Feedback Adaptive Learning for Medical and Educational Application Recommendation

Cem Tekin, *Senior Member, IEEE*, Sepehr Elahi, *Student Member, IEEE*, Mihaela van der Schaar, *Fellow, IEEE*

Abstract—Recommending applications (apps) to improve health or educational outcomes requires long-term planning and adaptation based on the user feedback, as it is imperative to recommend the right app at the right time to improve engagement and benefit. We model the challenging task of app recommendation for these specific categories of apps—or alike—using a new reinforcement learning method referred to as episodic multi-armed bandit (eMAB). In eMAB, the learner recommends apps to individual users and observes their interactions with the recommendations on a weekly basis. It then uses this data to maximize the total payoff of all users by learning to recommend specific apps. Since computing the optimal recommendation sequence is intractable, as a benchmark, we define an oracle that sequentially recommends apps to maximize the expected immediate gain. Then, we propose our online learning algorithm, named FeedBack Adaptive Learning (FeedBAL), and prove that its regret with respect to the benchmark increases logarithmically in expectation. We demonstrate the effectiveness of FeedBAL on recommending mental health apps based on data from an app suite and show that it results in a substantial increase in the number of app sessions compared with episodic versions of ϵ_n-greedy, Thompson sampling, and collaborative filtering methods.

Index Terms—Recommender systems, application recommendation, online learning, multi-armed bandit.

1 Introduction

As the use of mobile devices in everyday life keeps growing at an unprecedented rate, there has been a surge of mobile applications that affects various aspects of modern life such as personalized healthcare, education, and entertainment services. Choosing the applications that match with the needs of the users from a large and diverse set of alternatives requires development of sophisticated recommendation methods [1], [2], [3], [4]. In order to achieve a high level of personalization in this diverse environment, application stores have adopted taxonomy based categorization of the apps [5]. Therefore, to achieve optimal personalization, category specific challenges of user-app interaction need to be taken into account when designing recommendation algorithms.

Recently, there has been a growing interest in two specific app categories, namely medical [6] and educational [7] apps. App recommendations for both of these categories are goal driven, i.e., recommendations are made with the goal of improving health or educational outcomes. Therefore, what to recommend and when to recommend should be carefully chosen based on the past feedback from the users in order to align the sequence of recommendations with the ultimate goal. In this paper, we address this task by proposing a new learning framework for feedback adaptive app recommendation.

We start by explaining why medical and educational app recommendation require feedback adaptive learning by providing a set of real-world examples. For instance, a suite of apps with educational and interactional style to support users to acquire a set of skills of managing depression or anxiety is considered in IntelliCare [8]. The goal of this study was to observe user interactions with apps and improve users' conditions by randomly recommending apps so that by analyzing the user data, a recommendation engine could be developed. Improving a user's condition is challenging because it is difficult to asses the improvement in the patient's condition due to app recommendations. However, the patient's interactions with the apps, including time spent in an app, number of times an app is opened etc. (collectively referred to as user engagement), provide auxiliary information about patient outcomes, and thus can be used as a proxy to the benefit that the patient receives from the apps. Therefore, the efficiency of app recommendations can be measured by user engagement.

In this setting, the order of app recommendations play a central role in improving user engagement [9]. Moreover, further app recommendations are driven by current user engagement. Back to our example of medical app recommendations: when users participating in the IntelliCare experiment were recommended the same mental-illness-aiding apps each week but in different orders, their engagement vastly varied. Fig. 1 shows the cumulative number of app sessions that an average user had when recommended three different sequences of apps from a set of five different apps, for ten weeks. Notice that the "right" sequence results in more than four times the number of total app sessions than the next best sequence by the end of the tenth week. This example demonstrates the importance of recommending the right app at the right time.

Besides medical apps, educational apps for learning languages have also shown to be very effective and popular [10]. Notice that it makes sense to recommend different educational apps at different time steps (weeks), because

C. Tekin and S. Elahi are with the Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey. Email: cemtekin@ee.bilkent.edu.tr, sepehr.elahi@ug.bilkent.edu.tr. M. van der Schaar is with the Department of Applied Mathematics & Theoretical Physics, University of Cambridge, Cambridge, UK. Email: mihaelaucla@gmail.com.

some apps are geared to beginners whereas others are made for intermediate or advanced learners. For instance, it would be better to recommend a simple app that teaches basic English words, like Duolingo[1], to an English learner who is just starting out. As this learner practices and improves his English skills (i.e., time passes), he can be recommended a more advanced app like Grammarly[2] that checks the tone of an essay.

