Online Learning in Large-scale Contextual Recommender Systems

Linqi Song, Cem Tekin, and Mihaela van der Schaar, *Fellow, IEEE*

Abstract—In this paper, we propose a novel large-scale, context-aware recommender system that provides accurate recommendations, scalability to a large number of diverse users and items, differential services, and does not suffer from "cold start" problems. Our proposed recommendation system relies on a novel algorithm which learns online the item preferences of users based on their click behavior, and constructs online item-cluster trees. The recommendations are then made by choosing an item-cluster level and then selecting an item within that cluster as a recommendation for the user. This approach is able to significantly improve the learning speed when the number of users and items is large, while still providing high recommendation accuracy. Each time a user arrives at the website, the system makes a recommendation based on the estimations of item payoffs by exploiting past context arrivals in a neighborhood of the current user's context. It exploits the similarity of contexts to learn how to make better recommendations even when the number and diversity of users and items is large. This also addresses the cold start problem by using the information gained from similar users and items to make recommendations for new users and items. We theoretically prove that the proposed algorithm for item recommendations converges to the optimal item recommendations in the long-run. We also bound the probability of making a suboptimal item recommendation for each user arriving to the system while the system is learning. Experimental results show that our approach outperforms the state-of-the-art algorithms by over 20% in terms of click through rates.

Index Terms—Recommender systems, online learning, clustering algorithms, multi-armed bandit.

1 Introduction

Online websites currently provide a vast number of products or services to their users. Recommender systems have emerged as an essential method to assist users in finding desirable products or services from large sets of available products or services [1] [2]: movies at Netflix, products at Amazon, news articles at Yahoo!, advertisements at Google, reviews in Yelp etc. Moreover, Yahoo! Developer Network, Amazon's Amazon Web Services, Google's Google Developers and numerous other major companies also offer Web services that operate on their data, and recommendation ability is often provided to developers as an essential service functionality.

The products or services provided by the recommender systems are referred to generically as *items* [1] [2]. The performance, or *payoff*, of a recommendation is based on the response of the user to the recommendation. For example, in news recommendations, payoffs are measured by users' click behaviors (e.g., 1 for a click and 0 for no click), and the average payoff when a webpage is recommended is known as the Click-Through Rate (CTR) [6]. The goal of recommendations is to maximize the payoffs over all users that come to the website.

Although recommender systems have been deployed in many application areas during the past decade, the tremendous increase in both the number and diversity of users as well as items poses several key new challenges including dealing with heterogeneous user characteristics and item sets, existence of different user types such as registered and non-registered users, large number of items and users, and the cold start problem. We address all these challenges by designing online learning algorithms with three properties: contextualization, differential services and scalability.

(a) *Contextualization* has the potential to better predict or anticipate the preferences of users. First-generation recommender systems consider the expected payoff of each recommendation to only be a function of the item [4] [6] [8] [9]. However, such recommender systems model the preferences of users with limited accuracy and have thus a limited performance. Several recent works [5] [18] [23] show that incorporating the *context* in which a recommendation is made can greatly improve the accuracy of predicting how users respond to various recommendations. Moreover, contextualization allows the recommender to learn together for groups of users having similar contexts, which significantly speeds up the learning process.

The context is defined generally as an aggregate of various categories that describe the situations or circumstances of the user when a recommendation is made [18] [19], such as time, location, search query, purchase history, etc. When contexts are incorporated into the recommender systems, the expected payoff of a recommendation can be modeled as a function of the item *and* the context. If the number of contexts, X, is

L. Song, T. Cem, and M. van der Schaar are with Department of Electrical Engineering, UCLA. Email: songlinqi@ucla.edu, cmtkn@ucla.edu, mihaela@ee.ucla.edu.

finite and small, we can run X recommenders, one for each context, to tackle this issue. However, when there are large or infinite number of contexts, this method fails since the number of recommenders increases significantly. Hence, designing and building effective contextual recommender systems with large or infinite numbers of contexts remains a key challenge.

In this paper we propose methods for contextualization which helps the recommender to learn from its past experiences that are relevant to the current user characteristics, to recommend an item to the current user.

(b) *Differential services* are important for online services, including recommender systems. Existing works evaluate recommender systems based on their average performance [22] [23] [29]. However, it may also be desirable for service providers to build customer loyalty by providing registered users with a higher quality of service (in this case better recommendations) than the quality provided to anonymous users using the system. Hence, designing recommender systems which have the ability to provide differential services to various users represents a key challenge.

(c) *Scalability* is a key challenge faced by a recommender system with a large number of items and/or contexts. If not designed properly, the learning speed and the implementation complexity of such systems can suffer significantly [3] [7] [10]. Hence, the recommender system needs to easily scale to accommodate a large number of items and contexts.

Another important challenge is *cold start*, which appears in practice when the system has a very diverse set of items and very limited or no a priori information about the items that the users prefer [3] [7]. In this case, recommendation algorithms [2] [6] [9] fail to recommend items that are most liked by users due to the limited information about the user's preference for specific items. This problem is also often referred to as the "coverage" problem [3] [7]. New items are unlikely to be recommended and new users are unlikely to be given good recommendations due to the limited interaction histories of these new items/users. Therefore, dealing with cold start remains a key challenge for large-scale recommender systems. Two important features of our proposed algorithm address the issue of cold start: contextualization and continuous exploration. By contextualization, our algorithm clusters the users according to their contexts and items according to their features such that past information gained from users with similar contexts and items with similar features can be used to solve the cold start problem when a new user comes or a new item is introduced. Moreover, our algorithm explores continuously, i.e., it does not explore only initially based on limited data, and then exploit for the rest of the users. It explores and exploits in an alternating fashion, and keeps learning as new users come. This continuous exploration addresses the cold start problem since it helps the recommender to learn the payoffs of new items and preferences of new users that are not similar to the previous ones over time.

In this paper, we propose a novel, large-scale contextual recommender system which is able to address all of the above mentioned challenges. Our system is based on a novel contextual *Multi-Arm Bandit (MAB)* approach. When a user with a specific context arrives, the recommender utilizes information from past arrivals within a neighborhood of the current context (i.e., "similar" to the current context). Based on the payoff estimations through these past arrivals, the recommendation is made at an item-cluster level, instead of an item level. Our approach controls the size of the context neighborhood and the set of clusters to obtain sufficiently accurate and efficient payoff estimations. To learn about new items and contexts, our algorithms use a differential method for exploration: the system explores more often when the users have a lower priority, at the risk of providing lower quality recommendations, and explore less in order to provide consistently high quality recommendations when the users have a higher priority. After each recommendation, the realized payoff is received by the recommender system. Each item-cluster corresponds to an action (referred to as the *arm* in a bandit framework) for our learning algorithm, hence learning is performed at the cluster level, and this solves the scalability problem.

The goal of learning is to minimize the *regret*, which is the difference between the optimal expected total payoff up to time T, which can be achieved if all information (i.e., expected payoffs of each item/cluster in a certain context) is known, and the expected total payoff gained up to time T through our learning algorithm, in which the information is incomplete.

The remainder of this paper is organized as follows. In Section 2, we discuss the related works and highlight the differences from our work. Section 3 describes the system model of the recommender system. Section 4 introduces the adaptive clustering based contextual MAB recommendation algorithm. Section 5 provides simulation results. Section 6 concludes the paper.

2 Related Works

A plethora of prior works exists on recommendation algorithms. We compare our work with existing works in Table I. We categorize these algorithms based on the following characteristics: their consideration of contexts; their refinement of the context space (if any); their ability to address the issues of "scalability", "cold start", and "differential services"; and their ability to operate without a period of training. Two main categories are filtering-based and reinforcement learning methods [3]. Filtering-based approaches, such as collaborative filtering [7] [8], content-based filtering [2] [9] and hybrid approaches [10] [11], employ the

TABLE 1: Comparison with Existing Algorithms

	Contexts	Refinement in the context space	Solving "scalability"	Solving "cold start"	Requiring training data	Differential services
[2] [4] [6] [8] [9] [11] [16]	No	-	No	No	Yes	No
[5] [21]	Yes	Partition	No	No	Yes	No
[23] [24]	Yes	Linear payoff assumption	No	Yes	No	No
[25]	Yes	Partition	No	Yes	Yes	No
[27] [28]	No	-	Yes	Yes	No	No
[29]	Yes	Partition	Yes	Yes	Yes	No
Our work	Yes	An adaptive neighborhood	Yes	Yes	No	Yes

historical data of users' feedback to calculate the future payoffs of users based on some prediction functions. Collaborative filtering approaches predict users' preferences based on the preferences of their "peer" users, who have similar interests on other items observed through historical information [7] [8]. Content-based filtering methods help certain users to find the items that are similar to the past selected items [2] [9]. Hybrid approaches combine the collaborative filtering and content-based filtering approaches to gather more historical information in the user-item space when a new user or a new item appears in the system [10] [11]. However, filtering-based approaches are unable to solely tackle issues of cold start and scalability. Although the hybrid approaches can alleviate the cold start problem to some extent due to the consideration of both similar items and similar users when a new item or user appears, it cannot solve the cold start problem when no similar items/users of a new item/user exist in the system.

Reinforcement learning methods, such as MAB [23] [24] [25] and Markov Decision Processes (MDPs) [26], are widely used to derive algorithms for recommending items. MDP-based learning approaches model the last k selections of a user as the state and the available items as the action set, aiming at maximizing the long-term payoff [26]. However, the state set will grow fast as the number of items increases, thereby resulting in very slow convergence rates. In [32], "state aggregation" methods are used to solve this MDP problem with large state set, but the incurred regret grows linearly in time. MAB-based approaches [12] [13] [15] [17], such as ε-greedy [17] and UCB1 [15], provide convergence to the optimum, and not only asymptotic convergence, but also a bound on the rate of convergence for any time step. They do this by balancing exploration and exploitation, where exploration involves learning about new items' payoffs for a particular user by recommending new items while exploitation means recommending the best items based on the observations made so far.

To further improve the effectiveness of recommendations, contexts are considered in recent recommender systems [18] [21] [22]. For instance, it is important to consider the search queries and the purchase histories (which can be considered as the contexts) of users when making news article recommendations. Moreover, in many applications, such as news recommendations, the recommender system only observes the features of anonymous users, without knowing the exact users. In this case, the users' features (cookies) can be also considered as contexts. The context-aware collaborative-filtering (CACF) and the contextual MAB methods have been studied to utilize such information [21] [23] [25] [29]. The CACF algorithm extends the collaborative filtering method to a contextual setting by considering different weights for different contexts [21]. In [23] [24], the payoff of a recommendation is modeled as a linear function of the context with an unknown weight vector, and the LinUCB algorithm is proposed to solve news article recommendation problems. However, the linearity may not hold in practice for some recommender systems [22] [25] [29]. In a more general setting [25], the context space is modeled as a bounded metric space in which the payoff function is assumed to satisfy a Lipschitz condition, instead of being a linear function of contexts. In this context space, partition based algorithms are proposed to solve the contextual recommendation problems [25]. These works [18] [22] [25], however, do not take into account the large item space, which can cause scalability problems in practice.

A different strand of related works studies recommender systems that have a large number of items to recommend [3] [7] [27] [28]. In [28], the item space is partitioned into subspaces, and online learning is performed in each subspace. Alternatively, in [27], the problem is solved by considering a static clustering of items. These works [27] [28] can solve the scalability problem by reducing the number of choices of each recommendation. However, these works [3] [7] [27] [28] do not incorporate contexts in the recommendations. In contrast, we consider the contexts when making recommendations.

