Show Me the Whole World: Towards Entire Item Space Exploration for Interactive Personalized Recommendations

Yu Song[1,2,3,5], Shuai Sun[1,2,3], Jianxun Lian[4], Hong Huang[1,2,3,†], Yu Li[5], Hai Jin[1,2,3], Xing Xie[4]
[1]National Engineering Research Center for Big Data Technology and System, Wuhan, China
[2]Service Computing Technology and Systems Laboratory, Wuhan, China
[3]Huazhong University of Science and Technology, Wuhan, China
[4]Microsoft Research Asia, Beijing, China
[5]Meituan Group, Beijing, China
yusonghust@gmail.com,{honghuang,hjin}@hust.edu.cn,{jianxun.lian,xingx}@microsoft.com,liyu65@meituan.com

ABSTRACT

User interest exploration is an important and challenging topic in recommender systems, which alleviates the closed-loop effects between recommendation models and user-item interactions. *Contextual bandit* (CB) algorithms strive to make a good trade-off between exploration and exploitation so that users' potential interests have chances to expose. However, classical CB algorithms can only be applied to a small, sampled item set (usually hundreds), which forces the typical applications in recommender systems limited to candidate post-ranking, homepage top item ranking, ad creative selection, or online model selection (A/B test).

In this paper, we introduce two simple but effective hierarchical CB algorithms to make a classical CB model (such as LinUCB and Thompson Sampling) capable to explore users' interest in the entire item space without limiting to a small item set. We first construct a hierarchy item tree via a bottom-up clustering algorithm to organize items in a coarse-to-fine manner. Then we propose a *hierarchical CB* (HCB) algorithm to explore users' interest on the hierarchy tree. HCB takes the exploration problem as a series of decision-making processes, where the goal is to find a path from the root to a leaf node, and the feedback will be back-propagated to all the nodes in the path. We further propose a *progressive hierarchical CB* (pHCB) algorithm, which progressively extends visible nodes which reach a confidence level for exploration, to avoid misleading actions on upper-level nodes in the sequential decision-making process. Extensive experiments on two public recommendation datasets demonstrate the effectiveness and flexibility of our methods.

∗This work is supported by the National Key Research and Development Program of China under Grant (No. 2020AAA0108501) and National Natural Science Foundation of China (No. 62172174).
† Hong Huang is the corresponding author.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
WSDM '22, February 21–25, 2022, Tempe, AZ, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9132-0/22/02…$15.00
https://doi.org/10.1145/3488560.3498459

CCS CONCEPTS

• **Information systems** → **Recommender systems**; • **Computing methodologies** → **Sequential decision making**.

KEYWORDS

Recommender System, Contextual Bandit, Interest Exploration

ACM Reference Format:
Yu Song, Shuai Sun, Jianxun Lian, Hong Huang, Yu Li, Hai Jin, Xing Xie. 2022. Show Me the Whole World: Towards Entire Item Space Exploration for Interactive Personalized Recommendations. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining (WSDM '22), February 21–25, 2022, Tempe, AZ, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3488560.3498459

1 INTRODUCTION

Recommender systems help users to easily find their favorite items from massive candidates. Typically, recommender models, such as collaborative filtering [15] and DeepFM [9], exploit users' historical behaviors to learn users' preference for future recommendations. Recommender systems with only exploitation models usually suffer from closed-loop effects [11]: users mostly only interact with the items recommended by the system; the system further consolidates users' profiles with their interacted items recommended by the deployed model. Therefore, as time goes on, the system will be biased to a small, exposed set of interests for each user and keep recommending a limited range of items to a same user.

Contextual multi-armed bandit algorithms, such as LinUCB [16], are classical methods that leverage side information to provide a good trade-off between exploration and exploitation, so that the closed-loop effect can be alleviated. Items are treated as arms and the recommender model is treated as an agent. Basically, at each round, the agent chooses one arm which has the biggest potential from K arm candidates, then receives a corresponding reward based on user-item interaction. The goal is to maximize the cumulative reward over T rounds. However, these algorithms hold a premise that K is small, so enumerating all arm candidates' scores and pick up the best one is feasible. The premise is true for a few scenarios where the candidates are naturally small, for example, homepage breaking news ranking, ads creative ranking and online model selection. In the scenario of general recommender systems, to fully explore users' potential interests and truly alleviate the closed-loop effect, the arms candidates are the entire item repository, which

usually contains millions or even billions of items. Classical bandit methods become infeasible due to the high computation cost of enumerating every one of the arms.

To address the challenge, we first propose a generic *hierarchical contextual bandit* (**HCB**) algorithm to efficiently explore the interests of users for large-scale recommendation scenarios. Tree structure is widely employed to partition the search space to reduce the computational cost [12, 26, 41, 42]. HCB uses a tree structure as the index for coarse-to-fine retrieval. For example, in e-commerce scenario, *"Apparel > Clothing > Women's Clothing > Dresses"* is a path from general apparel to women's dresses. Instead of using the category taxonomy of items, we utilize a bottom-up clustering method on item embeddings to organize items as a hierarchy tree, on which each node contains a group of semantically similar items. As a result, the number of items associated to each node on the hierarchy tree can be balanced, and users' collaborative behaviors (such as co-click relations) can be encoded to form the hierarchy tree. HCB leverages the hierarchical information and turns the interest exploration problem into a series of decision-making problems. Starting from the root of the tree, on each non-leaf node, HCB performs a bandit algorithm among the children arms to choose a child node until a leaf node is reached. Afterward, another bandit algorithm is responsible for recommending an item from the leaf node to the user and collecting her feedback as reward. The reward will be back-propagated along the path to adjust the estimation of users' interest towards the hierarchy tree.

The process of HCB is like a *depth-first search* (DFS) idea. However, selecting a path in this DFS manner may cause new uncertainties, especially for a deep tree. First, if the selection of the parent node is misleading, all the subsequent choices will be impacted, which we call the *error propagation*. Second, since user interests are usually diverse, it is possible that the user is interested in many child nodes located in different parts of the tree. Therefore, we further propose a *progressive HCB* (**pHCB**) algorithm to reduce uncertainties and enhance the capacity of recommendation. Like the process of *breadth-first search* (BFS), pHCB explores items in an adaptive top-down manner. On the one hand, it gradually maintains a limited number of nodes as a receptive field. If one node has been explored multiple times and the user's interest on this node has been verified, the node's children nodes will be included to the receptive filed while the current node will be removed. On the other hand, pHCB learns user interests of different aspects by performing a bandit algorithm with visible nodes in the receptive field as arms. Consequently, the pHCB avoids greedily selecting only one node at each level to improve the HCB. To summarize, we make the following contributions:

- We highlight the importance of exploring users' interests in the entire item space to truly alleviate the closed-loop effect in personalized recommender systems. To the best of our knowledge, it is the first attempt to implement CB models on millions of items.
- Two simple yet effective algorithms, i.e. HCB and pHCB, are proposed to explore potential interests of users efficiently through a hierarchy item tree.
- We conduct experiments on two large-scale recommendation datasets. Results show the superiority of HCB and pHCB over baselines, as well as the flexibility to integrate with different exploration methods such as LinUCB, Thompson Sampling, and ϵ-greedy. In addition, we design an experiment to verify that with the exploring mechanism, both HCB and pHCB can effectively alleviate the closed-loop effects in recommender systems and learn better user profiles in the long term.