Both of the examples discussed above motivate us to model app recommendations as a structured reinforcement learning problem called *episodic multi-armed bandit* (eMAB). In eMAB, the learner proceeds in episodes $\rho = 1, 2, \ldots$ (users arriving sequentially over time)[3] composed of multiple steps (weeks), in which the learner selects actions (app recommendations) sequentially in steps, one after another, with each action belonging to the action set $\mathcal{A}$. After each taken action $a \in \mathcal{A}$, a feedback $f \in \mathcal{F}$ (number of app sessions in the week after the app recommendation) is observed. Based on all its previous observations in that episode, the learner either decides to continue to the next step by selecting another action or selecting a *stop* action (stop app recommendations) which ends the current episode, yields a terminal reward (total number of app sessions), and starts the next episode. Hence, the number of steps in each episode is a decision variable, and the terminal rewards and costs of the steps in an episode are observed only after the *stop* action is taken. The goal of the learner is to maximize its total expected gain (i.e., the terminal reward minus costs) over all episodes by learning to choose the best action sequence given the feedback. An illustration that shows the order of steps, costs, terminal rewards and episodes for app recommendation is given in Fig. 2.

In short, we depart from the prior work in recommender systems [11], [12], [13] by modeling the long-term interaction between the users and the system as a MAB. In particular, in medical application recommendation: (i) Users start using an application suite usually after a diagnosis; (ii) Continuously recommending applications to a user may have negative impact and result in user disengagement, thus one should stop recommendations after some time when the cost exceeds the future benefit; (iii) Users stop using the application suite when the treatment ends.[4] The learner does not know when a user should stop at the beginning, since when to stop should be adjusted based on the improvement in user's medical condition and user engagement, which can only be inferred from user feedback.

1. www.duolingo.com
2. www.grammarly.com
3. Our model can easily be adapted to handle batch user arrivals or new users arriving before the current user completes its episode. However, for the clarity of presentation and technical analysis, and to remove the potential bias that the user arrival process might have on the performance, we assume throughout the paper that users arrive sequentially over time. Batch arrivals are considered in the experimental results.
4. Educational application recommendations follow a similar trend: (i) Users start using an application suite/bundle to learn a subject, e.g., a language, up to a certain level; (ii) Level of recommended applications should match level of the user; otherwise the user may lose interest and drop out; (iii) Users stop using the application suite when they gain the expected level of expertise on the subject. Note that in cases where apps are to be recommended indefinitely, the stop action can be removed from the action set without any detriment to recommendation performance.

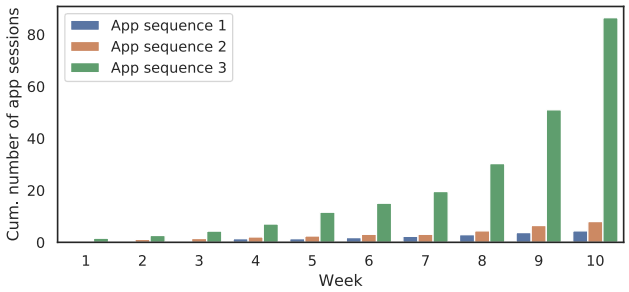


Fig. 1: The cumulative number of app sessions that a user in the IntelliCare experiment had when recommended three different sequences containing the same five apps.

In general, users can drop out of the system if they are unsatisfied with the recommendations. Thus, it might be beneficial to stop when we are confident that the remaining app recommendations will not provide future benefit. In addition, recommendations may have a monetary cost in many settings. Since the number of apps a user can simultaneously use is limited, recommendations can have a diminishing return over time. For instance, a user who is already satisfied with the apps that he/she is currently using, will have little incentive to download and use a new app.

The contributions are summarized as follows:

- We model app recommendations as a structured reinforcement learning problem called eMAB.
- We propose the *FeedBack Adaptive Learning* (FeedBAL) and compare FeedBAL with a benchmark that always chooses the myopic best action given the current feedback, and prove that it achieves $O(\log n)$ regret, where n denotes the number of episodes. Moreover, the regret has polynomial dependence on the number of steps, actions and states. This result also indicates that the difference between the average performance of FeedBAL and that of the benchmark diminishes at a rate $O(\log n/n)$.
- We use FeedBAL for recommending mental health apps based on real data from an app suite, and prove that it significantly improves user engagement.

The rest of the paper is organized as follows: Related works are given in Section 2, mathematical descriptions of eMAB, benchmark and the regret are given in Section 3, description of FeedBAL and its regret are given in Section 4, regret bounds of FeedBAL are given in Section 5, FeedBAL's effect on user engagement in medical app recommendation is given in Section 6, and finally concluding remarks are given in Section 7. Additionally, proofs are given in the supplemental document.

2 Related Works

Matrix Factorization: Matrix factorization methods have been extensively used for making personalized app recommendations. For instance, a method that mines context-aware user preferences from context logs is proposed in [11]. In order to deal with sparse and binary user-app interaction feedback, a method that performs kernel-incorporated matrix factorization is developed in [13]. In addition to these,

[12] addresses the problem of predicting users' preferences for apps using factorization machines. All of the works mentioned above require a priori data to train their models. In contrast, our model is completely online. It uses data gathered from the previous users to inform app recommendations for the current user. As the data gathering process is tied with the recommendation process, it is necessary to balance exploration and exploitation. Indeed, we are able to quantize how the accuracy of our recommendations improve as more data is gathered (Theorem 1, 2 and Corollary 1, 2) by judiciously balancing exploration and exploitation.