Building large-scale contextual recommender systems is more challenging. Few related works tackle both problems in an integrated manner. The most relevant theoretical works related to our proposed solution are [25] [29]; however, there are significant differences between our work and these existing works. First, the works in [25] [29] are theoretical and do not consider the key characteristics and requirements of recommender systems. For example, the algorithm proposed in [29] requires the knowledge of Lipschitz constants, which makes the algorithm data-dependent (different datasets may have different Lipschitz constants based on the characteristics of the data) and hence difficult to implement in practice; the algorithm in [25] requires knowing the covering constants of context and item spaces. In contrast, our learning algorithm does not need the Lipschitz constant or covering constants to operate; these constants are only used to analyze the performance of the algorithm (i.e., to show the regret). Second, the models in [25] [29] consider only two spaces: the context space and the item space; while in our work, we consider the user space, the context space, and the item space. Hence, we can provide differential services for different types of users by choosing different trade-offs between exploitation and exploration. Third, in [29], combining the items and contexts greatly increases the space dimension, resulting in a highly-complex online algorithm. In our work, we greatly reduce the dimension of the spaces by separately considering the context and the item space. We build the item-cluster tree offline, and only perform online calculations in the context space, resulting in lower-complexity online learning compared with that in [29].

In existing works [25] [27] [28] [29], the algorithms operate by fixing a partition of the context space. In contrast, our approach selects a dynamic subspace in the context space each time to ensure a sufficiently accurate estimation of the expected payoffs with respect to current context, with a precise control of the size of the context subspace. Moreover, our approach can provide differential services (different instantaneous performance) to ensure the satisfaction of higher type users, which are not considered before in existing works [4] [5] [6] [8] [9] [21] [23] [27] [28] [29]. A detailed comparison with the existing works is presented in Table 1.

3 Recommender System Model

We consider the contextual recommender system shown in Fig. 1, which operates in discrete time slots $t = 1, 2, \cdots$. At each time slot, the system operates as follows: (1) a user with a specific context arrives; (2) the recommender system observes the context, and recommends to the user an item from the set of items, based on the current context as well as the historical information about users, contexts, items, and payoffs of former recommendations; and (3) the payoff of this recommendation, whose expectation depends on the item and context, is received by the recommender system according to the user's response.

Users: We formally model the set of users as $\mathcal{U}$, which can be either a finite set (i.e., $\mathcal{U} = \{1, 2, \cdots, U\}$) or infinite set (i.e., $\mathcal{U} = \{1, 2, \cdots\}$, e.g., a news recommender system frequented by an infinite number of users). When a user $u \in \mathcal{U}$ arrives, the system can observe the user's type: important or ordinary users, registered or anonymous users, etc. We divide the users into S types, and denote the set of type $s \in \{1, 2, \cdots, S\}$ users by $\mathcal{U}_s$. Based on the users' types, the recommender system can provide differential services for users.

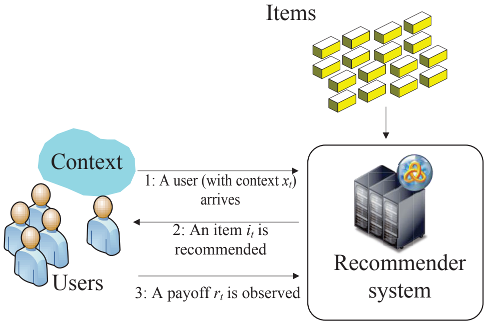

Items
Context
1: A user (with context x_t) arrives
2: An item i_t is recommended
3: A payoff r_t is observed
Users
Recommender system

Fig. 1: Contextual Recommender System Model.

Contexts: We model the context set $\mathcal{X}$ as a d_C dimensional continuous space, and the context $x \in \mathcal{X}$ is a d_C dimensional vector. For simplicity of exposition, we assume that the context space is $\mathcal{X} = [0,1]^{d_C}$, but our results will hold for any bounded d_C dimensional context space. In the context space, the similarity distance s_C is defined as a metric $s_C : \mathcal{X} \times \mathcal{X} \to [0, \infty)$. A smaller $s_C(x, x')$ indicates that the two contexts x and x' exhibit higher similarity. In the Euclidean space $\mathcal{X}$, the metric s_C can be the Euclidean norm or any other norm. Note that we will add the subscript t to user u and context x when referring to the user and context associated with time period t.

Items: We denote the set of items by $\mathcal{I} = \{1, 2, \cdots, I\}$. In the item space, the similarity distance is defined as a metric $s_I : \mathcal{I} \times \mathcal{I} \to \mathbb{R}$, which is based on the features of the items and known to the recommender system, as in [10] [22] [29]. A smaller $s_I(i, i')$ indicates that the two items i and i' exhibit higher similarity. For example, in a news recommender system, an item is a news item, and the similarity between two news items depends on their features: whether they are in the same category (e.g., Local News), whether they have the same keywords (e.g., Fire) etc.

Since the number of items in the system is large, we cluster them based on the metric s_I in the item space in order to reduce the possible options for recommendation. We use an item-cluster tree to represent the hierarchical relationship among item clusters, and an example is shown in Fig. 2. In an item-cluster tree, each leaf represents an item and each node represents an item cluster. The depth of a node is defined as the length of the path to the root node, and the depth of the tree is defined as the maximum depth of all nodes in the tree. We denote the depth of the tree by L. We define the collection of nodes at depth $0 \le l \le L$ of the tree as *layer* l and a node[1] at layer l is denoted by

[1] Note that each node of the tree corresponds to an item cluster, so we will use the terms "node" and "cluster" and their notations interchangeably.

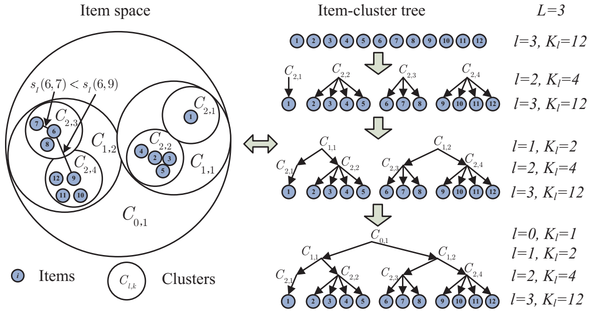


Fig. 2: Construction of Item-cluster Tree.

$C_{l,k}$, where $k \in \{1, 2, \cdots, K_l\}$ and K_l is the number of nodes at depth l. There is only one node $C_{0,1}$ at layer 0, i.e., the root node. The I nodes at layer L are the leaves of the tree, each of which is a single-element cluster that only contains one item. The hierarchical structure of the cluster tree implies that for a child node $C_{l+1,k'}$ of node $C_{l,k}$, we have item $i \in C_{l,k}$, for all $i \in C_{l+1,k'}$.

The cluster size $s(C_{l,k})$ is defined as the maximum distance of two items in that cluster, namely, $s(C_{l,k}) = \max_{i,i' \in C_{l,k}} \{s_I(i,i')\}$. In an item-cluster-tree structure, a general tree metric can be defined as a mapping from the set of depths of the nodes/clusters in the tree to maximum size of clusters, i.e., $s_T : \mathbb{N} \to \mathbb{R}$. Thus, $s_T(l)$ denotes the maximum cluster size at depth l, namely, $s_T(l) \geq \max_{i,i' \in C_{l,k}, 1 \leq k \leq K_l} \{s_I(i,i')\}$ for any cluster at depth l. For a tree metric, the maximum cluster size at depth l is not smaller than that of a cluster at depth $l+1$, i.e., $s_T(l) \geq s_T(l+1)$. We can see from Fig. 2 that item 6 and item 7 are in the same cluster $C_{2,3}$; item 6 and item 9 are in the same cluster $C_{1,2}$; and the metric between item 6 and item 7 is smaller than that between item 6 and item 9, namely, $s_I(6,7) < s_I(6,9)$. In Section 4, we will provide an algorithm to construct such item-cluster trees. Using the constructed item-cluster tree, we can control the size and number of the clusters when the recommendations are made.

The expected payoff of a recommendation depends on the context and the item. For a user with context $x \in \mathcal{X}$, the payoff of recommending item i is denoted by a random variable $r_{i,x} \in [0,1]$, whose distribution is fixed but unknown to the recommender system. The expectation of $r_{i,x}$ is denoted by $\mu_{i,x}$. Given the number of items $n_{l,k}$ in cluster $C_{l,k}$, the expected payoff of cluster $C_{l,k}$ for context x is denoted by $\mu_{l,k,x} = \sum_{i \in C_{l,k}} \mu_{i,x}/n_{l,k}$. Only the payoffs of the recommended items can be observed by the recommender system and can be used for further recommendations.

Context-aware collaborative-filtering algorithms consider that an item has similar expected payoffs in similar contexts [18] [19] [21] [22], while contextual bandit algorithms use a Lipschitz condition to model the similar expected payoffs of an item in two similar contexts [25] [29]. As in [25] [29], we formalize this in terms of a Lipschitz condition as follows.

Assumption 1 (Lipschitz condition for contexts): Given two contexts $x, x' \in \mathcal{X}$, we have the following assumption: $|\mu_{i,x} - \mu_{i,x'}| \leq L_C s_C(x,x')$ for any item i, where L_C is the Lipschitz constant in the context space.

Similarly to [28] [29], we model the similar payoffs for similar items in the same context using a Lipschitz condition, described in Assumption 2.

Assumption 2 (Lipschitz condition for items): Given two items $i, i' \in \mathcal{I}$, we have the following assumption: $|\mu_{i,x} - \mu_{i',x}| \leq L_I s_I(i,i')$ for any context $x \in \mathcal{X}$, where L_I is the Lipschitz constant in the item space.

Based on Assumption 2, in an item-cluster tree with tree metric $s_T(\cdot)$, we have $|\mu_{i,x} - \mu_{i',x}| \leq L_I s_T(l)$ for any $x \in \mathcal{X}$ and $i, i' \in C_{l,k}$. Note that the Lipschitz constants L_C and L_I are not required to be known by our recommendation algorithm. They will only be used in quantifying its performance.

4 Recommendation Algorithm

In this section, we first propose the item-cluster tree construction method to cluster the items, then propose the *Adaptive Clustering Recommendation (ACR)* algorithm and show its performance bound in terms of regret.

4.1 Construction of Item-Cluster Tree

We first cluster items using an item-cluster tree, such that the recommendations are made at a cluster level instead of an item level which reduces the possible recommendation options (arms) to solve the scalability problem. To explicitly characterize the hierarchy of the item-cluster tree, we denote the tree by a tuple $< L, \{\mathcal{K}_l\}_{0 \leq l \leq L}, \{Chd(C_{l,k})\}_{0 \leq l \leq L, 1 \leq k \leq K_l}, \{Prt(C_{l,k})\}_{0 \leq l \leq L, 1 \leq k \leq K_l} >$, where L is the depth of the tree; $\mathcal{K}_l = \{C_{l,k} : 1 \leq k \leq K_l\}$ is the cluster set at layer l, and $K_l = |\mathcal{K}_l|$ is the cardinality of $\mathcal{K}_l$; $Chd(C_{l,k})$ is the set of child clusters of cluster $C_{l,k}$ (a subset of $\mathcal{K}_{l+1}$ for $0 \leq l < L$, and $\emptyset$ for $l = L$); $Prt(C_{l,k})$ is the set of parent cluster of cluster $C_{l,k}$ (a one-element subset of $\mathcal{K}_{l-1}$ for $0 < l \leq L$, and $\emptyset$ for $l = 0$).

In this paper, we adopt an *exponential tree metric*: $s_T(l) = b^l$, where $b \in (0.5, 1)$ is the constant base. Our goal is to construct an item-cluster tree that fulfils this tree metric, based on an arbitrary given metric among items[2]. The constructed item-cluster tree will be used in the recommendation algorithm. The *Construction of Item-Cluster Tree* (CON-TREE) algorithm is described in Algorithm 1, and an example is shown in Fig. 2.

