}^t}} g_{i,j}^t$

end

The algorithm, Online Learning (OL), for a single pair of items is shown in Algorithm 1. The regret, which measures the difference in losses against any (constant) competing weight vector $p_{i,j} \in S$ after T rounds, is defined as

$$Regret_{OL_{i,j}}(T, p_{i,j}) = \sum_{t=1}^{T} \Big(l^t(p_{i,j}^t) - l^t(p_{i,j}) \Big). \quad (1)$$

Multiple Pairs of Items

We now relax the assumption of a fixed pair of items and return to our original problem of learning optimal incentives for multiple pairs of items, *i.e.* the algorithm receives specific items i^t and j^t as input for each user. If we consider all items n on the platform, the total number of pairs is $n^2 - n$.

For learning the optimal incentive for each pair of items, the algorithm maintains a specific learning rate proportional to $1/\sqrt{\tau_{i,j}^t}$ for each pair of items and performs one gradient update step using Algorithm 1. We refer to this straightforward adaptation of the OL algorithm as Independent Online Learning (IOL) and use this algorithm as a baseline for our analysis. Using regret bounds of (Zinkevich 2003) and denoting the number of pairs of items as K, we can upper bound the regret of IOL as

$$Regret_{IOL}(T) \leq \frac{3}{2}\sqrt{TK}\, \|S_{max}\|\, \|\boldsymbol{g}_{max}\|\,. \quad (2)$$

Structural Information

In a real-world setting, incentives for different pairs of items typically are not independent, and in some cases, certain structural information may help to speed up learning of optimal incentives. In the following, we discuss several relevant types of structural information.

Independent learning. In this baseline setting each pair of items is learned individually. Thus, the number of incentives that need to be learned grows quadratically with the number of items on a platform. While applicable for a small number of items, this approach is not favorable on typical platforms in the sharing economy.

Shared learning. Another commonly studied setting is shared learning. In this setting, all pairs of items are considered equivalent, and only one global incentive is learned. While allowing the platform to learn about many pairs of items at the same time, this approach fails to consider natural asymmetries in the problem. For example, the required incentive for switching from i to j is often different than the required incentive for switching from j to i, as can be also observed in the case study of this paper.

Metric/hemimetric structure. Assuming that the required incentives are related to the dissimilarity of items i and j, metrics are a natural choice to model structural dependencies, as they capture the property of triangle inequalities in dissimilarity functions. However, incentives for pairs of items are not necessarily required to be symmetric. For example, the required incentives for switching from a highly reviewed apartment on Airbnb to one without reviews is likely higher than vice versa. Therefore, we use hemimetrics, which are a relaxed form of a metric that satisfy only non-negativity constraints and triangular inequalities, capturing asymmetries in preferences (*cf.* (Singla, Tschiatschek, and Krause 2016)). The usefulness of the hemimetric structure for learning optimal incentives is demonstrated in the experiments.

In the following section, we introduce a general-purpose algorithm for learning with structural information, where the structure is defined by convex constraints on the solution space. The key idea of our algorithm is to coordinate between individual pairs of items by projecting onto the resulting convex set. We generalize our approach for contextual learning, where additional features, such as information about users, may be available. Since projecting onto convex sets may be computationally expensive, we further extend our analysis to allow projections to be sporadic (i.e. only after certain gradient steps) and approximate (i.e. with some error compared to the optimal projection).

Learning with Structural Information

We begin this section by introducing a general framework for specifying structural information via convex constraints. We denote each pair of items as a distinct problem $z \in [K]$, where K is the total number of pairs of items. Each problem z may be associated with additional features, for example with information about the current user. As is common in online learning, we consider a d dimensional weight vector $\boldsymbol{w}_z$ for each problem $z \in [K]$ for learning optimal incentives. The prediction p_z is equal to the inner product between $\boldsymbol{w}_z$ and the d dimensional feature vector. In the previous section, we described the special case with $d = 1$ and a unit feature vector, such that $\boldsymbol{w}_z$ is equivalent to the prediction p_z.

Specifying Structure via Convex Constraints

Similar to constraints on p_z, we allow for convex constraints on $\boldsymbol{w}_z$, such that $\boldsymbol{w}_z \in S_z \subseteq \mathbb{R}^d$. We assume S_z is a convex, non-empty, and compact set, where $\|S_z\|$ is the Euclidean norm of the solution space[3]. Further, we assume $\|S_z\| \leq \|S_{\text{max}}\|$ for some constant $\|S_{\text{max}}\|$. We denote the joint solution space of the K problems as $S = S_1 \times \cdots \times S_z \times \cdots \times S_K \subseteq \mathbb{R}^{d \cdot K}$ and define $\boldsymbol{w}^t \in S$ as the concatenation of the problem specific weight vectors at time t, *i.e.*

$$\boldsymbol{w}^t = \left[(\boldsymbol{w}_1^t)' \cdots (\boldsymbol{w}_z^t)' \cdots (\boldsymbol{w}_K^t)'\right]'.$$

The available structural information is modelled by a set of convex constraints, such that the joint competing weight vector $\boldsymbol{w}^*$, against which the loss at each round is measured, lies in a convex, non-empty, and closed set $S^* \subseteq S$, representing a restricted joint solution space, *i.e.* $\boldsymbol{w}^* \in S^*$. In the following, we provide several practical examples of how S^* can be defined.