Adaptive Treatment Strategies: Mobile apps have also been used for treating patients. There has been a surge of research on developing machine learning methods of estimating optimal treatment regimes to assign interventions or recommendations to patients. A dynamic programming based method using backward induction [14] and a structural nested mean method with G estimation [15] have laid a foundation for estimating a sequence of decision rules to optimize a long term health outcome. Based on the backward induction framework, the most commonly used methods to estimate optimal treatment regimes are Q(uality) learning [9], [16] and A(dvantage) learning [17], [18]. Q learning is a basic reinforcement learning method to model the cumulative reward conditioning on the state and action, while A learning models the contrast of cumulative reward under different actions.

MABs: eMAB is related to various existing classes of MAB with large action sets. These include combinatorial bandits [19], [20], [21] and matroid bandits [22]. In these works, at each time, the learner (simultaneously) chooses an action tuple and obtains a reward that is a function of the chosen action tuple. Unlike these works, in eMAB actions in an episode are chosen sequentially, and the previously chosen actions in an episode guide the action selection process within that episode. The differences between eMAB and these various classes of MAB are given in Table 1a.

One of the most closely related prior works is the work on adaptive submodularity [23] where it is shown that for adaptive submodular reward functions, a simple adaptive greedy policy (which resembles our benchmark) is $1-1/e$ approximately optimal. Hence, any learning algorithm that has sublinear regret with respect to the greedy policy is guaranteed to be approximately optimal. This work is extended to an online setting in [24], where prior distribution over the state is unknown and only the reward of the chosen sequence of actions is observed. However, an independence assumption is imposed over action states to estimate the prior in a fast manner.

Markov Decision Processes (MDPs): Our problem is also related to reinforcement learning in MDPs. For instance, in [25] and [26] algorithms with logarithmic regret with respect to the optimal policy are derived for finite, recurrent MDPs. However, the proposed algorithms rely on variants of value iteration or linear programming, and hence, have higher computational complexity than our proposed method. Episodic MDPs are studied in [27], and sublinear regret bounds are derived assuming that the loss sequence is generated by an adversary. eMAB differs from these works as follows: (i) The number of visited steps (states) in each episode is not fixed; (ii) During an episode, only feedbacks are observed and no reward observations are available for the intermediate states; (iii) Rewards of the intermediate states are only revealed at the end of the episode. Recently, improved gap-independent regret bounds are derived for reinforcement learning in MDPs by using an optimistic version of value iteration [28] for episodic MDPs and posterior sampling for non-episodic MDPs [29]. While it is possible to translate eMAB into an MDP, finding the optimal policy in the MDP is more challenging than competing with our benchmark, both in terms of the speed of learning and cost of computation. Thus, eMAB can be seen as a bridge between standard MAB and reinforcement learning in MDPs, where the order of actions taken in each episode matters and the learner aims to perform as good as a *moderate* benchmark which may not always be optimal, but outperforms the best fixed action and works well in a wide range of settings.

In [30], PAC bounds are derived for continuous state MDPs with unknown but deterministic state transitions and geometrically discounted rewards, using a metric called *policy-mistake count*. In contrast, we develop regret bounds for eMAB which hold uniformly over time for unknown, random state transitions, and undiscounted rewards. We would also like to note that, model-free methods like Q-learning [31] and TD(λ) [32] will be highly inefficient due to the size of the sequence of actions that can be taken, and the sequence of feedbacks that can be observed in each episode. In contrast, regret of eMAB depends only polynomially on the episode length and logarithmically on the number of episodes.

The differences between eMAB, and optimization and reinforcement learning algorithms for MDPs are given in Table 1b.

3 Problem Formulation

Notation: Sets are denoted by calligraphic letters, vectors are denoted by boldface letters and random variables are denoted by capital letters. For a set $\mathcal{E}$, $S_{\mathcal{E}} := |\mathcal{E}|$, where $|\cdot|$ denotes the cardinality. The set of positive integers up to integer t is denoted by $[t]$. $\mathrm{E}_p[\cdot]$ denotes the expectation with respect to probability distribution p. $\mathrm{I}(\mathcal{E})$ denotes the indicator function of event $\mathcal{E}$ which is one if $\mathcal{E}$ is true and 0 otherwise. For a set $\mathcal{E}$, $\Delta(\mathcal{E})$ denotes the set of probability distributions over $\mathcal{E}$. All inequalities that involve random variables hold with probability one.

Background on Reinforcement Learning and MAB: Reinforcement learning can be used to model the long-term interaction between the user and the system [32] in application recommendation. Within this context, user has a state x_t which is affected by the recommendation actions a_t chosen by the system. The user responds to the chosen action by generating a stochastic feedback f_t and by transitioning into a new state x_{t+1}. The goal is to maximize the expected cumulative feedback $\mathrm{E}[\sum_t f_t]$ (or expected discounted cumulative feedback), without knowing a priori stochastic dynamics of the system. Popular reinforcement learning methods include Q-learning and SARSA [32].