The algorithm first sets each leaf at depth L to be a single item cluster, and then operates from leaves to root by combining clusters that are "similar". The

[2] Item-cluster trees that fulfill a certain tree metric are not unique; however, they will provide the same performance bound for the recommendations as shown in our theorems.

Algorithm 1 Construction of Item-Cluster Tree

```
Input: item set $\mathcal{I} = \{1, 2, \cdots I\}$, base $b$, depth $L$ of the tree.
Initialization: Set $L := \min\{L, L_b\}$, where $L_b$ is the maximum $l$ such that $b^{l-1} \geq \min_{i,i' \in \mathcal{I}, s_I(i,i') \neq 0} s_I(i, i')$.

set node set at layer $L$ as $\mathcal{K}_L := \{C_{L,1}, C_{L,2}, \cdots C_{L,K_L}\}$, where $C_{L,k} := \{k\}, \forall 1 \leq k \leq K_L$, and $K_L = I$.
set representative point set at layer $L$ as
    $\mathcal{RP}_L := \{i_{L,1}, i_{L,2}, \cdots i_{L,K_L}\}$, where $i_{L,k} := k, \forall 1 \leq k \leq K_L$.
set the child sets of nodes at layer $L$ as empty sets:
    $Chd(C_{L,k}) := \emptyset, \forall 1 \leq k \leq K_L$.
for $l$=($L$-1):0 do
  set $k := 0$. //counter for the number of clusters at layer $l$
  while $\mathcal{RP}_{l+1} \neq \emptyset$ do
    set $k := k + 1$.
    arbitrary select an element $i_{l+1,\hat{k}} \in \mathcal{RP}_{l+1}$.
    find the subset $\mathcal{SK}$ of $\mathcal{K}_{l+1}$ such that
        $\mathcal{SK} := \{C_{l+1,k'} : 1 \leq k' \leq K_{l+1},\ i_{l+1,k'} \in \mathcal{RP}_{l+1},$
            $s_I(i_{l+1,\hat{k}}, i_{l+1,k'}) \leq \frac{(1-b)b^l}{1+b}\}$.
    set $C_{l,k} := \cup_{C_{l+1,k'} \in \mathcal{SK}} C_{l+1,k'}$.
    set $Chd(C_{l,k}) := \mathcal{SK}$.
    set $Prt(C_{l+1,k'}) := C_{l,k}, \forall C_{l+1,k'} \in \mathcal{SK}$.
    set $i_{l,k} := i_{l+1,\hat{k}}$.
    remove representative points of selected clusters from $\mathcal{RP}_{l+1}$:
        $\mathcal{RP}_{l+1} := \mathcal{RP}_{l+1} \backslash \{i_{l+1,k'} : C_{l+1,k'} \in \mathcal{SK}\}$.
  end while
  set $K_l := k$.
end for
set $Prt(C_{0,1}) := \emptyset$.
```

similarity between two clusters is evaluated through the distance between the representative points in the clusters. If the distance between two representative points is below a layer-varying threshold, the two clusters are considered similar and have the same parent cluster.

A key property of the CON-TREE algorithm is that it not only constructs an item-cluster tree that fulfils the exponential tree metric, but that it also has a bound on the maximum number of item clusters at each layer. This is important because we can balance the speed of learning and the accuracy by adjusting the number of clusters and the cluster size. We formalize this property in the following theorem.

Theorem 1: The item-cluster tree constructed through the CON-TREE algorithm has the following properties: (1) the constructed tree with a depth at most L_b fulfils the exponential tree metric, where L_b is the maximum l such that $b^{l-1} \geq \min_{i,i' \in \mathcal{I}, s_I(i,i') \neq 0} s_I(i, i')$, and (2) the maximum number of clusters at layer l of this tree is bounded by $c_I(\frac{1+b}{1-b})^{d_I}(\frac{1}{b})^{ld_I}$, where c_I and d_I are the covering constant and covering dimension for the item space[3].

Proof: see Appendix B.

We can see that according to the definition of the exponential tree metric, the proposed CON-TREE algorithm ensures that the cluster size at layer l is bounded by b^l, and from Theorem 1, the number of clusters at layer l is bounded by $O(b^{-ld_I})$. If l is larger, then the cluster size at layer l is smaller, but the number of clusters at layer l is larger, and vice versa. This implies that choosing clusters with larger depth l will make more specific recommendations to the users, but it will require learning more about specific characteristics of items since the number of clusters is larger. We also note that the item-cluster tree can be partitioned by a set of clusters $\mathcal{K}_l$ at layer l, which contain all the items and no two clusters contain the same item. Thus any item is included in one of the clusters at the same layer.

Note that since the CON-TREE algorithm needs up to K_l^2 calculations at layer l, the computational complexity of the CON-TREE algorithm is bounded by $O(\sum_{l=0}^{L_b} b^{-2ld_I})$. Since it is implemented offline, the CON-TREE algorithm does not affect the computational complexity of the online recommendation.

4.2 Adaptive Clustering Recommendation

We have previously proposed a one-layer clustering recommendation algorithm (OLCR) in [30] to solve the contextual recommendation problem with large number of items. However, the OLCR algorithm is based on fixed clustering method. A disadvantage of OLCR is that it is difficult to decide the optimal size of the clusters. In this work, we propose an adaptive clustering based ACR algorithm that can adaptively select the cluster size in order to balance between making specific recommendations and reducing the number of clusters. We choose the "optimal item" as the benchmark, which is defined as $i^*(t) = \arg\max_{i \in \mathcal{I}} \mu_{i,x_t}$ with respect to the context x_t arriving at time t. The regret of learning up to time T of an algorithm π, which selects the $\pi_{\mathcal{K}}(t)$ cluster at layer $\pi_{\mathcal{L}}(t)$ (i.e., cluster $C_{\pi_{\mathcal{L}}(t), \pi_{\mathcal{K}}(t)}$), given the context at time t, is defined as the difference of expected payoffs from choosing item $i^*(t)$, namely,

$$R_\pi(T) = \sum\nolimits_{t=1}^{T} [\mu_{i^*(t),x_t} - \mu_{\pi_{\mathcal{L}}(t), \pi_{\mathcal{K}}(t), x_t}]. \qquad (1)$$

Since for different types of users, the recommender system may provide differential services in order to maintain the higher type users' satisfaction. We assume that type s users have a certain fixed probability p_s to arrive at each time. For higher type users, the recommender system explores less , and for lower type users, the recommender system explores. In such a way, the average per-period regrets of higher type users are expected to be smaller than those of the

[3] See Appendix A for the definitions of covering dimension and covering constant.

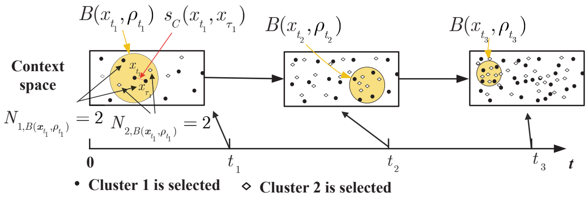


Fig. 3: Context arrivals and refinement in the context space.

lower type users. Formally, we define the instantaneous regret of type s users as $IR^s_\pi(t)=E[\mu_{i^*(t),x_t}-\mu_{\pi_{\mathcal{L}}(t),\pi_{\mathcal{K}}(t),x_t}|u_t\in\mathcal{U}_s]$ for an algorithm π, where the expectation is taken with respect to the context arrival given current user type.

Algorithm 2 Adaptive Clustering Recommendation Algorithm

```
Input: Item-cluster tree, periods $T$.
Initialization: initial partition $\mathcal{K}:=\mathcal{K}_0=\{C_{0,1}\}$, epoch $E:=\min\{L,\lfloor\log_2 T\rfloor\}$.
for $l=0:E-1$ do
  Set partition $\mathcal{K}:=\mathcal{K}_l=\{C_{l,k}:1\le k\le K_l\}$.
  for $t=2^l:2^{l+1}-1$ do
    Observe the user's type $s$ and the context $x_t$. Find a ball $B(x_t,\rho_t)$ with radius $\rho_t=t^{(\alpha-1)/d_C}$.
    Select the cluster with maximum index $k=\arg\max_{k'\in\mathcal{K}} I^s_{l,k}(t)$, with ties broken arbitrarily.
    Randomly select a child node $C_{l+1,\hat{k}}$ of $C_{l,k}$.
    Randomly recommend an item in cluster $C_{l+1,\hat{k}}$.
    Receive the reward: $r_t$.
    Record $\{t,x_t,C_{l,k},C_{l+1,\hat{k}},r_t\}$.
  end for
end for
Set $l:=E$, and set partition $\mathcal{K}:=\mathcal{K}_E=\{C_{E,k}:1\le k\le K_E\}$
for $t=2^l:T$ do
  Observe the context $x_t$. Find a ball $B(x_t,\rho_t)$ with radius $\rho_t=t^{(\alpha-1)/d_C}$.
  Select the cluster with maximum index $k=\arg\max_{k'\in\mathcal{K}} I^s_{l,k}(t)$, with ties broken arbitrarily.
  Randomly recommend an item in cluster $C_{l,k}$.
  Receive the reward: $r_t$.
  Record $\{t,x_t,C_{l,k},r_t\}$.
end for
```

We first present the ACR algorithm in Fig. 3, Fig. 4 and Algorithm 2. Initially, the item-cluster tree that fulfils the exponential tree metric has been built offline through the CON-TREE algorithm. We denote the epochs by $l=0,1,2\cdots$, and epoch l contains 2^l time slots. In epoch l, the ACR algorithm chooses the cluster set $\mathcal{K}_{\min\{l,L\}}$ that contains all the clusters at layer $\min\{l,L\}$, as shown in Fig. 4. Note that we will denote by $l(t)$ the layer selection corresponding to time slot t. As epoch goes on, the algorithm selects an item cluster from a cluster set that contains larger number of clusters, but these clusters have smaller cluster size. Hence, the algorithm can make more specific recommendations to users over time.

Each time, a user with context x_t arrives, and the algorithm chooses a relevant ball $B(x_t,\rho_t)$ in the context space, whose center is x_t and radius is $\rho_t=t^{(\alpha-1)/d_C}$ as shown in Fig. 3. It then selects a cluster in $\mathcal{K}_{l(t)}$, by calculating and comparing the index of each cluster at the current layer $l(t)$ based on the past history and current context. Subsequently, it randomly selects a child cluster at layer $l(t)+1$ (if available) and randomly recommends an item in the selected child cluster. Note that in the algorithm, to extract information of a former epoch, we record two-cluster selections each time[4]: one is the cluster $C_{l,k}$ at layer l, and the other one is the cluster $C_{l+1,\hat{k}}$ at layer l+1, which is a random selected child cluster of $C_{l,k}$.

The index represents a combination of exploration and exploitation, and its calculation is the main part of this algorithm. The set of past time periods when the contexts arrived at the relevant ball $B(x_t,\rho_t)$is denoted by $\mathcal{T}(B(x_t,\rho_t))=\{\tau:\tau<t, x_\tau\in B(x_t,\rho_t)\}$. The number of times that cluster $C_{l,k}$ has been selected in that ball is denoted by $N_{l,k,t}=\sum_{\tau\in\mathcal{T}(B(x_t,\rho_t))} I\{\pi_{\mathcal{L}}(\tau)=l,\pi_{\mathcal{K}}(\tau)=k\}$. Therefore, the index for cluster $C_{l,k}$ can be written as $I^s_{l,k}(t)=\bar{r}_{l,k,t}+\sqrt{A_s\ln t/N_{l,k,t}}$, where A_s is the exploration-exploitation trade-off factor for type s users, and $\bar{r}_{l,k,t}=(\sum_{\tau\in\mathcal{T}(B(x_t,\rho_t))} r_t I\{\pi_{\mathcal{L}}(\tau)=l, \pi_{\mathcal{K}}(\tau)=k\})/N_{l,k,t}$ is the sample-average payoff in the relevant ball. Without loss of generality, we assume that the exploration-exploitation trade-off factor A_s decreases as the users' type increases, namely, $A_1\ge A_2\ge\cdots\ge A_S$.