2 RELATED WORK

It is the first work to study entire space user interest exploration. Our work is relevant to two lines of research, and we will review them separately.

2.1 Contextual Bandit Algorithms

Contextual bandit algorithms aims to seek a balance between exploration and exploitation, which have been used in several applications, such as recommender systems [21], dynamic pricing [22], quantitative finance [31]. [4] reviews the existing practical applications of contextual bandit algorithms. By assuming the payoff model is linear, LinUCB [16] and Thompson Sampling [2] are two representative methods for solving contextual bandit problems. Beyond them, a variety of algorithms have been proposed to optimize the performance or learning speed. For example, ConUCB [40] introduces conversations between the agent and users to ask whether the user is interested in a certain topic occasionally. HATCH [39] considers the resource consumption of exploration and proposes a strategy to conduct bandit exploration with budget limitation. S-MAB [6] considers two aspects, one is to maximize the cumulative rewards and the other is to decide how many arms to be pulled so as to reduce the exploration cost. GRC [37] develops a graph regularized cross model to leverage the non-linearity of neural networks for better estimating the rewards. Different from them, our work commits to efficiently explore user interests in the entire space, rather than from a small subset of items.

2.2 Cluster-of-Bandit Algorithms

In the past few years, cluster-of-bandit algorithms have attracted the attention of some scholars. Generally, cluster-of-bandit algorithms aim to model the dependency since the items or users are always related to each other. As a result, cluster-of-bandit algorithms achieve better cumulative rewards than traditional contextual bandit algorithms due to knowledge sharing. For example, CLUB [8], DYNUCB [23], CAB [7], and COFIBA [17] assign users with similar interests into a same subset to make decisions together, thus it makes contributions to accelerate the learning speed. Different from them, this paper focuses on modeling the item dependency. ICTRTS and ICTRUCB [35] explicitly model the item dependencies via clustering of arms, but they are only designed for context-free bandits. Similarly, [25] uses a taxonomy structure to exploit arm dependencies with context-free bandits. Considering that context-free bandits cannot utilize the abundant side information for making decisions, their exploration ability has yet to be improved. HMAB [34] leverages a tree-structured hierarchy constructed by domain experts to design a hierarchical multi-armed bandit algorithm for online IT ticket automation recommendation. However, domain knowledge is hard to collect and HMAB can not be applied to large-scale recommender systems because it needs to traverse all the paths in the tree. Moreover, HMAB aims to learn latent parameters for the

Table 1: A collection of notations

Notation	Description
$\mathcal{U}$	User set, $\mathcal{U} = \{u^{(1)}, u^{(2)}, \cdots, u^{(M)}\}$
$\mathcal{A}$	Arm set, $\mathcal{A} = \{a^{(1)}, a^{(2)}, \cdots, a^{(K)}\}$
$\mathcal{I}$	Item set, $\mathcal{I} = \{i^{(1)}, i^{(2)}, \cdots, i^{(N)}\}$
$\mathcal{H}$	The hierarchy tree for item set partition.
$Ch(n)$	The set of child nodes of node n
$Pa(n)$	The parent node of node n
$I(n)$	The set of items mounted on node n
$\mathcal{V}_u(t)$	The receptive field of user u at the t-th round
$\boldsymbol{X}_a$	The (static) embedding features of arm a, $\boldsymbol{X}_a \in \mathbb{R}^{d\times 1}$
η	The Gaussian noise of reward, $\eta \sim \mathcal{N}(0, \sigma^2)$
$i_\pi(t)$	The selected arm by policy π at the t-th round
$r_\pi(t)$	Reward of policy π at the t-th round
$\boldsymbol{\theta}_u$, $\boldsymbol{\theta}_u^{(l)}$	Learnable parameter of user u. A superscript indicates the user parameter is for arms at level l on $\mathcal{H}$. $\boldsymbol{\theta}_u$, $\boldsymbol{\theta}_u^{(l)} \in \mathbb{R}^{d\times 1}$

nodes in the hierarchy tree, which is totally different from our goal of exploring users' latent interests. Distributed bandit algorithms, such as DCCB [14] and DistCLUB [20], aim to speed up the computation by distributing the workloads in parallel. However, these methods do not address the issue of searching from tremendous items, the computational cost is still too expensive for responding users' requests in an online manner (for example, how to response 100 users' concurrent requests within 10 milliseconds in a scenario involving one million items). In summary, compared with existing cluster-of-bandit algorithms, our HCB and pHCB algorithms leverage a bottom-up clustering method to build a hierarchical tree of items, then explore users' potential interests in the entire space of items based on the item hierarchy.

3 PRELIMINARY AND PROBLEM

We start by introducing the multi-armed bandit algorithms and the motivations of this paper. For better readability, we summarize most of the notations used throughout the paper in Table 1.

3.1 UCB for Recommender Systems

The recommender system is regarded as an agent, where there are M users and N items. At each round $t = 1, 2, \cdots, T$ of interactions, given a user u, the agent recommends an item $i_\pi(t)$ to the user according to a policy π. Then the agent receives a feedback $r_\pi(t)$ from the user, for example, if the user clicks on the item $i_\pi(t)$, $r_\pi(t)$ is 1 and otherwise it is 0. The optimal policy is denoted by π^*. The goal is to learn a good policy π, so that the cumulative regret over T rounds, which is defined as below, is minimized:

$$\boldsymbol{R}(T) = \sum_{t=1}^{T} (\boldsymbol{E}[r_{\pi^*}(t)] - \boldsymbol{E}[r_\pi(t)]) \tag{1}$$

In practice, due to the absence of the optimal policy π^*, we maximize the cumulative reward $\sum_{t=1}^{T} \boldsymbol{E}[r_\pi(t)]$ instead, because maximizing cumulative reward equals to minimizing cumulative regret [17, 34, 38].

As the core of bandit algorithms is to find an optimal trade-off between exploitation (to recommend fully based on user profiles learned from user interaction history) and exploration (find out the new items which user may potentially love better), so that users with diverse new interests have a certain chance to expose, meanwhile the system won't waste too many resources on items that users are not interested in. Let's consider the (user-centric) LinUCB [16] algorithm. Each item is regarded as an arm. At t-th round, when receiving a user visit request, the agent selects an arm $a_\pi(t)$ by:

$$a_\pi(t) = \arg\max_{a\in\mathcal{A}_t} R_a(t) + C_a(t) \tag{2}$$

The policy π of LinUCB is a linear function between the feature vector $\boldsymbol{x}_a$ and user hidden parameter θ_u, where the estimated reward is $R_a(t) = \boldsymbol{\theta}_u^T \boldsymbol{x}_a + \eta$, η is a Gaussian random variable representing environmental noise, whose mean is zero and variance is $\sigma^2 \leq 1$. The upper bound $C_a(t)$ measures the uncertainty of the reward estimation. The key point lies in how to determine the parameter $\boldsymbol{\theta}_u$ and the upper bound $C_a(t)$. With LinUCB, we have:

$$\boldsymbol{\theta}_u = \left(\boldsymbol{D}_t{}^T \boldsymbol{D}_t + I_d\right)^{-1} \boldsymbol{D}_t{}^T \boldsymbol{r}_t \tag{3}$$

$$C_a(t) = \alpha \sqrt{\boldsymbol{x}_a^T (\boldsymbol{D}_t^T \boldsymbol{D}_t + \boldsymbol{I}_d)^{-1} \boldsymbol{x}_a} \tag{4}$$

where $\boldsymbol{D}_t \in \mathbb{R}^{t\times d}$ is the matrix of interacted arms' features up to time t, α is a hyper-parameter to control the probability that the bound $C_a(t)$ holds, $\boldsymbol{r}_t \in \mathbb{R}^t$ is the user response vector up to time t.