Standard MAB [33] can be viewed as a reinforcement learning problem where the state x_t is fixed and does not evolve based on the chosen actions. Compared to general reinforcement learning, this simplification allows derivation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing

TABLE 1: Comparison of eMAB with

(a) combinatorial and matroid bandits.

	[19], [20], [21], [22]	eMAB
Arm selection in each episode	multiple - simultaneous	multiple - sequential
Reward in each episode	sum of rewards of selected arms	general function of selected arms and observed feedbacks
Action sequence length	fixed and limited by action set size	variable, not limited by action set size

(b) optimization and reinforcement learning algorithms.

	PI, VI	Q-learning, TD(λ)	eMAB
Transition probabilities	known	unknown	unknown
Convergence to optimal	always optimal	may converge asymptotically	converges asymptotically
Regret in time	zero	may be sublinear	logarithmic
Efficient for	small action sequences	small action sequences	large action sequences

of much sharper performance guarantees for the standard MAB, namely regret bounds. As the feedback depends on the chosen action, the goal is to maximize the expected cumulative feedback $\mathrm{E}[\sum_t f_t]$ without knowing arm feedback distributions beforehand. The optimal policy in the standard MAB is the one that always picks the action with the highest expected feedback. The regret is the difference between the expected cumulative feedback of the optimal policy and that of the learning policy. Since the optimal policy is unknown, one needs to tradeoff exploration and exploitation in order to maximize $\mathrm{E}[\sum_t f_t]$, which is equivalent to minimizing the regret. Popular algorithms for MAB include upper confidence bound based optimistic exploration algorithms [34] and Thompson sampling [35]. It is known that the regret of these algorithms scale as $O(\log t)$ in time. This means that the gap between their average feedback and that of the optimal policy diminishes at rate $O(\log t/t)$.

As we will describe in the following subsection, our model is much more intricate than a standard MAB. Unlike a standard MAB, we allow the user's state to evolve over time. Moreover, our model allows maximization of a more general set of performance indicators. At the same time, we are able to obtain sharp performance guarantees for the algorithm that we propose.

Problem Description: The system proceeds in episodes indexed by ρ. Each episode represents interaction of the system with a particular user, and is composed of multiple steps indexed by t. In each step the system takes an action from a finite set of actions denoted by $\bar{\mathcal{A}}$. There are two types of actions in $\bar{\mathcal{A}}$: (i) recommendation actions which move the system to the next step and allow it to acquire more information (feedback), (ii) termination action (also named as the *stop* action) which ends the current episode and yields a terminal reward.

Our mathematical model assumes that users arrive one after another, thus, the episode of a new user begins only after the episode of the current user ends. This assumption allows us to clearly quantize the amount of information learned from previous episodes, thereby leading to sharp performance bounds characterized in terms of the regret of learning. In reality, users presence might overlap in time. This is exactly what we consider in simulations (Section 6).

The set of recommendation actions is denoted by $\mathcal{A}$. The maximum number of steps in an episode is $l_{\max} < \infty$, which implies that the *stop* action must be selected in at most $l_{\max}$ steps. After an action $a \in \mathcal{A}$ is selected in a step t, the learner observes a feedback $f \in \mathcal{F}$ before moving to the next step, where $\mathcal{F}$ denotes the set of all feedbacks. In the context of application recommendation, feedback can be the indicator of downloading the application, rating, or usage statistics such as number of app sessions.

Let $\boldsymbol{a}_{[t]} := (a_1, a_2, \ldots, a_t)$ denote a length t sequence of recommendation actions and $\boldsymbol{f}_{[t]} := (f_1, f_2, \ldots, f_t)$ denote a length t sequence of feedbacks. Let $\mathcal{A}^t := \prod_{i=1}^t \mathcal{A}$ denote the set of length t sequences of recommendation actions and $\mathcal{F}^t := \prod_{i=1}^t \mathcal{F}$ denote the set of length t sequences of feedbacks. Set of all recommendation action sequences is denoted by $\mathcal{A}^{\text{all}} := \bigcup_{t=1}^{l_{\max}-1} \mathcal{A}^t$ and the set of all feedback sequences is denoted by $\mathcal{F}^{\text{all}} := \bigcup_{t=1}^{l_{\max}-1} \mathcal{F}^t$. At each step, the system is in one of the finitely many states, where the set of states is denoted by $\mathcal{X}$. State of the system can be interpreted as a summary of the past interaction of the user with the system. It can be viewed as endogenous contextual information of the user. It is composed of components that are extracted from application usage data. For instance, it can include cumulative number of app sessions, application specific regularity in usage [36] and average rating given to all apps in the system by the user so far.

When action a is chosen in step t, the feedback it generates depends on the state of the system in that step. Specifically, we assume that $f_t \sim p_{t,x,a} \in \Delta(\mathcal{F})$, where $p_{t,x,a}$ denotes the probability distribution of the feedback given the step-state-action triplet (t, x, a). Randomness of the feedback captures the uncertainty due to unobserved variables that govern the interaction between the users and the system. Importantly, we consider the challenging case where $p_{t,x,a}$ is unknown. Let $\phi_t : \mathcal{X} \times \mathcal{A} \times \mathcal{F} \to \mathcal{X}$ be the *state transition function* which encodes every state-action-feedback triplet to one of the states in $\mathcal{X}$. Since the feedback is random, the next state is not a deterministic function of the previous state. Moreover, the state transition probabilities are step dependent.