We define the suboptimal cluster set at time t as $\mathcal{S}(t)=\{(l(t),k):k\in\mathcal{K}_{l(t)},\bar{\mu}_{l(t),k^*_t,t}\mu_{l(t),k,t}>4L_C\rho_t\}$, where $k^*_t=\arg\max_k\mu_{l(t),k,t}$. We also define $\mathcal{W}^s_{k,t}=\{N_{l(t),k,t}<\frac{36A_s\ln t}{(\mu^*_{l(t),t}-\mu_{l(t),k,t}+2L_C\rho_t)^2}\}$. The following theorems prove a bound on the instantaneous and long-term regrets of the ACR algorithm.

Theorem 2: If $1/(1+d_C)<\alpha<1$ and the fully exploration assumption holds: $\Pr\{\pi(t)=(l(t),k)|u_t\in\mathcal{U}_s,\mathcal{W}^s_{k,t},(l(t),k)\in\mathcal{S}(t)\}\ge 1-t^{-\eta}$ for any time t, where $\eta>0$ is a parameter, then for independent and identically distributed (i.i.d.) user and context arrivals, the instantaneous regret $IR^s_{ACR}(t)$ of the type s user in period t is bounded by

$$IR^s_{ACR}(t)\le\begin{cases}K_{l(t)}2t^{-2A_1}+\frac{24A_1K_{l(t)}c_C\ln t}{L_C}t^{\frac{(1-\alpha)}{d_C}-\alpha}+4L_Ct^{-\frac{(1-\alpha)}{d_C}}\\ \qquad+2L_It^{\ln b/\ln 2},s=1\\ K_{l(t)}2t^{-2A_s}+4L_Ct^{-\frac{(1-\alpha)}{d_C}}+2L_It^{\ln b/\ln 2}\\ \quad+K_{l(t)}\prod_{s'=1}^{s-1}[1-\sum_{s''=1}^{s'}p_{s''}p_{B(x_t,\rho_t)}(1-t^{-\eta})]^{t_{s'}-t_{s'+1}},s\ne 1\end{cases},$$

where $t_{s'}=\min\{\tau:\frac{A_{s'}\ln(t-\tau)}{A_s\ln(t)}<(1+\frac{\rho_{t-\tau}}{3\rho_t})^2\}$, for $s'=1,2,...,s-1$, and $p_{B(x_t,\rho_t)}=\int_{x\in B(x_t,\rho_t)}f(x)dx$.

[4] Note that this can be generalized to arbitrary levels ahead, like children of children, but the computational complexity and memory requirements of each epoch grows.

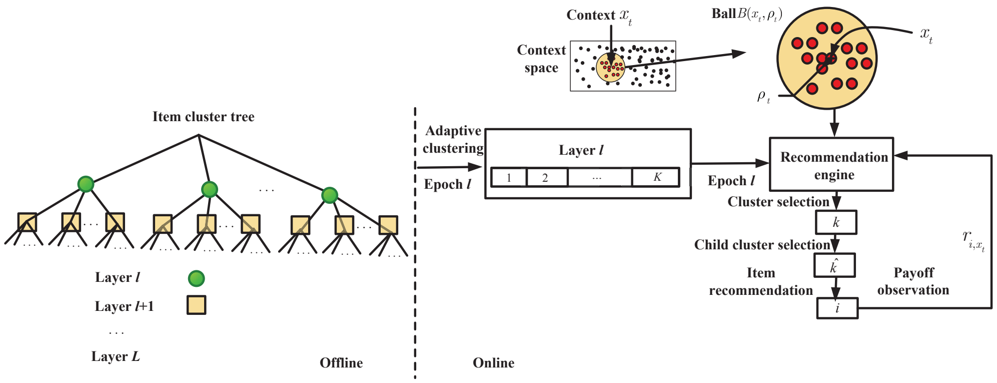


Fig. 4: Recommender system based on ACR algorithm.

Proof: See Appendix C.

We can see that type 1 users have a higher regret than other users, because they need to explore more than others. For type s users, we can see from Theorem 2 that as the percentage of lower type users ($s' < s$) increases, the instantaneous regret $IR^s_{ACR}(t)$ decreases, because these lower type users will explore more to benefit the type s user. We can also see that as the ratio of exploration-exploitation trade-off factors ($A_{s'}/A_s$, $s' < s$) increases, $t_{s'}$ increases, hence the regret $IR^s_{ACR}(t)$ decreases. This implies that a bigger ratio of $A_{s'}/A_s$ represents a higher differential service level.

From Theorem 2, we can observe that the instantaneous regret $IR^s_{ACR}(t)$ sublinearly decreases with t, namely, $IR^s_{ACR}(t) = O(t^{-\gamma})$, where $0 < \gamma < 1$. The instantaneous regret measures the performance loss in terms of the average payoff due to recommending a suboptimal item for the user. CTR is a common performance measure in applications such as news and advertisement recommendations [6] [23]. The regret bound on the CTR is $O(t^{-\gamma})$. Another interesting performance metric is the root-mean-square error (RMSE), which is commonly used in rating systems to measure the difference of the rating prediction from the user's actual rating [19] [20]. The regret can be converted to a regret bound on the RMSE. To do this, it is sufficient to consider a transformation in which the reward is considered as the negative mean-square of the suboptimality in CTR resulting from suboptimal item recommendations. From Theorem 2, it can be shown that the RMSE is $O(t^{-\gamma/2})$.

Theorem 3: For $1/(1+d_C) < \alpha < 1$ and i.i.d. user arrivals, the regret $R_{ACR}(T)$ up to time T of the ACR algorithm is bounded by

$$\begin{aligned} R_{ACR}(T) \leq 2K_E \sum_{s=1}^{S} p_s \sum_{t=1}^{\infty} t^{-2A_s} + \frac{2L_I T^{1+\frac{\ln b}{\ln 2}}}{1+\ln b/\ln 2} \\ + \frac{24A_1 K_E c_C (1.386+\ln T) T^{\frac{(1-\alpha)(1+d_C)}{d_C}}}{L_C(2^{(1-\alpha)(1+d_C)/d_C}-1)} \\ + \frac{48A_1 K_E c_C T^{\frac{(1-\alpha)(1+d_C)}{d_C}} \ln 2}{L_C(2^{(1-\alpha)(1+d_C)/d_C}-1)^2} + \frac{4L_C d_C T^{1-\frac{1-\alpha}{d_C}}}{d_C-1+\alpha} \end{aligned}, \quad (2)$$

where $E = \min\{L, \lfloor \log_2 T \rfloor\}$, c_C is the covering constant of the context space $\mathcal{X}$.

Proof: See Appendix C.

From Theorem 3, we observe that as long as the minimum exploration-exploitation trade-off factor satisfies $A_S \geq \frac{d_I \ln 2}{2\ln(1/b)}$, the the regret[5] $R_{ACR}(T)$ is sublinear in T, since from Theorem 1, $K_{l(t)}$ can be bounded through: $K_{l(t)} \leq c_I(\frac{1+b}{1-b})^{d_I}(\frac{1}{b})^{d_I \log_2 t} \leq c_I(\frac{1+b}{1-b})^{d_I} t^{-d_I \frac{\ln 2}{\ln b}}$. If the system does not consider different user types, then all $A_s = 1$, then the regret can be simplified as $O(T^{(d_C+d_I+1)/(d_C+d_I+2)} \ln T)$, by setting $\alpha = (d_I+2)/(d_I+d_C+2)$ and $b = 2^{-1/(d_I+d_C+2)}$.

4.3 Theoretical Comparison

In this subsection, we compare the computational complexity (time complexity), the memory required (space complexity), and the performance bound with the benchmark in [29], shown in Table 2. For the ACR algorithm, the computational complexity is $O(t+K_{l(t)})$ at each time t. In this case, the computational complexity up to time T is $O(T^2 + K_E T)$, which is polynomial in T. The space complexity for the ACR algorithm is $O(\sum_{l=0}^{E} K_l + T)$. The ACR algorithm achieves the a regret $O(T^{(d_C+d_I+1)/(d_C+d_I+2)} \ln T)$. Actually, the regret of the ACR algorithm is potentially better than $O(T^{(d_C+d_I+1)/(d_C+d_I+2)} \ln T)$ due to that K_E is often much smaller than $c_I(\frac{1+b}{1-b})^{d_I} T^{-d_I \ln 2/\ln b}$.

We also note that when the dimensions of the context and item spaces are small, the algorithm in [29] has a lower time complexity than our algorithm ($O(T^{1+2\lambda}+IT) < O(T^2 \ln T + K_E T)$) (when $\lambda < 0.5$)[6]. But when the dimensions of the context and item spaces are big, our algorithms achieve a lower time complexity ($O(T^2 \ln T + K_E T) < O(T^{1+2\lambda} + IT)$) (when $\lambda > 0.5$).

5 Experimental Results

In this section, we evaluate the performances of our algorithm through experiments, using the Yahoo! Today Module dataset.

[5] At the beginning of the online learning periods, the algorithm needs to explore more, hence incurring higher regret. To tackle this, we can use a prior information of payoffs for each item in different contexts.

[6] In [29], T^λ is the number of balls to be selected, $\lambda \in (0,1)$ and λ increases as the dimension of the context and item spaces increase.

TABLE 2: Comparison with Existing Solution

	Regret	Space complexity	Time complexity
[29]	$O(T^{\frac{d_C+d_I+1}{d_C+d_I+2}} \ln T)$	$O(I+T+T^{\lambda})$	$O(T^{1+2\lambda}+IT)$
ACR	$O(T^{\frac{d_C+d_I+1}{d_C+d_I+2}} \ln T)$	$O(\sum_{l=0}^{E} K_l+T)$	$O(T^2+K_E T)$

5.1 Description of the Dataset

The Yahoo! Today Module dataset[7] is based on the webpage recommendations of Yahoo! Front Page, which is a prominent Internet news website of Yahoo! This dataset contains around 40 million instances collected from Yahoo! Front Page during May 1 to 10, 2009. Each instance of the dataset includes (1) recommended item IDs and features, (2) contexts, and (3) click behavior of users. For each user the five-dimensional context vector describes the user's features, and is obtained from a higher dimensional raw feature vector of over 1000 categorical components by applying dimensionality reduction techniques [23]. The raw feature vector includes: 1) demographic information: gender and age; 2) geographic features; and 3) behavioral categories: about 1000 binary categories that summarize the user's consumption history. The high dimensional raw feature vector then is converted to a five dimensional context feature vector [23]. The number of items in this dataset is 271, all of which are used in our experiments.

The special data collection method [23] allows evaluating online algorithms on this dataset without introducing bias. Essentially, each instance in the dataset contains only the click behavior of a user for 1 item that is recommended to the user at the time of data collection. When collecting the dataset, the item recommendations are done completely independent of the context of the user and the previous item recommendations. As explained in [23], this removes any bias in data collection that might result from the algorithm used to make item recommendations when collecting the data.