3.2 The Challenges

However, as revealed in Eq.(2), LinUCB needs to enumerate and calculate the score for every arm and then select the best one. In a modern recommender system, the number of items is usually very large (millions or even billions), which makes it impossible to calculate scores for all items. Thus, in the research community, a typical setting for existing literature is to randomly sample a small number K (such as 50) arms from the entire N arms at time t, and perform LinUCB on this small arm set $\mathcal{A}_t$; in industry, the bandit algorithms can only be applied to scenarios whose candidate pool is small, such as post-ranking stage of a recommender system, homepage most popular item ranking, ad creative ranking, etc. We argue that in order to fully explore users' potential interest, it is better to place the bandit module in the item retrieval stage (aka the recall stage) of a recommender system, where the candidate pool is the entire item set. Otherwise, in the post-ranking stage of a recommender system, the candidates are actually proposed by recommendation models and are strongly related to users' past behaviors. Thus, it is less meaningful to explore users' interest in the latter stages of a recommender system. To fully alleviate the closed-loop effect, in this paper, we advocate to explore users' interest in the entire space of item repository. However, to the best of our knowledge, there is no work studying how to make the bandit algorithm like LinUCB fit for a large candidate set.

To address the challenge, we propose to use a tree structure to partition the entire item space into multiple sub-spaces and build the hierarchical dependencies among items, to accelerate the exploration. Formally, we define the Framework 1:

Framework 1. ***Tree-based Exploration*** *The entire item set can be organized as a hierarchical tree structure $\mathcal{H}$, where nodes are*

linked to a subset of items that share some common topics or user interests, and nodes moving from top to bottom reflect the topics/interests partition being coarse-to-fine. During the tree-based exploration, we will first select a node according to some mechanism, then select an item from the candidates linked to this node. The user feedback on the selected item will not only update the item-wise user preference estimation, but also update the node-wise user preference estimation along the hierarchical path.

4 METHODOLOGY

4.1 Tree Structure Construction

The tree structure plays a significant role in designing hierarchical bandit algorithms. Item category taxonomy can serve as the hierarchy tree. However, due to the imbalanced number of items under different categories and lack of leveraging of users' collective behaviors, simply using the category taxonomy may lead to suboptimal performance, which is verified in Section 5. In view of this, we first learn item embeddings based on item content and user co-click behaviors, then design a bottom-up clustering method based on K-Means clustering algorithm [18] to form a hierarchy tree for modeling dependencies among items.

Specifically, to construct a tree structure with L levels, at first, N items are clustered into k_L different subsets based on the similarity of item embeddings. We treat each subset as a new node on the tree, with an embedding vector being the average of all item embeddings belonging to this node. Afterward, these k_L nodes will be further clustered into k_{L-1} different subsets using K-Means and each subset will be treated as a new node on the tree, forming a parent-children relation. This step will be repeated several times until the depth of the tree structure researches L. As a result, the constructed tree structure, denoted by $\mathcal{H}$, contains $\{k_0, k_1, k_2, \cdots, k_L\}$ nodes at each level, where $k_0 = 1$ because only a root node appears at the first level. Intuitively, items within the same node are more similar to each other, thus the clustering results reflect the dependencies among items. In $\mathcal{H}$, only the root node does not have parent node, and leaf nodes have no children nodes.

4.2 Hierarchical Contextual Bandit

In this section, we introduce the proposed *hierarchical contextual bandit* (HCB) algorithm, which empowers a base bandit model to explore over the entire space of item repository. HCB can be generalized to different bandit models, without loss of generality, we take LinUCB as the base model to explain the algorithm.

There are two types of arms: nodes on the hierarchy tree $\mathcal{H}$ and items mounted to the leaf nodes. Each node on $\mathcal{H}$ represents a certain group of items. The feature vector of a leaf node is the average pooling of items mounted to it, and a non-leaf node's feature vector is the average pooling of its children nodes' feature vectors. The HCB algorithm makes decisions sequentially, starting from the root node to a leaf node. At any non-leaf node $n^{(l)}(t)$ at l-th level, the policy π selects one of the child nodes from $Ch(n^{(l)}(t))$ by assuming that the expected reward of an arm is in linear relation with its feature vector, i.e., ${\boldsymbol{\theta}_u^{(l)}}^T \boldsymbol{X}_n$, where $\boldsymbol{\theta}_u^{(l)}$ is the latent parameters of a given user u towards the nodes at level l, $\boldsymbol{D}^{(l)} \in \mathrm{R}^{m \times d}$ is the matrix comprised of interacted items at l-th level, each row of $\boldsymbol{D}^{(l)}$ represents an item's feature vector. Applying ridge regression to the training samples to estimate the coefficients, we have:

$$\boldsymbol{\theta}_u^{(l)} = \left({\boldsymbol{D}^{(l)}}^T \boldsymbol{D}^{(l)} + \boldsymbol{I}\right)^{-1} {\boldsymbol{D}^{(l)}}^T \boldsymbol{r}^l \tag{5}$$

where $\boldsymbol{I}$ is an identity matrix and $\boldsymbol{r}^l$ is the vector of historical rewards at node level l. LinUCB also considers confidence interval to better estimate the arm payoff. Let $\boldsymbol{A}^{(l)} = {\boldsymbol{D}^{(l)}}^T \boldsymbol{D}^{(l)} + \boldsymbol{I}$. According to [33], with probability $1 - \delta$, the upper bound is:

$$\left| {\boldsymbol{\theta}_u^{(l)}}^T \boldsymbol{X}_n - \boldsymbol{E}[r_{\pi^*} | \boldsymbol{X}_n] \right| \leq \alpha \sqrt{\boldsymbol{X}_n^T {\boldsymbol{A}^{(l)}}^{-1} \boldsymbol{X}_n} \tag{6}$$

for any $\delta > 0$ and $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$. In this way, the LinUCB algorithm tends to select an arm with:

$$n^{(l+1)}(t) = \underset{n \in Ch(n^{(l)}(t))}{\arg\max} \left({\boldsymbol{\theta}_u^{(l)}}^T \boldsymbol{X}_n + \alpha \sqrt{\boldsymbol{X}_n^T {\boldsymbol{A}^{(l)}}^{-1} \boldsymbol{X}_n} \right) \tag{7}$$

If policy π recommends $i_\pi(t)$ to a given user and receives the reward $r_\pi(t)$, similar as [41], then each node on $Path(root \rightarrow n^{(L)}(t))$ also receives the same reward $r_\pi(t)$. Therefore, the rewards of all selected nodes can be obtained, we can update the learnable parameters $\{\boldsymbol{\theta}_u^{(0)}, \boldsymbol{\theta}_u^{(1)}, \boldsymbol{\theta}_u^{(2)}, \cdots, \boldsymbol{\theta}_u^{(L)}\}$ at each level (where $\boldsymbol{\theta}_u^{(0)}$ means the parameter towards item arms, the other $\boldsymbol{\theta}_u^{(*)}$ means parameters towards node arms), which can be formulated as:

$$\begin{aligned} \boldsymbol{A}^{(l)} &\leftarrow \boldsymbol{A}^{(l)} + \boldsymbol{X}^{(l)} {\boldsymbol{X}^{(l)}}^T \\ \boldsymbol{b}^{(l)} &\leftarrow \boldsymbol{b}^{(l)} + r_\pi(t) \boldsymbol{X}^{(l)} \\ \boldsymbol{\theta}_u^{(l)} &\leftarrow {\boldsymbol{A}^{(l)}}^{-1} \boldsymbol{b}^{(l)} \end{aligned} \tag{8}$$

where $\boldsymbol{A}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are initialized as d-dimensional identity matrix and zero vector respectively. $\boldsymbol{X}^{(l)}$ is the contextual embedding of the selected node at l-th level.