Independent learning. $S^* \equiv S$ models the setting where the problems are unrelated/independent.

Shared learning. A shared parameter setting can be modeled as

$$S^* = \{\boldsymbol{w}^* \in S \mid \boldsymbol{w}_1^* = \cdots = \boldsymbol{w}_z^* = \cdots = \boldsymbol{w}_K^*\}.$$

Instead of sharing all parameters, another common scenario is to share only a few parameters. For a given $d' \leq d$, sharing d' parameters across the problems can be modeled as

$$S^* = \{\boldsymbol{w}^* \in S \mid \boldsymbol{w}_1^*[1\!:\!d'] \cdots = \boldsymbol{w}_z^*[1\!:\!d'] = \cdots \boldsymbol{w}_K^*[1\!:\!d']\}$$

where $\boldsymbol{w}_z^*[1:d']$ denotes the first d' entries in $\boldsymbol{w}_z^*$. This approach is useful for sharing certain parameters that do not depend on the specific problem. For example, in the case of apartments on Airbnb, a shared feature could be the distance between apartments.

Hemimetric structure. To model dissimilarities between items for learning optimal incentives, we use the hemimetric set. Specifically, we use r-bounded hemimetrics, which, next to non-negativity constraints and triangular inequalities, also include non-negativity and upper bound constraints. For $d = 1$, the convex set representing r-bounded hemimetrics is given by $S^* =$

$$\{\boldsymbol{w}^* \in S \mid \boldsymbol{w}_{i,j}^* \in [0, r], \boldsymbol{w}_{i,j}^* \leq \boldsymbol{w}_{i,k}^* + \boldsymbol{w}_{k,j}^* \; \forall i, j, k \in [n]\}$$

Our Algorithm

In the following, we introduce our algorithm, Coordinated Online Learning (CoOL).

Exploiting Structure via Weighted Projections. The CoOL algorithm exploits structural information in a principled way by performing weighted projections to S^*, with weights for a problem z proportional to $\sqrt{\tau_z^t}$. Intuitively, the weights allow us to learn about problems that have been observed infrequently while avoiding to "unlearn" problems that have been observed more frequently. A formal justification for using weighted projections is provided in the extended version of this paper (Hirnschall et al. 2018).

We define Q^t as a square diagonal matrix of size dK with each $\sqrt{\tau_z^t}$ represented d times. In the one-dimensional case ($d = 1$), we can write Q^t as

$$\boldsymbol{Q}^t = \begin{bmatrix} \sqrt{\tau_1^t} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\tau_K^t} \end{bmatrix}. \tag{3}$$

Using $\widetilde{\boldsymbol{w}}$ to jointly represent the current weight vectors of all the learners at time t (*cf.* Line 7 in Algorithm 2), we

[3] Euclidean norm is used throughout, unless otherwise specified.

Algorithm 2: CoOL – Coordinated Online Learning

1 **Input:**
- Projection steps: $(\xi^t)_{t\in[T]}$ where $\xi^t \in \{0,1\}$
- Projection accuracy: $(\delta^t)_{t\in[T]}$ where $\delta^t \geq 0$
- Learning rate constant: $\eta > 0$

2 **Initialize:** $\boldsymbol{w}_z^1 \in S_z$, $\tau_z^0 = 0$
3 **for** $t = 1, 2, \ldots, T$ **do**
4 Suffer loss $l^t(\boldsymbol{w}_z^t)$
5 Calculate (sub-)gradient $\boldsymbol{g}_z^t$
6 Set $\tau_z^t = \tau_z^{t-1} + 1$
7 Update $\widetilde{\boldsymbol{w}}^{t+1} = \boldsymbol{w}^t$; $\widetilde{\boldsymbol{w}}_z^{t+1} = \boldsymbol{w}_z^t - \frac{\eta}{\sqrt{\tau_z^t}}\boldsymbol{g}_z^t$
8 **if** $\xi^t = 1$ **then**
9 Define $\boldsymbol{Q}^t$ as per Equation (3)
10 Compute $\boldsymbol{w}^{t+1} = AProj(\widetilde{\boldsymbol{w}}^{t+1}, \delta^t, \boldsymbol{Q}^t)$
else
11 $\boldsymbol{w}_z^{t+1} = \underset{\boldsymbol{w}\in S_z}{\operatorname{argmin}} \|\boldsymbol{w} - \widetilde{\boldsymbol{w}}_z^{t+1}\|_2$
end
end

compute the new joint weight vector $\boldsymbol{w}^{t+1}$ (*cf.* Line 10 in Algorithm 2) by projecting onto S^*, using

$$\boldsymbol{w}^{t+1} = \underset{\boldsymbol{w}\in S^*}{\operatorname{argmin}} (\boldsymbol{w} - \widetilde{\boldsymbol{w}})' \boldsymbol{Q}^t (\boldsymbol{w} - \widetilde{\boldsymbol{w}}). \tag{4}$$

We refer to this as the weighted projection onto S^*. Since S^* is convex and the projection is a special case of the Bregman projection, the projection onto S^* is unique (*cf.* (Cesa-Bianchi and Lugosi 2006; Rakhlin and Tewari 2009)).