Similar to [23], we use this offline dataset to evaluate our online learning algorithms. When simulating our algorithms, we randomly pick users from this dataset. Since this data set is very large (around 40 million instances), for a user that has a context x, it is possible to find 271 instances, each of which contains a different item recommendation, a context that is within ε of x in terms of the Euclidian distance (in our experiment, we use $\varepsilon = 0.01$), and the user's click behavior for that item recommendation. In this way, the online algorithm is evaluated by using the users and their click behaviors in this dataset. Note that a very similar simulation method is also considered in [23] to run the online LinUCB algorithm on this offline dataset. In our numerical results we consider $T = 70000$ user arrivals.

5.2 Comparison with Existing Context-free Recommendation Algorithms

To show the importance of contexts on the payoffs, we compare the performances of our proposed ACR algorithm and our previously proposed OLCR algorithm [30] with those context-free algorithms. The performances of the algorithms are evaluated through CTR up to time t. We compare against the following three context-free algorithms.

- Random: The algorithm randomly selects an item each time. This can be seen as the benchmark for other algorithms.
- UCB1 [15]: This is a classical MAB algorithm, which performs well in the case of learning the best item without exploiting the context information.
- ε-greedy [15] [17]: This is another widely-used algorithm to perform online recommendations. In time period t, item cluster with the highest payoff is selected with probability $1 - \varepsilon_t$, and other item clusters are randomly selected with probability ε_t, where ε_t is in $1/t$ order, such that the exploration rate is inversely proportional to the number of users.

In simulations, depending on the number of clusters we consider two scenarios. In the first scenario, we choose $K = 50$ item clusters, and in the second scenario, we choose $K = 150$ item clusters for all the algorithms except the ACR algorithm, which can adaptively choose the number of item clusters over time. For the ACR algorithm, we do not distinguish users (i.e., $A_s = 1$) and choose the parameter $b = 0.75$.

The comparison results are given in Table 3. We can see that both the OLCR algorithm and the ACR algorithm significantly outperform the context-free algorithms, as expected. The simulation results show that the OLCR algorithm achieves a 10%-31% performance gain, in terms of CTR up to time T, over the UCB1 and ε-greedy algorithms. The CTR of the ACR algorithm is 58%-69% higher than those of the UCB1 and ε-greedy algorithms. We also notice that the UCB1 and ε-greedy algorithms learn faster than the OLCR and ACR algorithms. This is because the context-free algorithms estimate the CTR of an item by averaging the clicks over all users, while the contextual algorithms estimate the CTR of an item based on specific user contexts. However, the contextual algorithms can make more accurate estimations of the expected CTRs of items by employing the contextual information when the number of user arrivals is sufficiently large. Hence, the context-free algorithms converge faster (less than 20% of user arrivals) than the contextual algorithms and the contextual algorithms can achieve a higher CTR than the context-free algorithms in the long run.

[7] http://labs.yahoo.com/Academic_Relations

TABLE 3: Comparison with Context-free Algorithms

		Percentage of user arrivals ($K = 50$)					Percentage of user arrivals ($K = 150$)				
		20%	40%	60%	80%	100%	20%	40%	60%	80%	100%
CTR	Random	0.026	0.028	0.027	0.026	0.026	0.026	0.028	0.027	0.026	0.026
	UCB1	0.028	0.032	0.030	0.029	0.029	0.029	0.030	0.029	0.028	0.029
	ε-greedy	0.030	0.029	0.029	0.029	0.029	0.032	0.031	0.031	0.031	0.031
	OLCR	0.028	0.033	0.037	0.037	0.038	0.025	0.028	0.031	0.034	0.034
	ACR	0.039	0.044	0.046	0.049	0.049	0.039	0.044	0.046	0.049	0.049
Gain	OLCR over Random	8%	18%	37%	42%	46%	-4%	0%	15%	31%	31%
	OLCR over UCB1	0%	3%	23%	28%	31%	-14%	-7%	7%	21%	17%
	OLCR over ε-greedy	-7%	14%	28%	28%	31%	-22%	-10%	0%	10%	10%
	ACR over Random	50%	57%	70%	88%	88%	50%	57%	70%	88%	88%
	ACR over UCB1	39%	38%	53%	69%	69%	34%	47%	59%	75%	69%
	ACR over ε-greedy	30%	52%	59%	69%	69%	22%	42%	48%	58%	58%

5.3 Comparison with Existing Contextual Recommendation Algorithms

In this subsection, we compare the CTRs of the OLCR and ACR algorithms with those of the existing contextual recommendation algorithms. We compare against the following algorithms.

- CACF [21]: The algorithm exploits the information of historical selections of item clusters to predict the current payoff given current context arrival. Note that this algorithm only makes exploitations, regardless of explorations. Thus, it has a "cold start" problem, and we use the first 5% of data instances to train the algorithm.
- Hybrid-ε [22]: This algorithm combines the contexts and ε-greedy algorithms together, by extracting information within a small region of the context space and by running ε-greedy algorithms in that small region.
- LinUCB [23]: This algorithm assumes that the payoffs of items are linear in contexts. It calculates the index of each arm by adding an upper bound term to a linear combination of contexts observed in previous periods, and then recommends the arm with the maximum index.
- Contextual-zooming [29]: This algorithm combines the context space and the item space together and selects a ball in this space each time based on the past context arrivals and item selections.

In this subsection we consider two fixed item clusters for algorithms that do not have the item clustering property: $K = 50$ and $K = 150$ (algorithms except contextual-zooming and ACR). For the ACR algorithm, we do not distinguish users (i.e., $A_s = 1$) and choose the parameter $b = 0.75$. We can categorize the contextual algorithms into two classes depending on whether an exploitation-exploration tradeoff is considered. The CACF algorithm does not make exploration, and the Hybrid-ε, LinUCB, contextual-zooming, OLCR, and ACR algorithms consider both the exploitation and the exploration.

The simulation results are given in Table 4. We can see that the ACR algorithm significantly outperform the existing contextual algorithms. The simulation results show that the CTRs up to time T obtained by the OLCR algorithm are 31% and 10% higher than those obtained by the CACF algorithm, are 15% higher and 3% lower than those obtained by the Hybrid-ε algorithm, are 7% and 18% lower than those obtained by the LinUCB algorithm, and are the same and 16% lower than those obtained by the contextual-zooming algorithm, in the scenarios of $K = 50$ and $K = 150$, respectively. The CTRs up to time T obtained by the ACR algorithm are 69% and 69% higher than those obtained by the CACF algorithm, are 48% and 48% higher than those obtained by the Hybrid-ε algorithm, are 20% and 26% higher than those obtained by the LinUCB algorithm, and are 29% and 29% higher than those obtained by the contextual-zooming algorithm. In fact, our ACR algorithm aggregates information in an adaptive ball of the context space each time, but the Hybrid-ε algorithm considers the fixed small region, which causes a low efficient learning. The LinUCB assumes the linear payoff function, which can result in the inaccurate estimation of the CTR, while our proposed algorithms do not make such an assumption. The performance of the contextual-zooming algorithm is affected by the Lipschitz constant included in the algorithm, which may not fit the real data very well. The CACF algorithm learns faster (converges in less than 20% user arrivals) than our proposed ACR and OLCR algorithms, but the accuracy of learning is low due to the fact that the CACF algorithm only makes exploitation, and our algorithms makes explorations that cause a lower speed of learning in the short run but a higher accuracy of recommendation in the long run. Note that the proposed ACR algorithm outperforms the OLCR algorithm, because the ACR algorithm exploits the item-cluster tree to adaptively cluster items, but the OLCR algorithm does not use this tree structure but has fixed clusters.

5.4 Differential Services

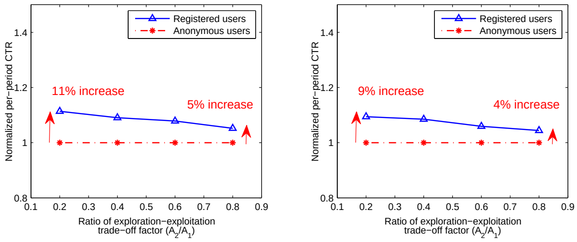
In this subsection, we show how the recommender system provides differential services according to users' types. We assume that the users in the system are divided into two types: registered users and anonymous users. For the registered users, we aim to provide a higher quality recommendation. In the sim-

TABLE 4: Comparison with Contextual Algorithms

		Percentage of user arrivals ($K = 50$)					Percentage of user arrivals ($K = 150$)				
		20%	40%	60%	80%	100%	20%	40%	60%	80%	100%
CTR	CACF	0.028	0.029	0.027	0.029	0.029	0.028	0.029	0.028	0.029	0.029
	Hybrid-ε	0.034	0.035	0.034	0.033	0.033	0.033	0.034	0.032	0.033	0.033
	LinUCB	0.035	0.039	0.040	0.041	0.041	0.032	0.037	0.039	0.039	0.039
	Contextual-zooming	0.037	0.039	0.038	0.038	0.038	0.037	0.039	0.038	0.038	0.038
	OLCR	0.028	0.033	0.037	0.037	0.038	0.025	0.028	0.031	0.032	0.032
	ACR	0.039	0.044	0.046	0.049	0.049	0.039	0.044	0.046	0.049	0.049
Gain	OLCR over CACF	0%	14%	37%	28%	31%	-11%	-3%	11%	10%	10%
	OLCR over Hybrid-ε	-18%	-6%	9%	12%	15%	-24%	-18%	-3%	-3%	-3%
	OLCR over LinUCB	-20%	-15%	-8%	-10%	-7%	-22%	-24%	-21%	-18%	-18%
	OLCR over contextual-zooming	-24%	-15%	-3%	-3%	0%	-32%	-28%	-18%	-16%	-16%
	ACR over CACF	39%	52%	70%	69%	69%	39%	52%	64%	69%	69%
	ACR over Hybrid-ε	15%	26%	35%	48%	48%	18%	29%	44%	48%	48%
	ACR over LinUCB	11%	13%	15%	20%	20%	22%	19%	18%	26%	26%
	ACR over contextual-zooming	5%	13%	21%	29%	29%	5%	13%	21%	29%	29%

ulation, we choose the ACR algorithm with parameter $b = 0.75$, and compare the average per-period CTR of two user types.

We show the average per-period CTR (normalized with respect to the anonymous users) of two user types in Fig. 5. We can see that the average per-period CTR of registered users is around 4% to 11% higher than that of the anonymous users when the ratio of exploration-exploitation trade-off factor A_2/A_1 (registered user/anonymous user) varies from 0.2 to 0.8 in the scenarios of 10% and 30% registered users. We can also see that as the ratio of exploration-exploitation trade-off factor increases, the CTR gain of the registered users over anonymous users decreases 5%-6%. In fact, as the ratio of exploration-exploitation trade-off factor increases, the recommendation strategy for both types of users become similar to each other.



(a) $p = 0.1$

(b) $p = 0.3$

Fig. 5: Normalized per-period CTR comparison between registered users and anonymous users, with percentage p of registered users.

5.5 Scalability of Algorithms

In practice, the number of items is large, which requires that the algorithms have the ability to scale up, i.e., learning speed and total CTR should not degrade as the number of items increases. In order to show this, we increase the number of items in the system by duplicating items to form the large item set. Since our proposed algorithms and the contextual-zooming algorithm make recommendations on cluster level, the increase in items in the system only affects the number of items in a cluster, and does not affect the performances of the algorithms. For the CACF, Hybrid-ε, and LinUCB algorithms, when they do not cluster the items and learn the CTR separately for each item, there is some performance losses when the number of items grows.