The expected cost of recommendation a in step t when the state is x is given by $c_{t,x,a} \in [0, c_{\max}]$. Cost can be the difficulty level of the app (some apps can be more challenging than the others so that the user needs to gain experience from the other apps before he/she can benefit from this particular app) or the cost of recommendation due to ad placement (text recommendation, video recommendation, etc.) The expected terminal reward in step t when the state is x is given by $r_{t,x} \in [0, r_{\max}]$. This could represent the average rating, the click probability, or the total number of app sessions. The *ex-ante* terminal reward of the triplet

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing

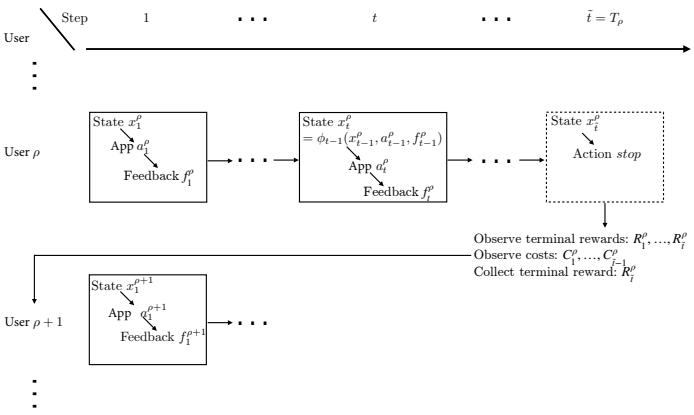


Fig. 2: x_t^ρ is the state observed, a_t^ρ is the action selected and f_t^ρ is the feedback observed in the step t of episode ρ. C_t^ρ is the cost of selecting action a_t^ρ and R_t^ρ is the terminal reward in step t of episode ρ. T_ρ is the step in which the learner (system) selects the *stop* action after which the costs and terminal rewards are revealed.

(t, x, a) is defined as $y_{t,x,a} := \mathrm{E}[r_{t+1,\phi_t(x,a,f_t)}]$ which gives the expected terminal reward of stopping at step $t+1$ after choosing action a in step t and before observing the feedback f_t. For the *stop* action the cost is always zero and $y_{t,x,stop} = r_{t,x}$, $\forall t \in [l_{\max}]$, $\forall x \in \mathcal{X}$. The *gain* of an action $a \in \bar{\mathcal{A}}$ in step t when the state is x is defined as $g_{t,x,a} := y_{t,x,a} - c_{t,x,a}$.

At each episode ρ, the system chooses a sequence of actions $\boldsymbol{a}^\rho := (a_1^\rho, \ldots, a_{T_\rho}^\rho)$, observes a sequence of feedbacks $\boldsymbol{f}^\rho := (f_1^\rho, \ldots, f_{T_\rho-1}^\rho)$ and encounters a sequence of states $\boldsymbol{x}^\rho := (x_1^\rho, \ldots, x_{T_\rho}^\rho)$, where T_ρ denotes the step in which the *stop* action is taken. Since no feedback is present in the first step, for simplicity we set $x_1^\rho = 0$. In principle, the system can work with different initial states $x_1^\rho \in \mathcal{X}$. We note that our model allows us to capture exogenous contextual information about the user through the initial state x_1^ρ. After the *stop* action is taken, the system observes costs of the selected actions $C_t^\rho = c_{t,x_t^\rho,a_t^\rho} + \eta_t^\rho$ for $t \in [T_\rho - 1]$ and the terminal rewards $R_t^\rho = r_{t,x_t^\rho} + \kappa_t^\rho$ for $t \in [T_\rho]$, where η_t^ρ and κ_t^ρ are independent σ-sub-Gaussian random variables that are also independent from $x_{1:t}^\rho, a_{1:t}^\rho, f_{1:t}^\rho, \kappa_{1:t-1}^\rho, \eta_{1:t-1}^\rho$, i.e., $\forall \lambda \in \mathbb{R}$ and $\theta_t^\rho \in \{\eta_t^\rho, \kappa_t^\rho\}$, $\mathrm{E}[e^{\lambda\theta_t^\rho}] \leq \exp\left(\frac{\lambda^2\sigma^2}{2}\right)$. When the episode is clear from the context, we will drop the superscripts from the expressions above.

We assume that the state transition function is known and the state of the system can be computed at any step by using the actions taken and feedbacks observed in the previous steps. Feedback, cost and terminal reward distributions are unknown beforehand. The goal is to maximize its cumulative gain over the episodes by repeated interaction with the system (Fig. 2).