Sporadic and Approximate Projections. For large scale applications (*i.e.* large K or large d), projecting at every step could be computationally very expensive: a projection onto a generic convex set S^* would require solving a quadratic program of dimension dK. To allow for computationally efficient updates, we introduce two novel algorithmic ideas: *sporadic* and *approximate* projections, defined by the above-mentioned sequences $(\xi^t)_{t\in[T]}$ and $(\delta^t)_{t\in[T]}$. Here, δ^t denotes the desired accuracy at time t and is given as input to Function 3, AProj, for computing approximate projections. This way, the accuracy can be efficiently controlled using the duality gap of the projections. As we shall see in our experimental results, these two algorithmic ideas of sporadic and approximate projections allow us to speed up the algorithm by an order of magnitude while retaining the improvements obtained through the projections.

Algorithm 2, when invoked with $\xi^t = 1, \delta^t = 0\ \forall t \in [T]$, corresponds to a variant of our algorithm with exact projections at every time step. When invoked with $\xi^t = 0\ \forall t \in [T]$, our algorithm corresponds to the IOL baseline.

Relation to existing approaches. A related algorithm is the AdaGrad algorithm (Duchi, Hazan, and Singer 2011), which uses the sum of the magnitudes of past gradients to determine the learning rate at each time t, where larger past gradients correspond to smaller learning rates. A key difference to the CoOL algorithm is that the AdaGrad algorithm

Function 3: AProj – Approximate Projection

Input: $\widetilde{\boldsymbol{w}}, \delta^t, \boldsymbol{Q}^t$
Define $f^t(\boldsymbol{w}) = (\boldsymbol{w} - \widetilde{\boldsymbol{w}})' \boldsymbol{Q}^t (\boldsymbol{w} - \widetilde{\boldsymbol{w}})$ for $\boldsymbol{w} \in S$
Choose
$\boldsymbol{w}^{t+1} \in \{\boldsymbol{w} \in S^* : f^t(\boldsymbol{w}) - \min_{\boldsymbol{w}'\in S^*} f^t(\boldsymbol{w}') \leq \delta^t\}$
Return: $\boldsymbol{w}^{t+1}$

enforces exact projections after every iteration. This is particularly problematic for large, complex structures since projections on these structures often rely on numeric approximations, that may not guarantee to converge to the exact solution in finite time.

Performance Guarantees and Analysis

In this section, we analyze worst-case regret bounds of the CoOL algorithm against a competing weight vector $\boldsymbol{w}^* \in S^*$. The proofs are provided in the extended version of this paper (Hirnschall et al. 2018).

General Bounds

We begin with a general result, without assumptions on the projection accuracy and rate.

Theorem 1. *The regret of the CoOL algorithm is bounded by* $Regret_{CoOL}(T) \leq$

$$\frac{1}{2\eta}\|S_{max}\|^2\sqrt{TK} + 2\eta\|\boldsymbol{g}_{max}\|^2\sqrt{TK} \tag{R1}$$

$$+ \sum_{t=1}^{T} \mathbb{1}_{\{(\neg\xi^{t-1})\wedge(\xi^t)\}}\|S_{max}\|\|\boldsymbol{g}_{max}\| \tag{R2}$$

$$+ \frac{1}{\eta}\sum_{t=1}^{T}\mathbb{1}_{\{\xi^t\}}\left(\delta^t + \sqrt{2\delta^t}(tK)^{1/4}\|S_{max}\|\right) \tag{R3}$$