The simulation results are shown in Fig. 6. We use a scale factor σ to denote the scalability of the system (i.e., number of items is σK, where K is the number of items without being scaled up). We calculate the learning gain of the evaluated algorithm over the random algorithm (i.e., $(CTR_{\pi,\sigma K} - CTR_{random})/CTR_{random}$, where $CTR_{\pi,\sigma K}$ denotes the CTR of the algorithm π with a scale factor σ), and normalize it with respect to that of the ACR algorithm as a performance measure to evaluate the scalability of the algorithm. We can see that when the number of items becomes 100 times of current number, the performance loss in terms of the normalized learning gain against the random algorithm is 63%, 61%, and 67% for the CACF, Hybrid-ε and LinUCB algorithms, respectively. The long-term CTRs of the contextual-zooming algorithm and our proposed OLCR and ACR algorithms do not change, as expected, implying that our algorithms are scalable.

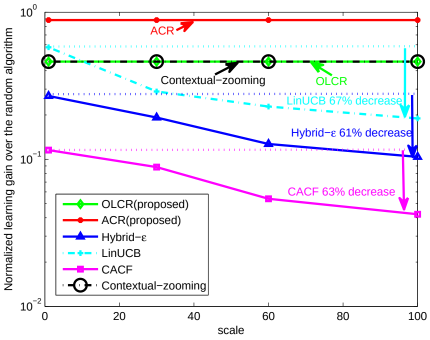


Fig. 6: Comparison of scalable performances of algorithms

5.6 Robustness of Algorithms

In this subsection, we evaluate the robustness of algorithms when input parameters vary. Note that, to construct the item-cluster tree in our proposed approach, the theoretical optimal value of b ($b \in (0.5, 1)$) given in Section 4.3 (i.e., $b = 2^{-1/(d_I + d_C + 2)} = 0.94$) may be conservatively large, and so optimizing this parameter

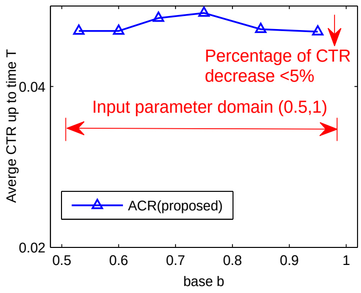


Fig. 7: The impact of b on the ACR algorithm.

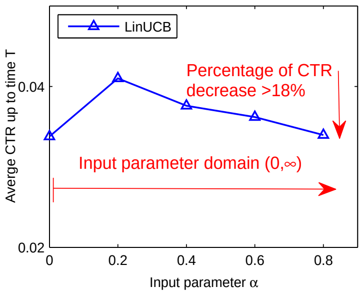


Fig. 8: The impact of α on the LinUCB algorithm.

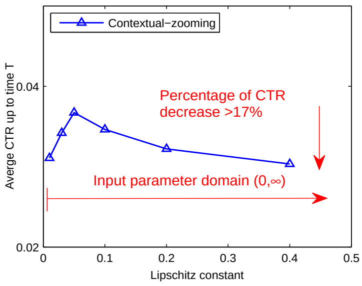


Fig. 9: The impact of the Lipschitz constant on the contextual-zooming algorithm.

may result in higher total payoffs. In particular, we compare the CTR of the ACR algorithm when the base b changes. In the existing works, the LinUCB algorithm [23] needs an input parameter $\alpha > 0$ (α is used to control size of the confidence interval) to run the algorithm, and the contextual-zooming algorithm [29] needs the Lipschitz constant to run the algorithm. So we also show the effect of parameters on the performances of the LinUCB and contextual-zooming algorithms.

Simulation results are shown in Fig. 7, 8, and 9. In Fig. 7, we can see that for the ACR algorithm, the CTR difference is less than 5% when the base b changes from 0.50 to 0.95. This is because the base b controls the trade-off of the number of clusters (learning speed) and the cluster size (accuracy). When b is large, learning is fast but less accurate; when b is small, learning is slow but more accurate. Hence, an optimal b exists to balance the speed of learning and the accuracy. For the LinUCB algorithm and the contextual-zooming algorithm, since the input parameters have a range from 0 to infinity, we test several possible values to evaluate the robustness of the algorithm. We can see from Fig. 8 that the LinUCB algorithm has more than 17% performance loss, in terms of CTR, compared to its maximum corresponding to the optimal value of α. We can also see from Fig. 9 that the contextual-zooming algorithm has more than 18% performance loss, in terms of CTR, when the Lipschitz constant is not estimated correctly. Moreover, since the range of input parameter α and Lipschitz constant is from 0 to infinity, it is difficult to find the optimal value of parameters without training data. Hence, we can see that the LinUCB algorithm and the contextual-zooming algorithm are less robust to the input parameters than the ACR algorithm.

5.7 User Arrival Dependent Performance Variation

In this subsection, we evaluate the robustness of our ACR algorithm to different types of user arrivals. We randomly generate 10 sequences of user arrivals, and evaluate the variation in the CTR of the ACR algorithm over these sequences. To evaluate the variations of CTR in different cases, we first calculate the average CTR up to time T of the 10 cases and normalize it to 1. We then compare the CTR variation of each case with the average CTR. The experimental results are shown in Table 5. For a specific sequence of user arrivals, a negative variation means the CTR is lower than the average CTR, while a positive variation means the CTR is higher than the average CTR. We can see from Table 5 that the variation of CTR up to time T obtained by our proposed algorithm is less than 5.5%.

6 Conclusions and Future Works

In this paper, we propose a contextual MAB based clustering approach to design and deploy recommender systems for a large number of users and items, while taking into consideration the context in which the recommendation is made. Our proposed ACR algorithm makes use of an adaptive item clustering method to improve the learning speed. More importantly, our algorithm can address the cold start problem and provide differential services to users of different types. Theoretical results show that the algorithm can achieve a sublinear regret, while our simulations show that our algorithm outperforms the state-of-the-art algorithms by 20% in terms of average CTRs.

Understanding the utility and impact of the proposed approach in the current practice of Web services and recommendations requires further study. Nevertheless, our online learning algorithm for recommendation systems is general and can potentially be used in numerous other application domains. However, each of these domains has its own unique set of contextual information, user characteristics, recommendation payoffs and associated challenges. For example, users' demographics, environmental variables (e.g., time, location), devices used (e.g. cellphones, laptops etc.) can be potential contexts, while the recommendation payoff can be the rating(s) given by users to the recommendation they receive. Hence, this will require adaptation of the proposed algorithm to the specific domain in which the recommendation system needs to operate.

While general and applicable to many domains, our approach has also several limitations. We mention here a few. First, we do not consider specific challenges associated with implementing the proposed

TABLE 5: The Variation of CTRs in Different User Arrival Processes

Cases	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8	Case 9	Case 10
Percentage of CTR variation compared to average CTR	4.4%	-3.6%	-4.0%	5.1%	-5.5%	-0.5%	5.1%	-5.2%	4.8%	-0.5%

system in a distributed manner, as several Web services and applications may require. However, this is essential and future work will need to consider such distributed solutions for implementing recommender systems. Second, recommender systems are increasingly co-evolving together with social networks: users often recommend items they liked or purchased to others in their social network. Hence, understanding how such recommendation systems and social networks co-evolve and interact with each other forms another interesting, yet challenging future work.

References

[1] P. Resnick and H. R. Varian, “Recommender systems,” *Communications of the ACM*, vol. 40, no. 3, pp. 56-58, 1997.

[2] M. Balabanovi and Y. Shoham, “Fab: content-based, collaborative recommendation,” *Communications of the ACM*, vol. 40, no.3, pp. 66-72, 1997.

[3] G. Adomavicius and A. Tuzhilin, “Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734-749, 2005.

[4] X. Chen, Z. Zheng, X. Liu, Z. Huang, and H. Sun, “Personalized QoS-aware web service recommendation and visualization,” *IEEE Transactions on Services Computing*, vol. 6, no. 1, pp. 35-47, 2013.

[5] Z. Zheng, H. Ma, M. R. Lyu, and I. King, “QoS-aware Web service recommendation by collaborative filtering,” *IEEE Trans.Transactions on Services Computing*, vol. 4, no. 2, pp. 140-152, 2011.

[6] J. Liu, P. Dolan, and E. R. Pedersen, “Personalized news recommendation based on click behavior,” *in Proceedings of the 15th International Conference on Intelligent User Interfaces*, pp. 31-40, 2010.

[7] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques,” *Advances in AI*, vol. 4, 2009.

[8] H. Kim, J. K. Kim, and Y. U. Ryu, “Personalized recommendation over a customer network for ubiquitous shopping,” *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 140-151, 2009.

[9] M. J. Pazzani and D. Billsus, “Content-based recommendation systems,” in *The Adaptive Web*, Springer Berlin Heidelberg, vol. 4321, pp. 325-341. 2007.

[10] R. Burke, “Hybrid recommender systems: Survey and experiments,” *User Modeling and User-adapted Interaction*, vol. 12, no. 4, pp. 331-370, 2002.

[11] K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H. G. Okuno, “An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model,” *IEEE Transactions on Audio, Speech, Language Processing*, vol. 16, no. 2, pp. 435-447, 2008.

[12] H. Liu, K. Liu, and Q. Zhao, “Logarithmic weak regret of non-Bayesian restless multi-armed bandit,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1968-1971, 2011.

[13] W. Dai, Y. Gai, B. Krishnamachari, and Q. Zhao, “The non-Bayesian restless multi-armed bandit: A case of near-logarithmic regret,” in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 2940-2943, 2011.

[14] K. Wang and L. Chen, “On optimality of myopic policy for restless multi-armed bandit problem: an axiomatic approach,” *IEEE Transactions on Signal Processing*, vol. 60, no. 1, pp. 300-309, 2012.

[15] P. Auer, N. Cesa-Bianchi, and P. Fischer, “Finite-time Analysis of the Multi-armed Bandit Problem,” *Machine Learning*, vol. 47, pp. 235-256, 2002.

[16] Y. Deshpande and A. Montanari, “Linear bandits in high dimension and recommendation systems,” in *2012 50th Annual Allerton Conference on Communication, Control, and Computing*, pp. 1750-1754, 2012.

[17] N. Cesa-Bianchi and G. Lugosi, “Prediction, learning, and games,” Cambridge, Cambridge University Press, 2006.

[18] G. Adomavicius and A. Tuzhilin, “Context-aware recommender systems,” in *Recommender Systems Handbook*, Springer US, pp. 217-253, 2011.

[19] G. Adomavicius, R. Sankaranarayanan, S. Sen, and A. Tuzhilin, “Incorporating contextual information in recommender systems using a multidimensional approach,” *ACM Transactions on Information Systems*, vol. 23, no. 1, pp. 103-145, 2005.

[20] A. Said, S. Berkovsky, E. W. De Luca, and J. Hermanns, “Challenge on context-aware movie recommendation: CAMRa2011,” in *Proceedings of the 5th ACM Conference on Recommender Systems*, pp. 385-386, 2011.

[21] A. Chen, “Context-aware collaborative filtering system: Predicting the user’s preference in the ubiquitous computing environment,” in *Location-and Context-Awareness*, Springer Berlin Heidelberg, pp. 244-253, 2005.

[22] D. Bouneffouf, A. Bouzeghoub, and A. L. Gancarski, “Hybrid-ε-greedy for mobile context-aware recommender system,” in *Advances in Knowledge Discovery and Data Mining*, Springer Berlin Heidelberg, pp. 468-479, 2012.

[23] L. Li, W. Chu, J. Langford, and R. E. Schapire, “A contextual-bandit approach to personalized news article recommendation,” in *Proceedings of the 19th International Conference on World Wide Web*, pp. 661-670, 2010.

[24] W. Chu, L. Li, L. Reyzin, and R. E. Schapire, “Contextual bandits with linear payoff functions,” in *Proceedings of International Conference on Artificial Intelligence and Statistics*, pp. 208-214, 2011.