The pseudo-code of HCB is provided in Algorithm 1. To illustrate HCB, we offer a toy example shown in Figure 1. It has three layers in the hierarchy tree. The agent makes three decisions sequentially,

Algorithm 1: The pseudo-code of HCB algorithm

Input: The tree structure $\mathcal{H}$ with depth as L, the total number of rounds T, hyper-parameter α.
Output: The policy π.

```
Initialize parameters of π;
for t = 1, 2, ··· ,T do
    receive a user u;
    set current node as the root of H;
    for l = 1, 2, ··· , L-1 do
        select a child node from current node with Eq. (7);
        set current node as the selected child node;
    end
    select an item from the set of items of current (leaf) node to user u with Eq. (2);
    receive the reward from user u;
    propagate rewards to all nodes in the path;
    according to Eq. (8), update parameters of π;
end
```

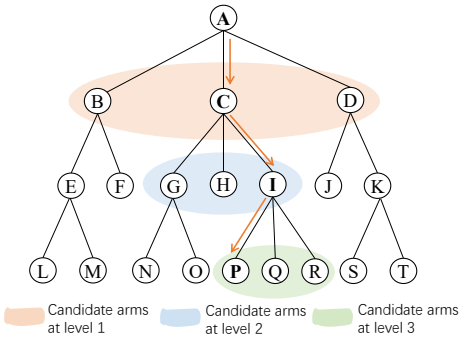


Figure 1: An illustration of HCB. The policy selects a path { A, C, I, P } from root to a certain leaf node.

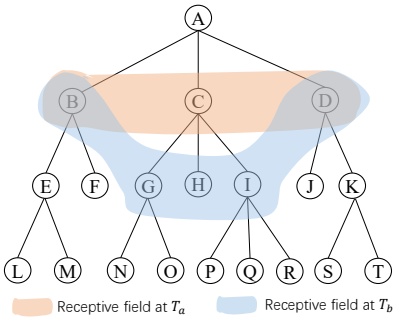


Figure 2: An illustration of pHCB. At round T_a, the receptive field consists of nodes B, C, and D. After several trials, at round T_b, node C meets the conditions of expansion, so the receptive field changes to nodes B, D, G, H and I.

and finally selects the path $\{A, C, I, P\}$. Then the agent will launch another bandit selection among the items mounted to the leaf node P. The reward on the selected item will impact the parameter estimation on the hierarchy tree $\{\boldsymbol{\theta}_u^{(0)}, \boldsymbol{\theta}_u^{(1)}, \boldsymbol{\theta}_u^{(2)}, \boldsymbol{\theta}_u^{(3)}\}$, by updating the reward history $\boldsymbol{r}^{(*)}$ and interaction history $\boldsymbol{D}^{(*)}$.

4.3 Progressive Hierarchical Contextual Bandit

The HCB learns the interests of each user via a sequential decision-making process and always select the item from the arriving leaf node, which may lead to two problems: (1) the decisions made in upper levels severely impact the scope of lower-level nodes. Once the policy makes a bad decision at a certain level, the rest selections are all sub-optimal. The issue is especially true when the tree hierarchy is deeper. We call this phenomenon *error propagation*; (2) Users may be interested in more than one child node, thus the greedy selection may fail to capture the comprehensive interests of users. Therefore, we further propose a *progressive hierarchical contextual bandit* (pHCB) algorithm for exploration in another manner on the tree. The main idea is that the policy continuously expands the receptive field from top to bottom according to the feedback obtained from historical exploration. We first give a definition of receptive field.

Definition 1. ***Receptive Field*** *is a personalized set of nodes representing the current potential interests for each user to explore. At the first round, the receptive field only consists of the root node (or is set with prior knowledge). With the exploration process progressing, the receptive field will be expanded (and reduced) when predetermined conditions are met in an adaptive top-down manner. The nodes in the receptive field are called visible nodes.*

In HCB, only the leaf node is associated with a set of items. In contrast, in pHCB we allow the policy to select a non-leaf node and then recommend an item from the item set associated with the non-leaf node. Hence, we have Definition 2 to define the item set of each non-leaf node.

Definition 2. *Given a non-leaf node n and the set of child nodes $Ch(n)$, the item set of node n will be the union of item sets of the nodes in $Ch(n)$, that is $I(n) = I(n_c^{(1)}) \cup I(n_c^{(2)}) \cup \cdots \cup I(n_c^{(k)})$ and $Ch(n) = \{n_c^{(1)}, n_c^{(2)}, \cdots, n_c^{(k)}\}$.*

At t-th round, the agent faces a user u whose receptive field is denoted as $\mathcal{V}_u(t)$. pHCB algorithm treats each node in $\mathcal{V}_u(t)$ as an arm, and selects the arm (denoted as $n(t)$) with the highest estimated reward according to Eq. (2). Next, another LinUCB is used to select an item $i_\pi(t)$ from $I(n(t))$ and collect feedback from u. pHCB directly selects an arm from the receptive field without performing sequential decision-making processes, which avoids the aforementioned concerns of HCB. If the number of items in $I(n(t))$ exceeds the single-round budget k, then pHCB will randomly sample k items from $I(n(t))$ to perform LinUCB exploration.

Here we offer an example in Figure 2 for illustrating the expanding process. Assuming at round T_a, the receptive field of the user u consists of three nodes: B, C, and D. In the next several rounds, if node C is selected multiple times and received several positive rewards, making it meet the conditions of expansion, its children nodes G, H, I will then be added into the receptive field to replace C. As a result, at round T_b, the receptive field includes nodes B, D, G, H, I. In this way, pHCB expands the receptive field from coarse to fine and gradually discovers the interests of users.

A critical mechanism of pHCB is how to expand the receptive field. Since the tree nodes are organized in different granularity, the nodes at top levels represent the coarse interests of users while the nodes at bottom levels depict specific interests of users. We want the receptive field be able to quickly converge to the leaf nodes, thus we set the expansion conditions as follows: for a non-leaf node n at the l-th level of $\mathcal{H}$, if (1) it has been selected at least $\lfloor q \log l \rfloor$ times, and meanwhile (2) the average reward on this node is larger than $p \log l$ ($0 \leq p \leq 1$), then we expand the receptive filed by replacing it with its children. q and p are hyper-parameters. The $\log l$ means that the nodes at a top-level are easier to be expanded than those at a low level. One can also design more flexible rules for expansion according to the actual application scenario. Overall, the pseudo-code of the pHCB is available in Algorithm 2.

5 EXPERIMENTS

5.1 Experimental Settings

5.1.1 ***Datasets.*** We perform experiments on two public recommendation benchmark datasets, basic statistics are shown in Table 2.