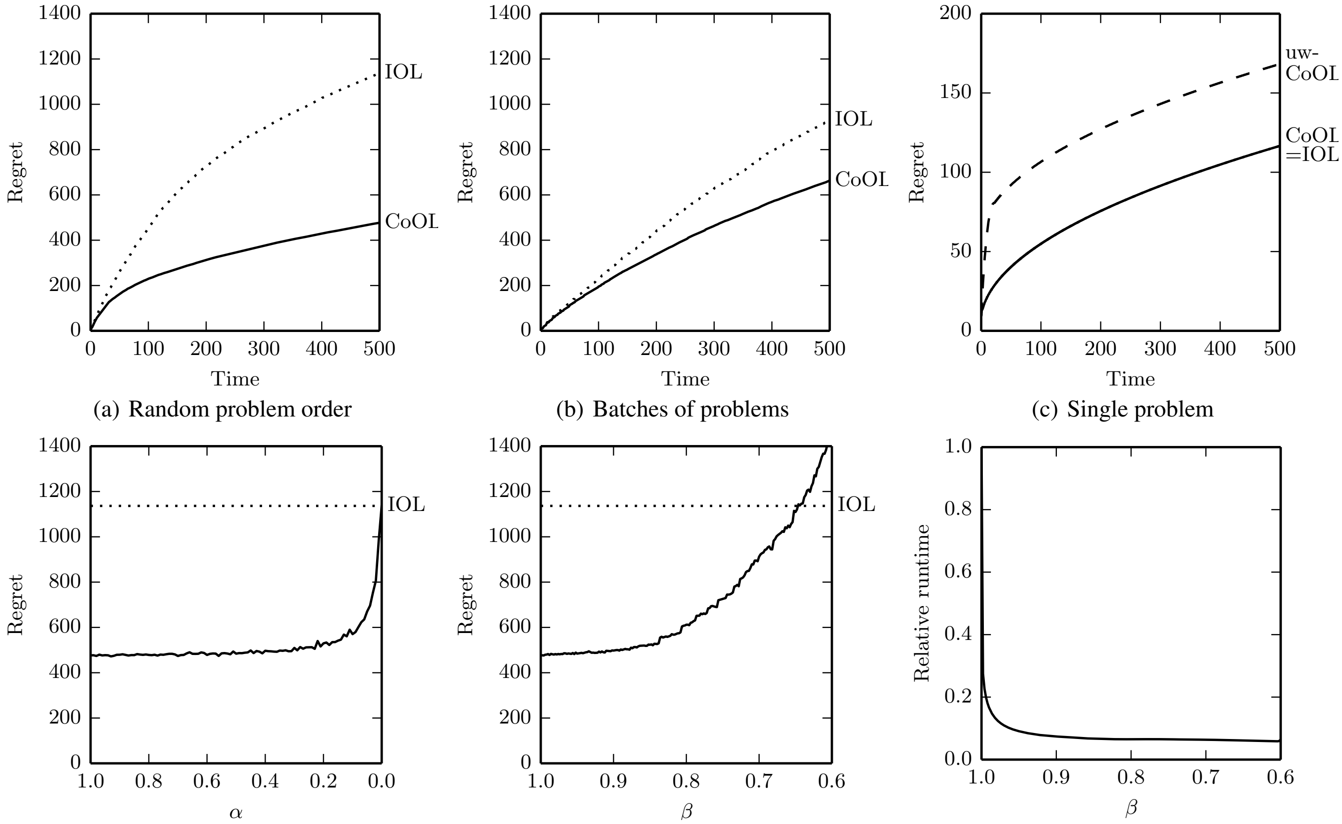
$$+ \frac{1}{2\eta}\|S_{max}\|^2 - 2\eta\|\boldsymbol{g}_{max}\|^2 K. \tag{R4}$$

The regret in Theorem 1 has four components. R1 comes from the standard regret analysis in the OCP framework, R2 comes from sporadic projections, R3 comes from the allowed error in the projections, and R4 is a constant.

Note that when $\xi^t = 0$ for all t (*i.e.* no projections are performed) and η is proportional to $1/\sqrt{\tau_{i,j}^t}$, we get the same regret bounds proportional to $\sqrt{T}$ as for the IOL algorithm. This also reveals the worst-case nature of the regret bounds of Theorem 1, *i.e.* the proven bounds for CoOL are agnostic to the specific structure S^* and the order of task instances.

Sporadic/Approximate Projection Bounds

To provide specific bounds for the practically useful setting of sporadic and approximate projections, we introduce α and β and the user chosen parameters c_α and c_β to control the frequency and accuracy of the projections.



(a) Random problem order (b) Batches of problems (c) Single problem

(d) Varying α (e) Varying β (f) Projection runtime varying β

Figure 1: Simulation results for learning hemimetrics. (a,b,c) compare the performance of CoOL against IOL for different orders of problem instances. (d,e,f) show the speed/performance tradeoff using sporadic and approximate projections.

Corollary 1. *Set* $\eta = \frac{1}{2}\frac{\|S_{max}\|}{\|\boldsymbol{g}_{max}\|}$. $\forall t \in [T]$, *define*

$$\xi^t \sim Bernoulli(\alpha) \text{ with } \alpha = \frac{c_\alpha}{\sqrt{T}},$$

$$\delta^t = c_\beta (1-\beta)^2 \frac{\sqrt{K}}{\sqrt{t}} \|S_{max}\|^2$$

where constants $c_\alpha \in [0, \sqrt{T}]$, $c_\beta \geq 0$, *and* $\beta \in [0,1]$. *The expected regret (w.r.t.* $(\xi^t)_{t\in[T]}$*) of the CoOL algorithm is bounded by* $\mathbf{E}\left[Regret_{CoOL}(T)\right] \leq$

$$2\sqrt{TK}\,\|S_{max}\|\,\|\boldsymbol{g}_{max}\| \cdot \left(1 + \frac{c_\alpha}{2\sqrt{K}}\left(1 - \frac{c_\alpha}{\sqrt{T}}\right) + c_\alpha(c_\beta + \sqrt{2c_\beta})(1-\beta)\right).$$

As shown in Corollary 1, projections are required to be more accurate for higher values of t. Intuitively, this is required so that already learned weights are not unlearned through inaccurate projections. Using the definitions under Corollary 1, we can prove worst-case regret bounds proportional to $\sqrt{T}$ for this setting.

Performance Analysis for Hememtric Structure

We now test the performance of the CoOL algorithm on synthetic data with an underlying hemimetric structure.