The Benchmark: Since the number of possible action and feedback sequences is exponential in $l_{\max}$, it is very inefficient to learn the best action sequence by separately estimating the expected gain of each action sequence $\boldsymbol{a} \in \mathcal{A}^{\text{all}}$. In this section we propose a (greedy) benchmark, given in Algorithm 1, whose action selection strategy can be learned quickly.

The benchmark[5] incrementally selects the next action based on the past sequence of feedbacks and actions. If the *stop* action is not taken up to step t, the benchmark selects its action in step t according to the following rule: Assume that the state in step t is x. If $g_{t,x,stop} \geq g_{t,x,a}$ for all $a \in \mathcal{A}$ (which implies that $r_{t,x} \geq y_{t,x,a} - c_{t,x,a}$ for all $a \in \mathcal{A}$), then the benchmark selects the *stop* action in step t. Otherwise, it decides to continue for one more step by selecting one of the actions $a \in \mathcal{A}$ which maximizes $g_{t,x,a}$.

Let $\boldsymbol{a}^{*\rho} := (a_1^{*\rho}, \ldots, a_{T_\rho^*}^{*\rho})$ be the action sequence selected, $\boldsymbol{x}^{*\rho} := (x_1^{*\rho}, \ldots, x_{T_\rho^*}^{*\rho})$ be the state sequence, $\boldsymbol{C}^{*\rho} := (C_1^{*\rho}, \ldots, C_{T_\rho^*-1}^{*\rho})$ be the cost sequence observed, and $R_{T^*\rho}^{*\rho}$ be the terminal reward collected by the benchmark in episode ρ, where T_ρ^* is the step in which the *stop* action is selected. The cumulative expected *gain*, i.e., the expected terminal reward minus costs, of the benchmark in the first n episodes is equal to $RW_{\mathrm{B}}(n) := \mathrm{E}\left[\sum_{\rho=1}^n \left(R_{T_\rho^*}^{*\rho} - \sum_{t=1}^{T_\rho^*-1} C_t^{*\rho}\right)\right]$.

Our benchmark is a computationally efficient strategy that provides near-optimal performance under a diverse set of settings, especially when recommendations have diminishing returns. Specifically, our benchmark is approximately optimal when the problem exhibits adaptive monotone submodularity. In addition, cumulative expected gain of our benchmark is in general much higher than that of the best fixed sequence of actions. Technical discussion related to these special cases can be found in the supplemental document.

The Regret: The (pseudo) regret of a learning algorithm which selects the action sequence $\boldsymbol{a}^\rho$ and observes the feedback sequence $\boldsymbol{f}^\rho$ in episode ρ with respect to the benchmark in the first n episodes is given by

$$R(n) := \left(\sum_{\rho=1}^n \left(r_{T_\rho^*, x_{T_\rho^*}^{*\rho}} - \sum_{t=1}^{T_\rho^*-1} c_{t,x_t^{*\rho},a_t^{*\rho}}\right)\right) \tag{1}$$

$$- \left(\sum_{\rho=1}^n \left(r_{T_\rho, x_{T_\rho}^\rho} - \sum_{t=1}^{T_\rho-1} c_{t,x_t^\rho,a_t^\rho}\right)\right). \tag{2}$$

When we take expectation of (2) over all sources of randomness, we obtain the expected regret, which is equivalent to

$$\mathrm{E}[R(n)] = RW_B(n) - \mathrm{E}\left[\sum_{\rho=1}^n \left(R_{T_\rho}^\rho - \sum_{t=1}^{T_\rho-1} C_t^\rho\right)\right]. \tag{3}$$

Any algorithm whose expected regret increases at most sublinearly, i.e., $\mathrm{E}[R(n)] = O(n^\gamma)$, $0 < \gamma < 1$, in the number of episodes will converge in terms of the average reward to the average reward of the benchmark as $n \to \infty$. In the next section we propose an algorithm whose expected regret increases only logarithmically in the number of episodes and polynomially in the number of steps.

4 A Learning Algorithm for eMAB

In this section, we propose *Feedback Adaptive Learning* (FeedBAL) (pseudocode given in Algorithm 2), which learns the

5. This benchmark is similar to the best first search algorithms for graphs [37]. Moreover, it is shown that this benchmark is approximately optimal for problems exhibiting adaptive submodularity [23].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing

Algorithm 1 The Benchmark

Require: $\mathcal{A}$, $\mathcal{X}$, $l_{\max}$
Initialize: $\rho = 1$
while $\rho \geq 1$ **do**
  $t = 1, x_1 = 0$
  while $t \in [l_{\max}]$ **do**
    if $(stop \in \arg\max_{a \in \bar{\mathcal{A}}} g_{t,x_t,a})$ || $(t = l_{\max})$ **then**
      $a_t^* = stop, T_\rho^* = t$ //BREAK
    else
      Select a_t^* from $\arg\max_{a \in \mathcal{A}} g_{t,x_t,a}$
    end if
    Observe feedback f_t
    Set $x_{t+1} = \phi_t(x_t, a_t^*, f_t)$
    $t = t + 1$
  end while
  Observe the costs $C_t^*, t \in [T_\rho^* - 1]$
  Collect terminal reward $R_{T_\rho^*}$
  $\rho = \rho + 1$
end while

sequence of actions to select based on the observed feedbacks to the actions taken in previous steps of an episode (as shown in Fig. 2). In order to minimize the regret given in (3), FeedBAL balances exploration and exploitation when selecting the actions.