Algorithm 2: The pseudo-code of pHCB algorithm

Input: The tree structure $\mathcal{H}$ with depth as L, the total number of rounds T, the original receptive field $\mathcal{V}(0)$, hyper-parameters α, p and q.
Output: The policy π.

```
Initialize parameters of π;
for t = 1, 2, ··· ,T do
    receive a user u;
    select a node n from current receptive field V_u(t);
    select an item from I(n) to user u with Eq. (2);
    receive the reward from user u;
    if node n satisfies the expansion conditions then
        add the nodes in Ch(n) into the receptive field to
          replace node n;
    end
    update parameters of π;
end
```

Table 2: Datasets Overview

Dataset	#users	#items	# categories	# interactions
MIND	1,000,000	161,013	285	24,155,470
Taobao	987,994	4,162,024	9,439	100,150,807

- **MIND**[1] [36]: The MIND dataset is the largest public benchmark dataset for news recommendations so far, which is constructed from the click logs of Microsoft News. We use Sentence BERT [27] to train news embeddings from their contents, and adopt a GRU [24] as the user model to fine-tune news embeddings from the sequence of click logs.
- **Taobao**[2]: The Taobao dataset is constructed from user behaviors of Taobao for E-commerce recommendations. Similar to MIND dataset, we also utilize GRU to learn item embeddings from the sequence of user behavior logs.

5.1.2 ***Baselines***. We compare the proposed algorithms against the following related and competitive bandit algorithms:

- **LinUCB** [16] is a classical contextual bandit algorithm. It only works with item-level recommendation.
- **HMAB** [34] organizes arms into hierarchy tree purely by domain knowledge. It utilizes category information to model the dependencies among items. Then the algorithm selects a path from root node to a leaf node, and the leaf node is an item.
- **ICTRUCB** [35] formulates the item dependencies as the clusters on arms. Different from our methods, it does not consider the hierarchy.
- **ConUCB** [40] utilizes key-terms to organize items into different subsets to represent dependencies among items. The algorithm occasional converses with users and leverages conversational feedbacks on key-terms from users to accelerate the speed of bandit learning.

5.1.3 ***Our Methods and Variants***. Our goal is to propose a generic algorithm that can empower different bandit models to be more effective on large-scale item set exploration. Therefore, our main experiments contain two groups: first, with LinUCB as the base bandit model, to compare our algorithms (i.e., HUCB and pHUCB) with the aforementioned baselines (because most of the listed baselines are based on LinUCB); second, with three different base bandit models, including LinUCB, Thompson Sampling (TS) [2], and ϵ-greedy, to verify whether our algorithms are effective under different settings. In the second group of experiments, we also compare two variants of our models:

- **CB-Category**: It borrows the idea from HMAB [34] by using the prior knowledge, i.e. the category taxonomy, to assign items into different subsets. Each subset will be treated as an arm, the policy first selects a subset and then recommends an item from the subset to users with a base bandit algorithm.
- **CB-Leaf**: It is a variant of ICTRUCB [35]. Since the ICTRUCB is designed for context-free bandits, to keep a fair comparison, we also utilize K-Means clustering on item embeddings to assign items into different subsets. Here, we treat the leaf nodes of $\mathcal{H}$ as the clustering results. Each leaf node will be treated as an arm, the policy first selects a leaf node and then recommends an item from the node to users with a base bandit algorithm.

Here, *CB-* can take a value from {LinUCB, Thompson Sampling (TS), and ϵ-greedy}, our experiments are separated into three groups. For example, if the *CB-* model is LinUCB, then the involved variants are *LinUCB-Category*, *LinUCB-Leaf*, and our final models are *HUCB* and *pHUCB*. For all models, including our proposed ones and baselines, we set the maximum times of score-computing per round to 50 for a fair comparison. For example, in LinUCB, if the number of arm candidates is 1000, then originally we need to compute 1000 scores per round to select the best one in estimation, which exceeds our budget, so we randomly sample 50 arms from the 1000 arms and then perform the LinUCB on the small set. For hierarchical CB methods, the budget is evenly distributed to each level, e.g., for pHCB, we have two levels of bandit, then each level will get 25.

5.1.4 ***Evaluation***. We evaluate the performance of different bandit algorithms with *off-policy user simulator evaluation*. To reduce the biases of the simulation, we utilize the IPS estimator [10, 28, 29], which is a standard method used for *off-policy evaluation* of bandit algorithms[3]. IPS learns to re-weigh the training samples by the propensity score to learn an unbiased simulator. The simulator is trained on the whole data to make the best use of information. This evaluation enables us to compare the performance of candidate hypothetical policies without expensive online A/B tests. Specifically, the simulator learns the unbiased embeddings of users and items, and the reward of a user u towards an item i is derived from the inner product of their embeddings.

5.1.5 ***Reproducibility***. For MIND dataset, each item is represented by a 64-dimensional embedding vector. The dataset has been split for training/validation/test, only the click logs of the training set is used for learning item embeddings and building tree structures. The users without history logs or impression logs are removed. The click is treated as positive feedback and non-click is treated as negative feedback. The hierarchy tree structure is set to {1, 100, 10000}, which means there is 10000 leaf nodes, and only

[1] https://msnews.github.io/index.html
[2] https://tianchi.aliyun.com/dataset/dataDetail?dataId=649&userId=1
[3] https://groups.google.com/g/open-bandit-project

one layer of non-leaf nodes. For Taobao dataset, each item is represented by a 32-dimensional embedding vector. As the same method does in [41], we remove the users who have less than 10 behaviors or the items appearing less than 10 times. We randomly select 10000 users for testing and 1000 users for validation, and the behavior logs of the rest users are used for learning item embeddings. The click is treated as positive feedback and the negative feedback is generated via negativing sampling. The hierarchy tree structure is set to {1, 50, 5000, 50000}, so that number of items mounted to a leaf node is less than 100.

For the LinUCB-based algorithms, all learnable parameters are initialized as all zero matrices or vectors, and the hyper-parameter α is set as 0.5. The Gaussian prior is used to design Thompson Sampling-based algorithms for contextual bandit. For ϵ-greedy-based algorithms, ϵ is set as 0.05. With the help of validation set, the hyper-parameters q and p of pHCB and its variants are set as 10 and 0.1, respectively. Follow the setting in [40], we consider a general configuration in which at each round, the computational cost is limited as 50. Note here the arms can be nodes or items depending on different bandit algorithms. The learning rate for training GRU is 0.001, the hidden size is the same as the embedding size, and optimizing with Adam optimizer [13]. All algorithms are implemented with Python and PyTorch, and repeated 10 times to report the average performance. The code and processed datasets are released at https://github.com/CGCL-codes/HCB-pHCB for easy reproducibility.

Table 3: Cumulative rewards of different algorithms

Dataset	MIND			TaoBao		
Round	100	1000	2000	100	1000	2000
LinUCB	4.76	158.41	357.91	10.70	106.12	212.26
HMAB	4.41	217.40	520.97	10.74	120.89	255.22
ICTRUCB	5.60	300.83	709.05	10.98	135.99	290.33
ConUCB	7.47	188.50	409.23	13.04	273.86	584.84
HUCB	7.48	407.82	**918.38**	16.20	320.68	699.29
pHUCB	**7.93**	**419.74**	866.60	**16.51**	**391.36**	**806.66**

5.2 Experimental Results

5.2.1 ***Comparison with Baselines.*** Since all baseline algorithms are on the basis of LinUCB, we also choose LinUCB as the base algorithm for HCB and pHCB to keep a fair comparison. Note that HCB and pHCB can work with different base algorithms, and we discuss their generality in Sec. 5.2.2. We compare the cumulative rewards over 100/1000/2000 rounds[4] of different algorithms in Table 3, and the best results are presented in bold. As can be seen, our proposed algorithms, HUCB and pHUCB, outperform all baselines across different datasets consistently at different rounds, and pHUCB is generally better than HCB. For example, on the largest TaoBao dataset, at 100/1000/2000 rounds, the performance of pHUCB was improved by 54.3%, 268.7%, and 280% compared with LinUCB, respectively. Although HMAB, ICTRUCB, and ConUCB achieve higher cumulative rewards over LinUCB, there is still a considerable gap between these algorithms and ours. The superiority of pHUCB over HUCB further verifies that by expanding the receptive field in a progressive manner, the pHCB algorithm is able to better discover the comprehensive interests of users.