Hemimetric projection. To be able to perform weighted projections onto the hemimetric polytope, we use the metric nearness algorithm (Sra, Tropp, and Dhillon 2004) as a starting point. For our purposes, three modifications of the algorithm are required: First, we lift the requirement of symmetry to generalize from metrics to hemimetrics. Second, the metric nearness algorithm does not guarantee a solution in the metric set in finite time. However, to calculate the duality gap, the solution is required to be feasible. Thus, we apply the Floyd-Warshall algorithm (Floyd 1962) after every iteration to receive a solution in the hemimetric set. Third, we add weights to the triangle inequalities to allow for weighted projections and further add upper and lower bound constraints.

Data structure. To empirically test the performance of the CoOL algorithm on the hemimetric set, we synthetically generate data with $d = 1$ and model the underlying structure S^* as a set of r-bounded hemimetrics with $n = 10$, resulting in $K = 90$ problems. We use a simple underlying *ground-truth* hemimetric $\boldsymbol{w}^*$, where the n items belong to two equal-sized clusters, with $p^*_{i,j} = 1$ if i and j are from the same cluster and $p^*_{i,j} = 9$ otherwise. The results of our experiment in Figure 1 illustrate the potential runtime improvement using sporadic/approximate projections.

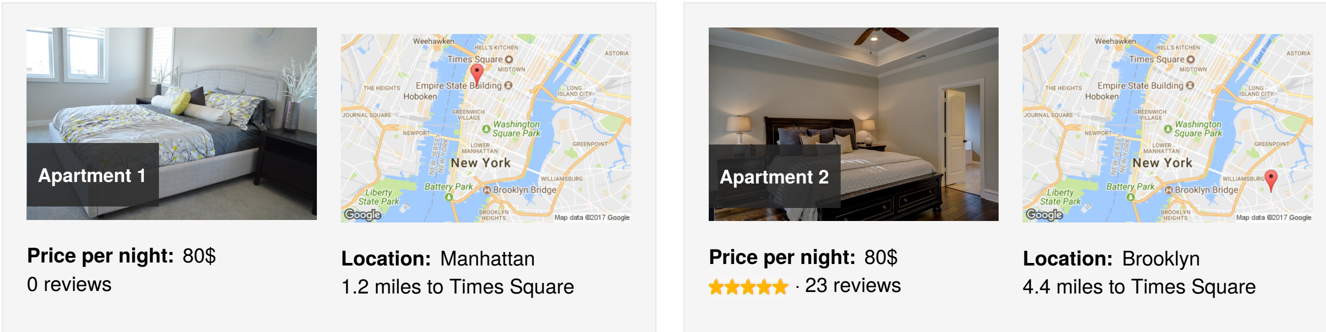


Figure 2: Snapshot of the user survey study on MTurk.[4]

Random order of problems. Problem instances z^t are chosen uniformly at random at every time step. The CoOL algorithm achieves a significantly lower regret than the IOL algorithm, benefiting from the weighted projections onto S^*. At $T = 500$, the regret of CoOL is less than half of that of the IOL, *cf.* Figure 1(a).

Batches of problems. In the batch setting, a problem instance is chosen uniformly at random, then it is repeated five times before choosing a new problem instance. Compared to the above-mentioned random order, the IOL algorithm suffers a lower regret because of a higher probability that problems are repeatedly shown. Furthermore, the benefit of the projections onto S^* for the CoOL algorithm is reduced, *cf.* Figure 1(b), showing that the benefit of the projections depends on the specific order of the problem instances for a given structure.

Single-problem setting. A single problem z is repeated in every round. As illustrated, in this case the IOL algorithm and the CoOL algorithm have the same regret, *cf.* Figure 1(c). In order to get a better understanding of using weights Q^t for the weighted projection, we also show a variant uw-CoOL using Q^t as identity matrix. Unweighted projection or using the wrong weights can hinder the convergence of the learners, as shown in Figure 1(c) for this extreme case of a single-problem setting.

Varying the rate of projection (α). The regret of the CoOL algorithm monotonically increases as α decreases, and is equivalent to the regret of the IOL algorithm at $\alpha = 0$, *cf.* Figure 1(d). In the range of α values between 1 and 0.1, the regret of the CoOL algorithm is relatively constant and increases strongly only as α approaches 0. With α as low as 0.1, the regret of the CoOL algorithm in this setting is still almost half of that of the IOL algorithm.

Varying the accuracy of projection (β). The regret of the CoOL algorithm monotonically increases as β decreases, and exceeds that of the IOL algorithm for values smaller than 0.65 because of high errors in the projections, *cf.* Figure 1(e). In the range of β values between 1 and 0.85, the regret of the CoOL algorithm is relatively constant and less than half of that of the IOL algorithm.

Runtime vs. approximate projections. As expected, the runtime of the projection monotonically decreases as β decreases, *cf.* Figure 1(f). For values of β smaller than 0.95, the runtime of the projection is less than 10% of that of the exact projection. Thus, with β values in the range of 0.85 to 0.95, the CoOL algorithm achieves the best of both worlds: the regret is significantly smaller than that of IOL, with an order of magnitude speed up in the runtime compared to exact projections.

Airbnb Case Study

To test the viability and benefit of the CoOL algorithm in a realistic setting, we conducted a user study with data from the marketplace Airbnb.