[25] T. Lu, D. Pal, and M. Pal, “Contextual multi-armed bandits,” in *Proceedings of International Conference on Artificial Intelligence and Statistics*, 2010.

[26] G. Shani, D. Heckerman, and R. I. Brafman, “An MDP-Based Recommender System,” in *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, vol. 6, pp. 1265-1295, 2005.

[27] S. Pandey, D. Chakrabarti, and D. Agarwal, “Multi-armed bandit problems with dependent arms,” in *Proceedings of the 24th International Conference on Machine Learning*, pp. 721-728, 2007.

[28] R. Kleinberg, A. Slivkins, and E. Upfal, “Multi-armed bandits in metric spaces,” in *Proceedings of the 40th annual ACM symposium on Theory of computing*, pp. 681-690, 2008.

[29] A. Slivkins, “Contextual bandits with similarity information” in *24th Annual Conference on Learning Theory*, pp. 679-701, 2011.

[30] L. Song, C. Tekin, and M. van der Schaar, “Clustering based online learning in recommender systems: a bandit approach,” in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4528-4532, 2014.

[31] N. Ailon and M. Charikar, “Fitting tree metrics: hierarchical clustering and phylogeny,” in *46th Annual IEEE Symposium on Foundations of Computer Science*, pp. 73-82, 2005.

[32] Z. Ren and B. H. Krogh, “State aggregation in Markov decision processes,” in *Proceedings of the 41st IEEE Conference on Decision and Control*, pp. 3819-3824, 2002.

[33] E. Chlebus, “An approximate formula for a partial sum of the divergent p-series,” *Applied Mathematics Letters*, vol. 22, no. 5, pp. 732–737, 2009.

Appendix for “Online Learning in Large-scale Contextual Recommender Systems”

Linqi Song, Cem Tekin, and Mihaela van der Schaar, *Fellow, IEEE*

Appendix A Covering Dimension

Let $(\mathcal{X}, s)$ be a metric space, where $\mathcal{X}$ is a set, and $s : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ is the metric (distance). The ρ-covering $\mathcal{F}_\rho^{cov}$ of $\mathcal{X}$ is defined as a collection of subsets of $\mathcal{X}$, each with a diameter strictly smaller than ρ, that cover $\mathcal{X}$, i.e., $\mathcal{F}_\rho^{cov} = \{B : B \subseteq \mathcal{X}, s(x, y) < \rho/2, \forall x, y \in B\}$. The minimum subsets in $\mathcal{F}_\rho$ is called the ρ-covering number of $\mathcal{X}$ and is denoted by $N_\rho^{cov}(\mathcal{X})$. The *covering dimension* of $\mathcal{X}$ is defined as the smallest d such that $N_\rho^{cov}(\mathcal{X}) \le c\rho^{-d}$ for each $\rho > 0$, where c is the *covering constant*. For a general Euclidean space R^d, the covering dimension is the geometric dimension d.

We also use another concept ρ-packing $\mathcal{F}_\rho^{pac}$ of $\mathcal{X}$ as a subset of $\mathcal{X}$, such that any two points in $\mathcal{F}_\rho^{pac}$ have a distance at least ρ, i.e., $\mathcal{F}_\rho^{pac} \subseteq \mathcal{X}$, and $s(x, y) \ge \rho, \forall x, y \in \mathcal{F}_\rho^{pac}$. The ρ-packing number $N_\rho^{pac}(\mathcal{X})$ of $\mathcal{X}$ and is defined as the largest size of $\mathcal{F}_\rho^{pac}$. It is easy to see that $N_\rho^{pac}(\mathcal{X}) \le N_\rho^{cov}(\mathcal{X})$, since each element in $\mathcal{F}_\rho^{cov}$ can at most cover one point in $\mathcal{F}_\rho^{pac}$ according to the definition. We will use ρ-packing number of the context/item space to prove the proposed theorems in this paper.

Appendix B Proof of Theorem 1

First, we prove that the item-cluster tree fulfills the exponential tree metric. For a cluster $C_{l,k}$ at layer l, the distance between any item in that cluster and the representative point is not greater than $\frac{b^l}{1+b}$. To validate this, we compute $s_I(i, i_{l,k})$ for any item $i \in C_{l,k}$. For $l = L$, this holds obviously because $C_{l,k}$ is a single item-cluster. We assume that for any $l + 1$, this holds. Then for l, we have

$$\begin{aligned} &s_I(i, i_{l,k}) \le s_I(i_{l+1,k'}, i_{l+1,k''}) + s_I(i, i_{l+1,k''}) \\ &\le \tfrac{1-b}{1+b}b^l + \tfrac{b^{l+1}}{1+b} \le \tfrac{b^l}{1+b}, \end{aligned} \tag{1}$$

where $i_{l+1,k'} = i_{l,k}$ is the representative point of cluster $C_{l+1,k'}$ such that item $i_{l,k} \in C_{l+1,k'}$, and $i_{l+1,k''}$ is the representative point of cluster $C_{l+1,k''}$ such that item $i \in C_{l+1,k''}$. The first inequality of (1) is due to the triangle inequality of $s_I(\cdot)$. The second inequality of (1) is due to the construction of the item-cluster tree of the CON-TREE algorithm and the hypothesis. According to mathematical induction, we have

$$s_I(i, i_{l,k}) \le \frac{b^l}{1+b}, \forall i \in C_{l,k}. \tag{2}$$

Therefore, we can prove that for any $i, j \in C_{l,k}$:

$$\begin{aligned} &s_I(i, j) \le s_I(i_{l+1,k'}, i_{l+1,k''}) + s_I(i, i_{l+1,k'}) \\ &+s_I(j, i_{l+1,k''}) \le \tfrac{1-b}{1+b}b^l + \tfrac{b^{l+1}}{1+b} + \tfrac{b^{l+1}}{1+b} = b^l, \end{aligned} \tag{3}$$

where $i_{l+1,k'}$ is the representative point of cluster $C_{l+1,k'}$ such that $i \in C_{l+1,k'}$, and $i_{l+1,k''}$ is the representative point of cluster $C_{l+1,k''}$ such that $j \in C_{l+1,k''}$. The first inequality of (3) is due to the triangle inequality of $s_I(\cdot)$. The second inequality of (3) is due to the construction of the item-cluster tree of the CON-TREE algorithm and the inequality (2).

Next, we prove the second part of the theorem. According to the CON-TREE algorithm, the distance between two representative points at layer l is at least $\frac{(1-b)b^l}{1+b}$. Hence, according to the covering dimension of the item space, the maximum number of representative points at layer l is at most

$$c_I[\frac{(1-b)b^l}{1+b}]^{-d_I} = c_I(\frac{1+b}{1-b})^{d_I}(\frac{1}{b})^{ld_I}, \tag{4}$$

where c_I is the covering constant for item space. Therefore, Theorem 1 follows.

Appendix C Proof of Theorem 2 and Theorem 3

For notational simplicity, let us define $\mu_t^* = \arg\max_{i \in \mathcal{I}} \mu_{i,t}$ and $\mu_{l(t),t}^* = \mu_{l(t),k_t^*,t}$. We also define the suboptimal cluster set as $\mathcal{S}(t) = \{k \in \mathcal{K}_{l(t)} : \mu_{l(t),t}^* - \mu_{l(t),k,t} > 4L_C\rho_t\}$, and define the clusters other than the optimal and suboptimal clusters as the *near optimal clusters*. Then the regret $R_{ACR}(T)$ can be expressed as $R_{ACR}(T) = R_{ACR,s}(T) + R_{ACR,n}(T) + R_{ACR,u}(T)$, and the regret $IR_{ACR}^s(t)$ can be expressed as $IR_{ACR}^s(t) = IR_{ACR,s}^s(t) + IR_{ACR,n}^s(t) + IR_{ACR,u}^s(t)$. The first terms $R_{ACR,s}(T)$ and $IR_{ACR,s}^s(t)$ are caused by suboptimal cluster selection; the second terms

L. Song, T. Cem, and M. van der Schaar are with Department of Electrical Engineering, UCLA. Email: songlinqi@ucla.edu, cmtkn@ucla.edu, mihaela@ee.ucla.edu.

$R_{ACR,n}(T)$ and $IR^s_{ACR,n}(t)$ are caused by near optimal cluster selection; the third terms $R_{ACR,u}(T)$ and $IR^s_{ACR,u}(t)$ are caused by selecting a cluster instead of an item.

(a) We first consider $R_{ACR,s}(T)$ and $IR_{ACR,s}(t)$. We consider the instantaneous regret caused by suboptimal cluster $C_{l,k}$ selection in epoch l:

$$\begin{aligned} IR^s_{l(t),k}(t) = E[\Delta_{l(t),k,t} I\{\pi(t) = (l,k), \Delta_{l(t),k,t} \\ > 4L_C\rho_t\} | u_t \in \mathcal{U}_s] \leq IR^{s,1}_{l(t),k}(t) + IR^{s,2}_{l(t),k}(t), \end{aligned} \tag{5}$$

where $\Delta_{l(t),k,t} = \mu^*_{l(t),t} - \mu_{l(t),k,t}$, $IR^{s,1}_{l(t),k}(t) = \Pr\{I^s_{l(t),k}(t) > I^s_{l(t),k^*_t}(t),\ \Delta_{l(t),k,t} > 2L_C\rho_t + 2\sqrt{\frac{A_s \ln t}{N_{l(t),k,t}}}\}$, and $IR^{s,2}_{l(t),k}(t) = E[\Delta_{l(t),k,t} I\{I^s_{l(t),k}(t) > I^s_{l(t),k^*_t}(t), 4L_C\rho_t < \Delta_{l(t),k,t} \leq 2L_C\rho_t + 2\sqrt{\frac{A_s \ln t}{N_{l(t),k,t}}}\}]$.

1) For the first term $IR^{s,1}_{l(t),k}(t)$, we have

$$\begin{aligned} & IR^{s,1}_{l(t),k}(t) \leq \Pr\{I^s_{l(t),k} > \mu_{l(t),k,t} + L_C\rho_t + 2\sqrt{\tfrac{A_s \ln t}{N_{l(t),k,t}}}\} \\ & + \Pr\{I^s_{l(t),k^*_t} < \mu^*_{l(t),t} - L_C\rho_t\} \\ & \leq \Pr\{\bar{r}_{l(t),k,t} > E[\bar{r}_{l(t),k,t}] + \sqrt{\tfrac{A_s \ln t}{N_{l(t),k,t}}}\} \\ & + \Pr\{\bar{r}_{l(t),k^*_t,t} < E[\bar{r}_{l(t),k^*_t,t}] - \sqrt{\tfrac{A_s \ln t}{N_{l(t),k^*_t,t}}}\} \leq 2t^{-2A_s}, \end{aligned} \tag{6}$$

where the last inequality is due to the Chernoff-Hoeffding bound with $N_{l(t),k,t}$ and $N_{l(t),k^*_t,t}$ samples; the second inequality is due to the fact that $E[\bar{r}_{l(t),k,t}] \leq \mu_{l(t),k,t} + L_C\rho_t$ and $E[\bar{r}_{l(t),k^*_t,t}] \geq \mu^*_{l(t),t} - L_C\rho_t$; and the first inequality holds because one of the following two must holds when $\Delta_{l(t),k,t} > 2L_C\rho_t + 2\sqrt{\frac{A_s \ln t}{N_{l(t),k,t}}}$.