5.2.2 ***Flexibility and Variants Study.*** Next, we report the cumulative reward of our algorithms and their variants, in Figure 3, based on three different base bandit algorithms. From the experimental results, we have the following observations.

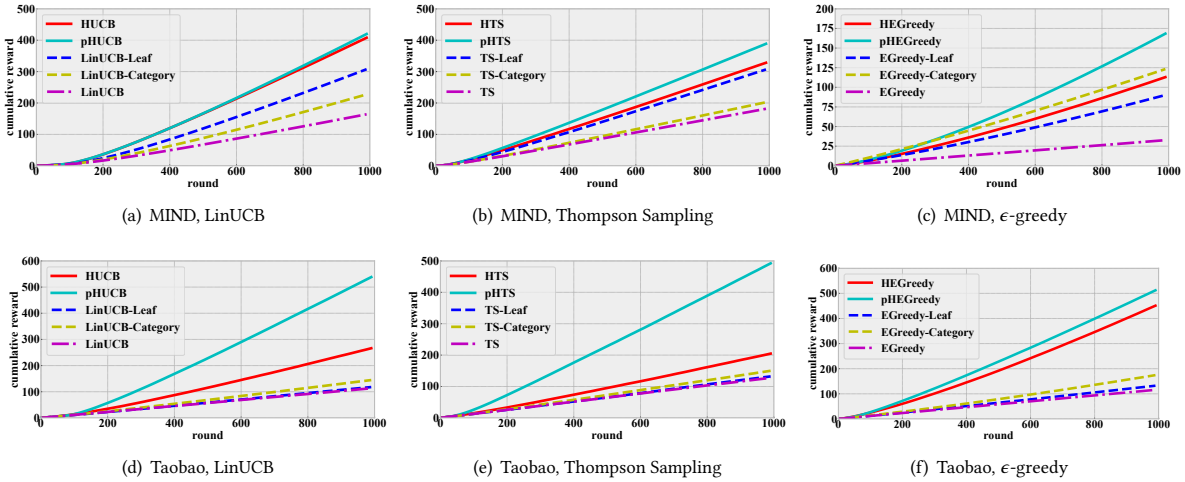
- Constructing item dependencies in the form of clusters indeed helps a lot in accelerating the exploration. This can be verified from that both our algorihtms (HCB and pHCB) and their variants (CB-category and CB-Leaf) outperform the corresponding base bandit model.
- HCB and pHCB achieve the highest cumulative rewards on both datasets in most of the cases, indicating that the proposed hierarchical algorithms are effective. As we can see, the performance of baseline algorithms has a noticeable gap between our proposed algorithms. This result is reasonable since our proposed methods introduce the hierarchy knowledge to take the item dependencies into consideration, which greatly improves the efficiency of exploration. Although the two variants, such as CB-Category and CB-Leaf, also organize items into different clusters, they fail to model the coarse-to-fine item dependencies as tree structures. Apart from that, the pHCB algorithm beats other methods, which verifies that the progressive exploration can adaptively discover the diverse interests of users with a receptive field.
- Our algorithms have strong flexibility. We have tested the performance with base models varying in {LinUCB, Thompson Sampling, ϵ-greedy}, the conclusions are consistent, which proves our frameworks can well generalize to various bandit algorithms.

5.2.3 ***Parameter Sensitivity.*** In this subsection, we study the impacts of hyper-parameters. Since the pHCB performs best in most of the cases, we particularly study the key hyper-parameter of it, i.e., q and p, which control the expansion conditions: q determines the number of trails at one arm and p indicates the threshold of average reward for expanding child nodes. To study their impacts, we take pHUCB as an example and vary q from $\{0, 5, 10, 15, 20, 25\}$ and p from $\{0, 0.05, 0.1, 0.15, 0.2, 0.25\}$ to see how the final rewards (at round 1000) will be affected.

As shown in Figure 4, different hyper-parameters have a noticeable influence on the cumulative reward of pHUCB algorithm over 1000 rounds. For MIND dataset, if q is too small or p is too large, the cumulative rewards become worse since the former makes the expansion conditions unreliable and the latter makes the receptive field difficult to expand. As for the Taobao dataset, the trend of impacts caused by q is similar to the MIND dataset. Meanwhile, the model is less sensitive with parameter p. Overall, from Figure 4 we learn that a suggested configuration is $p = 0.1$ and $q = 10$.

5.2.4 ***Alleviate Closed-Loop Effects.*** Typically, recommender models trained from historical logs are designed for exploitation purposes, which we denote as exploitation models. Such recommendation systems suffer from closed-loop effects [11] because they only learn users' interests from historical logs, but they do not have the ability to explore new interests of users. In contrast, contextual bandit algorithms are more effective to break the information

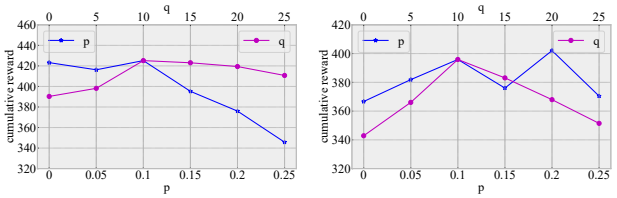
[4]Considering that the two datasets have different number of users and items, for better alignment, we denote *one round* as *one pass of all users receiving one recommended item.*



(a) MIND, LinUCB (b) MIND, Thompson Sampling (c) MIND, ϵ-greedy

(d) Taobao, LinUCB (e) Taobao, Thompson Sampling (f) Taobao, ϵ-greedy

Figure 3: Cumulative rewards of our algorithms and variants based on LinUCB, Thompson Sampling, and ϵ-greedy, on the MIND dataset and Taobao dataset, respectively.



(a) MIND (b) Taobao

Figure 4: Effect of hyper-parameters of pHUCB.

cocoons with exploration strategies. To verify that our model is able to explore the potential interests of users thus alleviate the closed-loop effects, we design an additional experiment as follows.

We select three exploitation models, i.e., Linear model, GRU, and Transformers [32] as baselines. These exploitation models are first pre-trained by the historical logs of existing users to get the deployed models. Then, we use exploitation models as well as our proposed models as "deployed models" to serve users. Specifically, in this stage, for each new user (whose logs are not used in the first pre-trained stage), we randomly sample only three clicked items as visible historical logs for generating her initial user embedding with a deployed model. Then we can recommend two hundred items to the user with a deployed model and collect the user's feedback. Note here the user embedding will be refreshed once the model receives positive feedback. This stage is performed for every deployed model respectively. The third stage is about evaluating the quality of impression logs produced by the deployed models. We utilize the collected historical logs together with all the rest historical logs of existing users (which are used in the first stage) as training samples to train an evaluating model (here we use the matrix factorization (MF) model as the evaluating model, each user and item will be mapped to an embedding vector. MF-based collaborative filtering method is one of the most popular models for personalized recommendations) and evaluate the trained model on the same test samples for a fair comparison. In order to prove the advantages of our bandit algorithms in exploring user interests, we select two hundred of test users with the most diversified interests as new users: we calculate the Gini impurity [19] of the historical items clicked by the user according to the category of items. Obviously, the larger the Gini impurity, the more diverse the interests of the user. The users of the training set are treated as existing users.