Experimental Setup

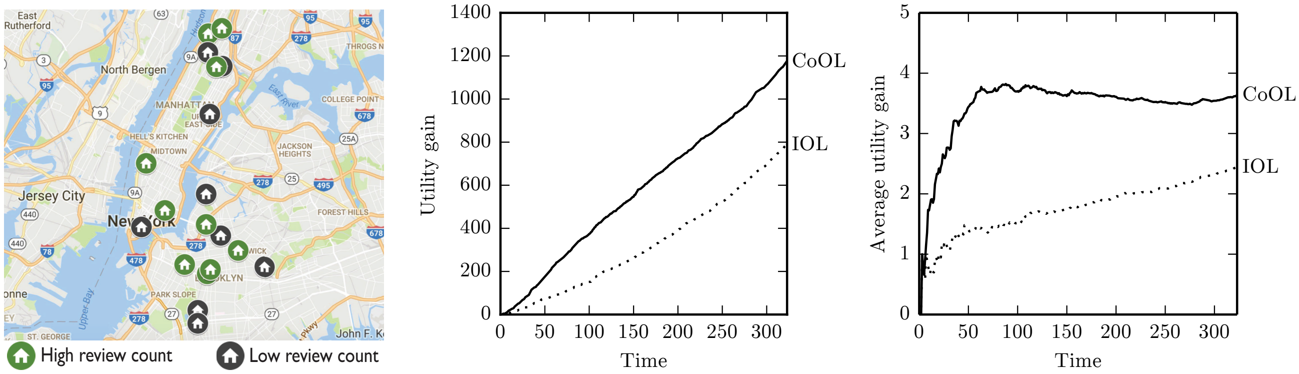
We use the following setup in our user study:

Airbnb dataset. Using data of Airbnb apartments from `insideairbnb.com`, we created a dataset of 20 apartments as follows: we chose apartments from 4 types in New York City by location (Manhattan or Brooklyn) and number of reviews (high, $\geq$ 20 or low, $\leq$ 2). From each type we chose 5 apartments, resulting in a total sample of $n = 20$ apartments.

Survey study on MTurk platform. In order to obtain real-world distributions of the users' private costs, we collected data from Amazon's Mechanical Turk marketplace. After several introductory questions about their preferences and familiarity with travel accommodations, participants were shown two randomly chosen apartments from the Airbnb dataset. To choose between the apartment, participants were given the price, location, picture, number of reviews and rating of each apartment, as shown in Figure 2. Participants were first asked to select their preferred choice between the two apartments. Next, they were asked to specify their private cost for choosing the other, less preferred apartment instead. The collected data from the responses consists of tuples $((i,j),c)$, where i is the preferred choice, j is the suggested alternative, and c is the private cost of the user.

Sample. In total, we received 943 responses, as summarized in Table 1. The sample for the performance analysis of the CoOL algorithm consists of 323 responses, in which i was a frequently reviewed apartment, j an infrequently reviewed apartment, and participants were willing to explore

[4]Real pictures from Airbnb replaced with illustrative examples.



(a) Dataset of 20 apartments (b) Cummulative utility gain (c) Average utility gain

Figure 3: Results of experiments with Airbnb dataset.

the infrequently reviewed apartment for a discount (*i.e.* they did not select NA).

Utility gain. The utility gain u for getting a review for infrequently reviewed apartments is set to $u = 40$ in our experiments, based on referral discounts given by Airbnb in the past.

Loss function. As introduced in the methodology section, we require a convex version of the true loss function for our online learning framework, ideally acting as a surrogate of the true loss. Additionally, the gradient of the loss function needs to be calculated from the binary feedback of acceptance/rejection of the offers. However, in the analyzed model with binary feedback, a loss function that satisfies both requirements cannot be constructed. Instead, we consider a simplified piece-wise linear convex loss function given by $l^t(p^t) = \mathbb{1}_{\{p^t \geq c^t\}} \cdot (p^t - c^t) + \mathbb{1}_{\{p^t < c^t\}} \cdot \frac{u}{\Delta} \cdot (c^t - p^t)$, where $\frac{u}{\Delta}$ denotes the magnitude of the gradient when a user rejects the offer. For the experiment, we use a delta value of 20. Due to this transformation, we use the utility gain rather than the loss as a useful measure of the performance of the CoOL algorithm.

Structure. Due to the small number of apartments, we consider a non-contextual setting with $d = 1$ and use an r-bounded hemimetric structure to model the relationship of the tasks, where r is set to 40 to avoid recommending incentives $p^t > u$. Using a setting with $d > 1$ would allow for real-world applications with additional context.

Main Results

We now present and discuss the results of the user study.

Descriptive statistics. Out of all responses, 758 (80.4%) respondents were willing to accept an offer for their less preferred apartment, given a certain discount per night. Out of these respondents, the average required discount for accepting the alternative apartment was 27.9 USD per night. The average required discount for switching from a frequently reviewed apartment to an infrequently reviewed apartment was 6% higher.

Out of the respondents who could choose between a frequently and an infrequently reviewed apartment, 83.9% respondents chose the frequently reviewed apartment, while only 16.1% respondents chose the infrequently reviewed apartment.