$$I^s_{l(t),k} > \mu_{l(t),k,t} + L_C\rho_t + 2\sqrt{\frac{A_s \ln t}{N_{l(t),k,t}}}; \tag{7}$$

$$I^s_{l(t),k^*_t} < \mu^*_{l(t),t} - L_C\rho_t. \tag{8}$$

Otherwise, we have the following:

$$\begin{aligned} & I^s_{l(t),k} \leq \mu_{l(t),k,t} + L_C\rho_t + 2\sqrt{\tfrac{A_s \ln t}{N_{l(t),k,t}}} \\ & \leq \mu^*_{l(t),t} - L_C\rho_t \leq I^s_{l(t),k^*_t}. \end{aligned} \tag{9}$$

2) For the second term $IR^{s,2}_{l(t),k}(t)$, we have $IR^{s,2}_{l(t),k}(t) \leq IR^{1,2}_{l(t),k}(t)$, for $s \neq 1$. We first bound the instantaneous regret of the type 1 user. We choose the ball $B(x_t,\rho_t)$ and $t_{last} \leq t$ is the last time that $(l(t),k)$ is selected in epoch l. Then the aggregate regret due to suboptimal cluster k selection in ball $B(x_t,\rho_t)$:

$$\begin{aligned} & E\textstyle\sum_{\tau\in\mathcal{T}(B(x_t,\rho_t))} IR^{1,2}_{l(t),k}(\tau) I\{\pi(\tau) = (l(t),k)\} \\ & \leq E\textstyle\sum_{\tau\in\mathcal{T}(B(x_t,\rho_t))} (\Delta_{l(t),k,t_{last}} + 2L_C\rho_{t_{last}}) I\{\pi(\tau) = \\ & (l(t),k), 4L_C\rho_\tau < \Delta_{l(t),k,\tau} \leq 2L_C\rho_\tau + 2\sqrt{\tfrac{A_1 \ln \tau}{N_{l(t),k,\tau}}}\} \\ & \leq (\Delta_{l(t),k,t_{last}} + 2L_C\rho_{t_{last}}) N_{l(t),k,t_{last}} \\ & \leq \tfrac{36A_1 \ln t_{last}}{(\Delta_{l(t),k,t_{last}} + 2L_C\rho_{t_{last}})} \leq \tfrac{6A_1 \ln t}{L_C\rho_t}, \end{aligned} \tag{10}$$

where the third inequality holds because $L_C\rho_{t_{last}} \leq \sqrt{\frac{A_1 \ln t_{last}}{N_{l(t),k,t_{last}}}}$ and $\Delta_{l(t),k,t_{last}} + 2L_C\rho_{t_{last}} \leq 6\sqrt{\frac{A_1 \ln t_{last}}{N_{l(t),k,t_{last}}}}$ hold, which implies that $N_{l(t),k,t_{last}} \leq \frac{36*A_1 \ln t_{last}}{(\mu^*_{l(t),t_{last}} - \mu_{l(t),k,t_{last}} + 2L_C\rho_{t_{last}})^2}$.

Since $IR^{1,2}_{l(t),k}(\tau) \leq IR^{1,2}_{l(t),k}(\tau')$ for time $\tau \leq \tau'$ in the same epoch l according to (10), we can use $IR^{1,2}_{l(t),k}(2^l)$ to bound the instantaneous regret of time slots in epoch l. Hence, the regret $IR^{1,2}_{l(t),k}(t)$ can be bounded by

$$\begin{aligned} & IR^{1,2}_{l(t),k}(t) \leq IR^{1,2}_{l(t),k}(2^l) \\ & \leq \tfrac{6A_1 \ln t}{[2^l - (2^{l-1} - 1)]L_C\rho_t} \leq \tfrac{6A_1 \ln t}{2^{l-1} L_C\rho_t}, \end{aligned} \tag{11}$$

where $2^{l-1} - 1$ is the total time periods before epoch l-1, and 2^l is the first time period of epoch l. Since for period t in epoch l, we have $t \leq 2^{l+1} - 1$, then $IR^{1,2}_{l(t),k}(t)$ can be bounded by $IR^{1,2}_{l(t),k}(t) \leq \frac{24A_1 \ln t}{tL_C\rho_t}$.

The maximum number of balls with radius ρ_t that can cover the whole context space is bounded by $c_C(\rho_t)^{-d_C}$, and the number of clusters in epoch l is $K_{l(t)}$, then the instantaneous regret $IR^1_{ACR,s}(t)$ can be bounded by

$$IR^1_{ACR,s}(t) \leq K_{l(t)} 2t^{-2A_1} + \frac{24A_1 K_{l(t)} c_C \ln t}{L_C} t^{\frac{1-\alpha}{d_C} - \alpha}.$$

We then bound the instantaneous regret $IR^{s,2}_{l(t),k}(t)$ of type s users. We first introduce the concepts of virtual time for type s user at time t. We define the virtual time with respect to type $s'(s' < s)$ as $t_{s'} = \min\{\tau : \frac{A_{s'} \ln(t-\tau)}{(\Delta_{l(t),k,t-\tau} + 2L_C\rho_{t-\tau})^2} < \frac{A_s \ln(t)}{(\Delta_{l(t),k,t} + 2L_C\rho_t)^2}, t - \tau \in \mathcal{T}(B(x_t,\rho_t))\}$. According to the fully exploration assumption, the event $[\mathcal{W}^s_{k,\tau}]^C$ will happen with probability 1 for some large enough τ. The virtual time can be interpreted as: if type s user selects the suboptimal cluster $(l(t),k)$ at time t (i.e., event $\mathcal{W}^s_{k,t}$ happens), then for the context arrivals in the ball $B(x_t,\rho_t)$, it occurs that type 1 user does not select $(l(t),k)$ from time $t-t_1$ to t; type 2 user does not select $(l(t),k)$ from time $t-t_2$ to t; …; and type $s-1$ does not select $(l(t),k)$ from time $t-t_{s-1}$ to t. If the last time event $[\mathcal{W}^s_{k,\tau}]^C$ happens is before $t - t_1 - 1$, the probability that event $\mathcal{W}^s_{k,t}$ happens is smaller than the probability when the last time event $[\mathcal{W}^s_{k,\tau}]^C$ happens is just $t - t_1 - 1$. Hence, we can compute the virtual time $t_{s'}$ and bound the regret $IR^{s,2}_{l(t),k}(t)$.

We approximate $t_{s'} = \min\{\tau : \frac{A_{s'} \ln(t-\tau)}{A_s \ln(t)} < (1 + \frac{\rho_{t-\tau}}{3\rho_t})^2\}$. Then the regret $IR^{s,2}(t)$ can be bounded by

$$\begin{aligned} & IR^{s,2}(t) = \textstyle\sum_{k\in\mathcal{S}(t)} IR^{s,2}_{l(t),k}(t) \\ & \leq K_{l(t)}[(1-p_1) + p_1(1-p_{B(x_t,\rho_t)}) + p_1 p_{B(x_t,\rho_t)} t^{-\eta}]^{t_1-t_2} \cdot [(1- \\ & p_1 - p_2) + (p_1+p_2)(1-p_{B(x_t,\rho_t)}) + (p_1+p_2)p_{B(x_t,\rho_t)} t^{-\eta}]^{t_2-t_3} \\ & \cdots [(1 - p_1 - \cdots p_{s-1}) + (p_1 + \cdots p_{s-1})(1 - p_{B(x_t,\rho_t)}) \\ & +(p_1 + \cdots p_{s-1}) p_{B(x_t,\rho_t)} t^{-\eta}]^{t_{s-1}-t_s} \\ & = K_{l(t)} \textstyle\prod_{s'=1}^{s-1} [1 - \sum_{s''=1}^{s'} p_{s''} p_{B(x_t,\rho_t)} (1 - t^{-\eta})]^{t_{s'} - t_{s'+1}}, \end{aligned}$$

where $p_{B(x_t,\rho_t)} = \int_{x\in B(x_t,\rho_t)} f(x)dx$ is the probability that the context arrive in ball $B(x_t,\rho_t)$. The three terms $1-p_1$, $p_1(1-p_{B(x_t,\rho_t)})$, and $p_1 p_{B(x_t,\rho_t)} t^{-\eta}$ represent the probability that the type 1 user does not arrive, the probability that the type 1 user arrives, but the context is not in $B(x_t,\rho_t)$, and the probability that type 1 user arrives with context in $B(x_t,\rho_t)$, but the cluster $C_{l(t),k}$ has not been selected.

For the long-term regret $R_{ACR,s}(T)$, we have

$$\begin{array}{l}
R_{ACR,s}(T) \leq \sum_{l=0}^{E-1} K_E \sum_{t=2^l}^{2^{l+1}-1} IR_{l,k}(t) + K_E \sum_{t=2^E}^{T} IR_{l,k}(t) \\
\leq E[2K_E \sum_{t=1}^{\infty} t^{-2A_s} I\{u_t \in \mathcal{U}_s\}] \\
\quad + \sum_{l=0}^{E} \frac{6A_1 K_l c_C (\rho_{2^{l+1}-1})^{-d_C} \ln(2^{l+1}-1)}{L_C \rho_{2^{l+1}-1}} \\
\leq 2K_E \sum_{s=1}^{S} p_s \sum_{t=1}^{\infty} t^{-2A_s} \\
\quad + \frac{6A_1 K_E c_C \ln 2}{L_C} \sum_{l=0}^{\lfloor \log_2 T \rfloor} (l+1)[2^{\frac{(1-\alpha)(1+d_C)}{d_C}}]^{l+1} \\
\leq 2K_E \sum_{s=1}^{S} p_s \sum_{t=1}^{\infty} t^{-2A_s} \\
\quad + \frac{24A_1 K_E c_C (1.386+\ln T) T^{(1-\alpha)(1+d_C)/d_C}}{L_C (2^{(1-\alpha)(1+d_C)/d_C}-1)} \\
+ \frac{48A_1 K_E c_C T^{(1-\alpha)(1+d_C)/d_C} \ln 2}{L_C (2^{(1-\alpha)(1+d_C)/d_C}-1)^2}
\end{array},$$

where the second inequality is due to the inequalities of (6) and (10).

(b) Since the regret $IR_{ACR,n}(t)$ is caused by near optimal cluster selections, according to the definition of near optimal clusters, $IR_{ACR,n}(t) \leq 4L_C\rho_t$. The long-term regret $R_{ACR,n}(T)$ can be bounded by

$$R_{ACR,n}(T) \leq \sum_{t=1}^{T} 4L_C\rho_t \leq \frac{4L_C d_C T^{\frac{1-(1-\alpha)}{d_C}}}{d_C - 1 + \alpha}, \quad (12)$$

where the second inequality holds because of the divergent series $\sum_{t=1}^{T} t^{-y} \leq T^{(1-y)}/(1-y)$, for $0 < y < 1$ [1].

(c) We consider the regret $R_{ACR,u}(T)$ and $IR_{ACR,u}(t)$. Since the maximum regret caused by the random selection of items in a cluster and the best item selection in that cluster is bounded by $L_I b^l$, the instantaneous regret $IR_{ACR,u}(t)$ is bounded by $2L_I b^l$ for a period in epoch l, namely

$$IR_{ACR,u}(t) \leq 2L_I b^{l(t)} \leq 2L_I b^{\log_2 t} = 2L_I t^{\ln b/\ln 2} \quad (13)$$

The regret $R_{ACR,u}(T)$ can be bounded by

$$R_{ACR,u}(T) \leq \sum_{t=1}^{T} 2L_I b^{l(t)} \leq \frac{2L_I T^{1+\ln b/\ln 2}}{1+\ln b/\ln 2} \quad (14)$$

where the second inequality holds by the divergent series [1].

Then Theorem 3 and 4 follow the analyses (a), (b), and (c).

References

[1] E. Chlebus, "An approximate formula for a partial sum of the divergent p-series," *Applied Mathematics Letters*, vol. 22, no. 5, pp. 732–737, 2009.