Table 4: Test LogLoss and AUC of different algorithms

Dataset	MIND		TaoBao	
Method	**LogLoss**	**AUC**	**LogLoss**	**AUC**
Linear	$1.679_{\pm 0.005}$	$0.703_{\pm 0.005}$	$0.693_{\pm 0.001}$	$0.530_{\pm 0.001}$
GRU	$1.759_{\pm 0.004}$	$0.686_{\pm 0.003}$	$0.688_{\pm 0.001}$	$0.535_{\pm 0.002}$
Transformer	$1.377_{\pm 0.008}$	$0.695_{\pm 0.006}$	$0.683_{\pm 0.001}$	$0.546_{\pm 0.001}$
HUCB	$\mathbf{0.681}_{\pm 0.004}$	$\mathbf{0.720}_{\pm 0.003}$	$\mathbf{0.660}_{\pm 0.001}$	$\mathbf{0.649}_{\pm 0.002}$
pHUCB	$\mathbf{0.680}_{\pm 0.005}$	$\mathbf{0.723}_{\pm 0.002}$	$\mathbf{0.661}_{\pm 0.002}$	$\mathbf{0.647}_{\pm 0.003}$

We report the test log loss (LogLoss) and area under curve (AUC) score in Table 4. We can observe that both our methods, HUCB and pHUCB, achieve much higher performance than exploitation models including the Linear model, GRU, and Transformers, which demonstrates that our proposed models can effectively help to alleviate the closed-loop effects in recommender systems.

6 CONCLUSION

In this paper, we propose a general hierarchical bandit framework for entire space user interest exploration. Specifically, we design two algorithms, i.e., HCB and pHCB. The HCB algorithm makes a sequence of decision-making tasks to find a path from the root to a leaf node, while pHCB progressively expands the receptive field in a top-down manner to explore the user interests, which is more flexible and also achieves more satisfactory results. Extensive experiments are conducted to demonstrate the effectiveness of the proposed framework on two real-world datasets with three different base bandit algorithms. In the future, we plan to combine our methods with the start-of-the-art deep learning methods to estimate reward for making more reasonable decisions. Moreover, we assume the items are static in this paper by fixing the tree structure unchanged. It would be interesting to extend the proposed frameworks to the non-static setting, which has not been well studied yet.

REFERENCES

[1] Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. 2011. Improved algorithms for linear stochastic bandits. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*. 2312–2320.

[2] Shipra Agrawal and Navin Goyal. 2013. Thompson Sampling for Contextual Bandits with Linear Payoffs. In *Proceedinds of International Conference on Machine Learning*. 127–135.

[3] Peter Auer. 2002. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research* 3, Nov (2002), 397–422.

[4] Djallel Bouneffouf and Irina Rish. 2019. A survey on practical applications of multi-armed and contextual bandits. *arXiv preprint arXiv:1904.10040* (2019).

[5] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 208–214.

[6] Edouard Fouché, Junpei Komiyama, and Klemens Böhm. 2019. Scaling multi-armed bandit algorithms. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1449–1459.

[7] Claudio Gentile, Shuai Li, Purushottam Kar, Alexandros Karatzoglou, Giovanni Zappella, and Evans Etrue. 2017. On context-dependent clustering of bandits. In *Proceedinds of International Conference on Machine Learning*. PMLR, 1253–1262.

[8] Claudio Gentile, Shuai Li, and Giovanni Zappella. 2014. Online clustering of bandits. In *Proceedinds of International Conference on Machine Learning*. 757–765.

[9] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: A Factorization-Machine Based Neural Network for CTR Prediction. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, 1725–1731.

[10] Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof. 2020. Keeping Dataset Biases out of the Simulation: A Debiased Simulator for Reinforcement Learning based Recommender Systems. In *Proceedinds of Fourteenth ACM Conference on Recommender Systems*. 190–199.

[11] Amir H. Jadidinejad, Craig Macdonald, and Iadh Ounis. 2020. Using Exploration to Alleviate Closed Loop Effects in Recommender Systems. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2025–2028.

[12] Himanshu Jain, Yashoteja Prabhu, and Manik Varma. 2016. Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 935–944.

[13] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Nathan Korda, Balazs Szorenyi, and Shuai Li. 2016. Distributed clustering of linear bandits in peer to peer networks. In *Proceedinds of International Conference on Machine Learning*. PMLR, 1301–1309.

[15] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009), 30–37.

[16] Lihong Li, Wei Chu, John Langford, and Robert E. Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international Conference on World Wide Web*. 661–670.

[17] Shuai Li, Alexandros Karatzoglou, and Claudio Gentile. 2016. Collaborative filtering bandits. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 539–548.

[18] Aristidis Likas, Nikos Vlassis, and Jakob J. Verbeek. 2003. The global k-means clustering algorithm. *Pattern recognition* 36, 2 (2003), 451–461.

[19] Wei-Yin Loh. 2011. Classification and regression trees. *Wiley interdisciplinary reviews: data mining and knowledge discovery* 1, 1 (2011), 14–23.

[20] Kanak Mahadik, Qingyun Wu, Shuai Li, and Amit Sabne. 2020. Fast distributed bandits for online recommendation systems. In *Proceedings of the 34th ACM International Conference on Supercomputing*. 1–13.

[21] Jérémie Mary, Romaric Gaudel, and Philippe Preux. 2015. Bandits and recommender systems. In *Proceedinds of International Workshop on Machine Learning, Optimization and Big Data*. Springer, 325–336.

[22] Kanishka Misra, Eric M. Schwartz, and Jacob Abernethy. 2019. Dynamic online pricing with incomplete information using multiarmed bandit experiments. *Marketing Science* 38, 2 (2019), 226–252.

[23] Trong T. Nguyen and Hady W. Lauw. 2014. Dynamic clustering of contextual multi-armed bandits. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. 1959–1962.

[24] Shumpei Okura, Yukihiro Tagami, Shingo Ono, and Akira Tajima. 2017. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1933–1942.

[25] Sandeep Pandey, Deepak Agarwal, Deepayan Chakrabarti, and Vanja Josifovski. 2007. Bandits for taxonomies: A model-based approach. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 216–227.

[26] Yashoteja Prabhu and Manik Varma. 2014. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 263–272.

[27] Nils Reimers, Iryna Gurevych, Nils Reimers, Iryna Gurevych, Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[28] Yuta Saito, Shunsuke Aihara, Megumi Matsutani, and Yusuke Narita. 2020. A Large-scale Open Dataset for Bandit Algorithms. In *Proceedinds of ICML 2020 Workshop on Real World Experiment Design and Active Learning*.

[29] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. 2016. Recommendations as treatments: Debiasing learning and evaluation. In *Proceedinds of International Conference on Machine Learning*. PMLR, 1670–1679.

[30] David Sculley. 2010. Web-scale k-means clustering. In *Proceedings of the 19th International Conference on World Wide Web*. 1177–1178.