In the responses to an open question about the factors respondents considered to decide on the discount, we captured the frequency at which different factors were mentioned by defining several keywords for each factor. The number of times each factor was mentioned is shown in Table 2.

Algorithm performance. We use the cumulative utility gain to measure the performance of the IOL and the CoOL algorithm. The utility gains of both algorithms after 323 responses are shown in Figure 3(c). The utility gain in Figure 3(b) is almost 50% higher for the CoOL algorithm than for the IOL algorithm. Figure 3(c) reveals that this gain is mainly achieved due to a significant speed up in learning over the first 50 problems.

Discussion. The results of the user study confirm several findings of (Fradkin 2014), who studied the booking behavior on Airbnb. Similar to this study, we find that apartments with a high number of reviews are significantly more likely

i	j	Responses	Accepted	Avg. Discount
High	Low	416	77.6%	29.5$
Low	Low	228	83.3%	28.1$
High	High	219	82.2 %	25.4$
Low	High	80	81.3%	25.9$

Table 1: Responses for different apartment types.

Category	Example keywords	Mentions
Location	neighborhood, distance	477
Reviews	rating, star	309
Price	expensive, cheap	182
Picture	image, photo	169

Table 2: Mentioned categories for deciding on a discount.

to be selected. We also find that the average required discount per night is higher when the alternative choice is an infrequently reviewed apartment. This also points toward a difference in willingness to pay between frequently and infrequently reviewed apartments. Similar results have been found in earlier studies on other marketplaces (Resnick et al. 2006; Ye, Law, and Gu 2009; Luca 2011).

The user study also confirms that incentives influence buying behavior and can help increase exploration on online marketplaces (Avery, Resnick, and Zeckhauser 1999; Robinson, Nightingale, and Mongrain 2012); when respondents chose a frequently reviewed apartment and were asked to instead choose an infrequently reviewed apartment, 77.6% of respondents were willing to accept a sufficiently large offer. More than 10% of those respondents were willing to accept a discount of 10 USD per night or less.

The performance of the IOL and CoOL algorithm in Figures 3(b) and 3(c) suggests that incentives can be learned via online learning, and that structural information can be used to significantly speed up the learning. Further, the speed up in learning directly increases the marketplace's utility gain from suggesting alternative items. To reduce the problem size on a real-world application such as Airbnb, items could be grouped by features such as location or number of reviews. Further, problem-specific features, such as the distance between apartments could be added to increase the accuracy of the prediction.

Related Work

Multi-armed bandit / Bayesian games. A related path of research are multi-armed bandit and Bayesian games, where a principal attempts to coordinate agents to maximize its utility. Research in this area mainly focuses on changing the behavior of agents in the way information is disclosed, rather than through provision of payments. (Kremer, Mansour, and Perry 2014) provide optimal information disclosure policies for deterministic utilities and only two possible actions. (Mansour, Slivkins, and Syrgkanis 2015) generalize the results for stochastic utilities and a constant number of actions. Further, (Mansour et al. 2016) consider the interaction of multiple agents, and (Chakraborty et al. 2017) analyze a multi-armed bandits in the presence of communication costs. Our problem is different to previous research in that utilities are not required to be stochastic, and additional structural information is available to the principal.

Recommender systems. A different approach to encouraging exploration in online marketplaces are recommender systems, which are known to influence buyers' purchasing decisions and can be used to encourage exploration (Resnick and Varian 1997; Senecal and Nantel 2004). For example, ϵ-greedy recommender systems recommend a product closest to the buyer's preferences with probability $(1-\epsilon)$ and a random product with probability ϵ (Ten Hagen, Van Someren, and Hollink 2003). Such recommender systems can be extended using ideas studied in this paper.

Online/distributed multi-task learning. Multi-task learning has been increasingly studied in online and distributed settings recently. Inspired by wearable computing, a recent work by (Jin et al. 2015) studied online multi-task learning in a distributed setting. They considered a setup, where tasks arrive asynchronously, and the relatedness among the tasks is maintained via a correlation matrix. However, there is no theoretical analysis on the regret bounds for the proposed algorithms. (Wang, Kolar, and Srerbo 2016) recently studied the multi-task learning for distributed LASSO with shared support. Their work is different from ours — we consider general convex constraints to model task relationships and consider the adversarial online regret minimization framework.

Conclusions and Future Work

We highlighted the need in the sharing economy to actively shape demand by incentivizing users to differ from their preferred choices and explore different options instead. To learn the incentives users require to choose different items, we developed a novel algorithm, CoOL, which uses structural information in user preferences to speed up learning. The key idea of our algorithm is to exploit structural information in a computationally efficient way by performing *sporadic* and *approximate* projections. We formally derived no-regret bounds for the CoOL algorithm and provided evidence for the increase in performance over the IOL baseline through several experiments. In a user study with apartments from the rental marketplace Airbnb, we demonstrated the practical applicability of our approach in a real-world setting. To conclude, we discuss several additional considerations for offering incentives in a sharing economy platform.