FeedBAL keeps the sample mean estimates $\hat{g}_{t,x,a}^\rho$ of the gains $g_{t,x,a}^\rho$ of the actions $a \in \bar{\mathcal{A}}$ and the sample mean estimates $\hat{r}_{t,x}^\rho$ of the terminal rewards $r_{t,x}^\rho$ for all step-state pairs (t, x). Using the definition of the gain for the *stop* action it sets $\hat{g}_{t,x,stop}^\rho = \hat{r}_{t,x}^\rho$ for all (t, x). In addition to these, FeedBAL also keeps the following counters: $N_{t,x}^\rho$ which counts the number of times step-state pair (t, x) is observed[6] prior to episode ρ, and $N_{t,x,a}^\rho$ which counts the number of times action $a \in \bar{\mathcal{A}}$ is selected after step-state pair (t, x) is observed prior to episode ρ.

Next, we explain the operation of FeedBAL. Consider step t of episode ρ. If FeedBAL has not selected the *stop* action yet, using its knowledge of the state x_t^ρ, it calculates the following upper confidence bounds (UCBs): $u_{t,x_t^\rho,a}^\rho := \hat{g}_{t,x_t^\rho,a}^\rho + \text{conf}_{t,x_t^\rho,a}^\rho$ for the actions in $\bar{\mathcal{A}}$, where $\text{conf}_{t,x_t^\rho,a}^\rho$ denotes the *confidence number* for the triplet (t, x, a), which is given as

$$\text{conf}_{t,x_t^\rho,a}^\rho = \sqrt{\frac{(1 + N_{t,x_t^\rho,a}^\rho)}{(N_{t,x_t^\rho,a}^\rho)^2} \left(4\sigma^2 \log\left(\frac{K(1 + N_{t,x_t^\rho,a}^\rho)^{1/2}}{\delta}\right)\right)} \tag{4}$$

for $a \in \mathcal{A}$ and

$$\text{conf}_{t,x_t^\rho,stop}^\rho = \sqrt{\frac{(1 + N_{t,x_t^\rho}^\rho)}{(N_{t,x_t^\rho}^\rho)^2} \left(4\sigma^2 \log\left(\frac{K(1 + N_{t,x_t^\rho}^\rho)^{1/2}}{\delta}\right)\right)} \tag{5}$$

where $K = l_{\max} S_{\mathcal{X}} S_{\bar{\mathcal{A}}}$. If $stop \in \arg\max_{a \in \bar{\mathcal{A}}} u_{t,x_t^\rho,a}^\rho$, then FeedBAL selects the *stop* action in step t. Otherwise, FeedBAL selects one of the actions in $\mathcal{A}$ with the maximum UCB, i.e., $a_t \in \arg\max_{a \in \mathcal{A}} u_{t,x_t^\rho,a}^\rho$. After selecting the action in step t, FeedBAL observes the feedback $f_t^\rho \sim p_{t,x_t^\rho,a_t}$, which is then used to calculate the next state as $x_{t+1}^\rho = \phi_t(x_t^\rho, a_t, f_t^\rho)$.

This procedure repeats until FeedBAL takes the *stop* action, which will eventually happen since the number of steps is bounded by $l_{\max}$. This way the length of the sequence of selected actions is adapted based on the sequence of received feedbacks and costs of taking the actions. After episode ρ ends, FeedBAL observes the costs $C_t^\rho, t \in [T_\rho - 1]$ and the terminal rewards $R_t^\rho, t \in [T_\rho]$. Finally, using these values, FeedBAL updates the values of the sample mean gains and the counters before episode $\rho + 1$ starts, and reaches its objective of maximizing the expected cumulative gain by capturing the tradeoff between the rewards and the costs of selecting actions.

In a nutshell, the principle behind FeedBAL can be explained as follows. FeedBAL uses the principle of optimism in the face of uncertainty in order to tradeoff exploration and exploitation. Basically, it keeps optimistic estimates of the gains ($u_{t,x,a}$) that correspond to each step-state-action triplet. These optimistic estimates are formed by adding an exploration bonus ($\text{conf}_{t,x,a}$) to the sample mean estimates of the gains such that they become upper confidence bounds for the unknown gains ($g_{t,x,a}$). FeedBAL tries to mimic the benchmark (Algorithm 1) by selecting actions with the maximum upper confidence bounds instead of unknown gains. As the number of observations of (t, x, a)'s increases, the noise in the sample mean estimates of the gains decreases. Accordingly, the exploration bonus decreases. This allows rarely encountered (t, x, a)'s to be explored and (t, x, a)'s with high and accurate empirical gains to be exploited. By carefully tuning the exploration bonus (as in Equations 4 and 5), FeedBAL is able to achieve $O(\log n)$ regret with respect to the benchmark.