[31] Weiwei Shen, Jun Wang, Yu-Gang Jiang, and Hongyuan Zha. 2015. Portfolio choices with orthogonal bandit learning. In *Proceedinds of Twenty-fourth International Joint Conference on Artificial Intelligence*.

[32] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1441–1450.

[33] Thomas J. Walsh, István Szita, Carlos Diuk, and Michael L. Littman. 2009. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*. 591–598.

[34] Qing Wang, Tao Li, S.S. Iyengar, Larisa Shwartz, and Genady Ya Grabarnik. 2018. Online IT ticket automation recommendation using hierarchical multi-armed bandit algorithms. In *Proceedings of the 2018 SIAM International Conference on Data Mining*. SIAM, 657–665.

[35] Qing Wang, Chunqiu Zeng, Wubai Zhou, Tao Li, S. Sitharama Iyengar, Larisa Shwartz, and Genady Ya Grabarnik. 2018. Online interactive collaborative filtering using multi-armed bandit with dependent arms. *IEEE Transactions on Knowledge and Data Engineering* 31, 8 (2018), 1569–1580.

[36] Fangzhao Wu, Ying Qiao, Jiun-Hung Chen, Chuhan Wu, Tao Qi, Jianxun Lian, Danyang Liu, Xing Xie, Jianfeng Gao, Winnie Wu, and Ming Zhou. 2020. Mind: A large-scale dataset for news recommendation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 3597–3606.

[37] Xian Wu, Suleyman Cetintas, Deguang Kong, Miao Lu, Jian Yang, and Nitesh Chawla. 2020. Learning from Cross-Modal Behavior Dynamics with Graph-Regularized Neural Contextual Bandit. In *Proceedings of The Web Conference 2020*. 995–1005.

[38] Liu Yang, Bo Liu, Leyu Lin, Feng Xia, Kai Chen, and Qiang Yang. 2020. Exploring Clustering of Bandits for Online Recommendation System. In *Proceedinds of Fourteenth ACM Conference on Recommender Systems*. 120–129.

[39] Mengyue Yang, Qingyang Li, Zhiwei Qin, and Jieping Ye. 2020. Hierarchical Adaptive Contextual Bandits for Resource Constraint based Recommendation. In *Proceedings of The Web Conference 2020*. 292–302.

[40] Xiaoying Zhang, Hong Xie, Hang Li, and John CS Lui. 2020. Conversational contextual bandit: Algorithm and application. In *Proceedings of The Web Conference 2020*. 662–672.

[41] Han Zhu, Xiang Li, Pengye Zhang, Guozheng Li, Jie He, Han Li, and Kun Gai. 2018. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1079–1088.

[42] Jingwei Zhuo, Ziru Xu, Wei Dai, Han Zhu, Han Li, Jian Xu, and Kun Gai. 2020. Learning Optimal Tree Models under Beam Search. In *Proceedinds of International Conference on Machine Learning*. PMLR, 11650–11659.

APPENDIX

1. Regret Analysis

As defined in Eq. (1), the regret is defined as the difference between the expected reward under hindsight knowledge and the actual reward under the algorithm. The regret bound we would like to obtain is established on a premise that the clustering structure is known to the algorithm ahead of time, which is consistent with the scenario of this paper. In this case, each cluster is viewed as an independent arm, according to [1, 3, 5], the regret bound is up to logarithmic terms $\ln(T)$, $\ln(N)$, and $\ln(1/\delta)$. By hiding the logarithmic factors with notation $\widetilde{O}$, the cumulative regret over T rounds is bounded with probability $1-\delta$ as:

$$R(T) = \widetilde{O}\left(\sum_{k=1}^{K}(\sigma d + ||X_k||\sqrt{d})\sqrt{T}\right) \tag{9}$$

In Eq. (9), we shall assume that $||X_k|| = 1$ for all clusters. Without loss of generality, we can conduct l_2 normalization over the contextual vector associated with each arm. Then as proven by [8], one can replace $\sqrt{T}$ of each arm by a term formulated as $\sqrt{T}(\frac{1}{K}+\sqrt{\frac{|V_k|}{N}})$, where $|V_k|$ is the size of k-th cluster and N is the total number of items. As a result, the regret bound becomes:

$$R(T) = \widetilde{O}\left((\sigma d + \sqrt{d})\sqrt{T}\left(1 + \sum_{k=1}^{K}\sqrt{\frac{|V_k|}{N}}\right)\right) \tag{10}$$

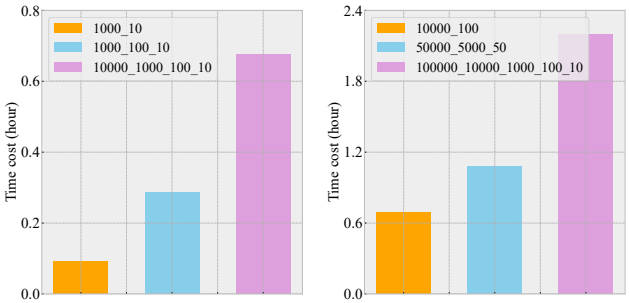
In Eq. (10), according to [8], the worst case occurs when each cluster has the same size $\frac{N}{K}$, leading to the regret bound:

$$R(T) = \widetilde{O}\left((\sigma d + \sqrt{d})\sqrt{KT}\right) \tag{11}$$

Here we first discuss the regret bound for HCB. For simplicity, we assume that the number of clusters are reduced m times that of the previous level. At the beginning, each item is treated as a cluster, i.e. the number of clusters should be N. In this case, the regret bound holds $\widetilde{O}\left((\sigma d + \sqrt{d})\log_m(N)\sqrt{mT}\right)$. For pHCB, the receptive field expands in a progressive manner, assuming the final size of receptive field is r, the regret bound is at most $\widetilde{O}\left((\sigma d + \sqrt{d})\sqrt{rT}\right)$. As proven in [5], if the arm set is fixed over time and contains N arms, the regret bound of the contextual bandits with linear payoff functions is up to $O(\sqrt{Td}\ln^{3/2}(NT\ln(T)/\delta))$. It is significantly larger than the regret bound of HCB and pHCB due to the higher order of logarithmic terms and $d \ll N$, $m \ll N$, and $r \ll N$ in most instances. Therefore, our proposed algorithms can improve the exploration efficiency substantially by reducing the cumulative regret. Moreover, the exploration time complexity is reduced from $O(N)$ to $O(\log(N))$ with the help of hierarchy.

2. Tree Construction Time Cost

Next, we study the time cost spent on constructing different tree structures. K-Means clustering is utilized to assign items into multiple subsets, which has also proven to be one of the effective clustering methods. However, when dealing with millions of items, the time cost is still unacceptable. Therefore, we adopt the MiniBatchK-Means [30] clustering to speed up K-Means when the number of items exceeds half a million. We build different tree structures on a Linux server with an Intel Xeon CPU E5-2680, 250GB memory. As shown in Figure 5, the construction of tree structures is very fast on MIND dataset, with less than one hour's cost even for the deepest structure. On Taobao dataset, although it contains millions of items, the construction process only spends around two hours for the deepest structure. This demonstrates that the bottom-up clustering method for tree structure construction is quite time-efficient.



(a) MIND (b) Taobao

Figure 5: Time cost of tree construction on two datasets. *a_b_c* means there are a, b, and c nodes from bottom to up levels of the tree structure.