Safety/individual consumer loss. Generally, exploration in the sharing economy may be risky, and individuals can face severe losses while exploring. For example, new hosts might not be trustworthy, and new drivers in ridesharing systems might not be reliable. In our approach, the items to be explored are controlled by the platform, and appropriate preconditions would need to be implemented to minimize risks.

Reliability/Consistency. In order for platforms to implement an algorithmic provision of monetary incentives, it is important that incentives are reliable and consistent over time. Ideally, similar users should receive similar incentives, and offers should be consistent with the user's preferences. Using the CoOL algorithm, consistency can be controlled through appropriate convex constraints.

Strategy-proofness. Providing monetary incentives based on user preferences creates possibilities for opportunistic behavior. For example, users could attempt to repeatedly decline offers to receive higher offers in the future or browse certain items hoping to receive offers for similar items. To control for such behavior, markets need to be large enough so that behavior of individuals does not affect overall learning. Further, platforms can control the number and frequency with which individual users receive offers to minimize opportunistic possibilities.

Acknowledgments

This work was supported in part by the Swiss National Science Foundation, and Nano-Tera.ch program as part of the Opensense II project, ERC StG 307036, and a Microsoft Research Faculty Fellowship. Adish Singla acknowledges support by a Facebook Graduate Fellowship.

References

Avery, C.; Resnick, P.; and Zeckhauser, R. 1999. The market for evaluations. *American Economic Review* 564–584.

Cesa-Bianchi, N., and Lugosi, G. 2006. *Prediction, learning, and games*. Cambridge university press.

Chakraborty, M.; Chua, K. Y. P.; Das, S.; and Juba, B. 2017. Coordinated versus decentralized exploration in multi-agent multi-armed bandits. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 164–170.

Duchi, J.; Hazan, E.; and Singer, Y. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12:2121–2159.

Floyd, R. W. 1962. Algorithm 97: shortest path. *Communications of the ACM* 5(6):345.

Fradkin, A. 2014. Search frictions and the design of online marketplaces. *NBER Working Paper*.

Frazier, P.; Kempe, D.; Kleinberg, J.; and Kleinberg, R. 2014. Incentivizing exploration. In *Proceedings of the fifteenth ACM conference on Economics and computation*, 5–22. ACM.

Hirnschall, C.; Singla, A.; Tschiatschek, S.; and Krause, A. 2018. Learning user preferences to incentivize exploration in the sharing economy (extended version).

Jin, X.; Luo, P.; Zhuang, F.; He, J.; and He, Q. 2015. Collaborating between local and global learning for distributed online multiple tasks. In *CIKM*.

Kremer, I.; Mansour, Y.; and Perry, M. 2014. Implementing the wisdom of the crowd. *Journal of Political Economy* 122(5):988–1012.

Luca, M. 2011. Reviews, reputation, and revenue: The case of yelp. com. *Harvard Business School NOM Unit Working Paper*.

Mansour, Y.; Slivkins, A.; Syrgkanis, V.; and Wu, Z. S. 2016. Bayesian exploration: Incentivizing exploration in bayesian games. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, EC '16, 661–661. New York, NY, USA: ACM.

Mansour, Y.; Slivkins, A.; and Syrgkanis, V. 2015. Bayesian incentive-compatible bandit exploration. In *Proceedings of the Sixteenth ACM Conference on Economics and Computation*, EC '15, 565–582. New York, NY, USA: ACM.

Rakhlin, A., and Tewari, A. 2009. Lecture notes on online learning. *Draft, April*.

Resnick, P., and Varian, H. R. 1997. Recommender systems. *Communications of the ACM* 40(3):56–58.

Resnick, P.; Zeckhauser, R.; Swanson, J.; and Lockwood, K. 2006. The value of reputation on ebay: A controlled experiment. *Experimental economics* 9(2):79–101.

Robinson, J. G.; Nightingale, T. R.; and Mongrain, S. A. 2012. Methods and systems for obtaining reviews for items lacking reviews. US Patent 8,108,255.

Senecal, S., and Nantel, J. 2004. The influence of online product recommendations on consumers online choices. *Journal of retailing* 80(2):159–169.

Singla, A.; Santoni, M.; Bartók, G.; Mukerji, P.; Meenen, M.; and Krause, A. 2015. Incentivizing users for balancing bike sharing systems. In *AAAI*.

Singla, A.; Tschiatschek, S.; and Krause, A. 2016. Actively learning hemimetrics with applications to eliciting user preferences. In *ICML*.

Sra, S.; Tropp, J.; and Dhillon, I. S. 2004. Triangle fixing algorithms for the metric nearness problem. In *Advances in Neural Information Processing Systems*, 361–368.

Ten Hagen, S.; Van Someren, M.; and Hollink, V. 2003. Exploration/exploitation in adaptive recommender systems. *Proceedings of Eunite 2003*.

Wang, J.; Kolar, M.; and Srerbo, N. 2016. Distributed multi-task learning. In *AISTATS*.

Ye, Q.; Law, R.; and Gu, B. 2009. The impact of online user reviews on hotel room sales. *International Journal of Hospitality Management* 28(1):180–182.

Zinkevich, M. 2003. Online convex programming and generalized infinitesimal gradient ascent. In *ICML*.