5 Performance Bounds for FeedBAL

We bound the regret of FeedBAL by bounding the number of times that it will take an action that is different from the action selected by the benchmark.

Let $g_{t,x}^* = \max_{a \in \bar{\mathcal{A}}} g_{t,x,a}$ be the gain of the best action and $\Delta_{t,x,a} = g_{t,x}^* - g_{t,x,a}$ be the suboptimality gap of action a for the step-state pair (t, x). The set of optimal actions for step-state pair (t, x) is given by $\mathcal{O}_{t,x} := \{a \in \bar{\mathcal{A}} : \Delta_{t,x,a} = 0\}$. We impose the following assumption in the rest of this section.

Assumption 1. *For any step-state pair (t, x): (i) $stop \in \mathcal{O}_{t,x} \Rightarrow |\mathcal{O}_{t,x}| = 1$, (ii) $|\mathcal{O}_{t,x}| > 1 \Rightarrow \mathcal{O}_{t,x} \subset \mathcal{A}$.*

Assumption 1 implies that $\mathcal{O}_{t,x}$ cannot include both the *stop* action and another action in $\mathcal{A}$. This assumption is required for our regret analysis. If $\mathcal{O}_{t,x}$ includes both the *stop* action and another action in $\mathcal{A}$, then any learning algorithm may incur linear regret. The reason for this is that the benchmark will always choose the *stop* action in this case, whereas the learner may take the other action more than it takes the *stop* action due to the fluctuations of the sample mean gains around their expected values. To circumvent this effect, the learner can add a small positive bias $\epsilon > 0$ to the gain of the *stop* action. If this bias is

6. We say that a step-state pair (t, x) is observed in episode ρ if the state is x at step t of episode ρ.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing

Algorithm 2 FeedBack Adaptive Learning (FeedBAL)

Require: $\mathcal{A}$, $\mathcal{X}$, $l_{\max}$, σ, δ

Initialize counters: $N_{t,x} = 0$, $N_{t,x,a} = 0$, $\forall t \in [l_{\max}]$, $\forall x \in \mathcal{X}$, $\forall a \in \mathcal{A}$, and $\rho = 1$.

Initialize estimates: $\hat{r}_{t,x} = 0$, $\hat{g}_{t,x,a} = 0$, $\forall t \in [l_{\max}]$, $\forall x \in \mathcal{X}$, $\forall a \in \mathcal{A}$.

while $\rho \geq 1$ **do**
  $t = 1$, $x_1 = 0$
  while $t \in [l_{\max}]$ **do**
    Calculate UCBs: $u_{t,x_t,a} = \hat{g}_{t,x_t,a} + \text{conf}_{t,x_t,a}$, $\forall a \in \bar{\mathcal{A}}$, where $\text{conf}_{t,x_t,a}$ is given in (4) and (5)
    if $(stop \in \arg\max_{a \in \bar{\mathcal{A}}} u_{t,x_t,a})$ || $(t = l_{\max})$ **then**
      $a_t = stop$, $T_\rho = t$ // BREAK
    else
      Select a_t from $\arg\max_{a \in \mathcal{A}} u_{t,x_t,a}$
    end if
    Observe feedback f_t
    Set $x_{t+1} = \phi_t(x_t, a_t, f_t)$
    $t = t + 1$
  end while
  Observe the costs C_t^ρ, $t \in [T_\rho - 1]$ and the terminal rewards R_t^ρ, $t \in [T_\rho]$
  Collect terminal reward $R_{T_\rho}^\rho$
  Update:
  (i) $\hat{g}_{t,x,stop} = \hat{r}_{t,x} = \frac{N_{t,x}\hat{r}_{t,x} + R_t^\rho \mathrm{I}(x_t = x)}{N_{t,x} + \mathrm{I}(x_t = x)}$, for $t \in [T_\rho]$ and $x \in \mathcal{X}$
  (ii) $N_{t,x} = N_{t,x} + \mathrm{I}(x_t = x)$ for $t \in [T_\rho]$ and $x \in \mathcal{X}$;
  (iii) $\hat{g}_{t,x,a} = \frac{N_{t,x,a}\hat{g}_{t,x,a} + (R_{t+1}^\rho - C_t^\rho)\mathrm{I}(x_t = x, a_t = a)}{N_{t,x,a} + \mathrm{I}(x_t = x, a_t = a)}$ for $t \in [T_\rho - 1]$, $x \in \mathcal{X}$ and $a \in \mathcal{A}$;
  (iv) $N_{t,x,a} = N_{t,x,a} + \mathrm{I}(x_t = x, a_t = a)$ for $t \in [T_\rho - 1]$, $x \in \mathcal{X}$ and $a \in \mathcal{A}$
  $\rho = \rho + 1$
end while

small enough such that he *stop* action remains suboptimal for any step-state pair (t, x) in which the *stop* action was suboptimal, then our regret analysis can also be applied to the case when Assumption 1 is violated. Let

$$\mathcal{E}_{\text{conf}} := \{|\hat{g}_{t,x,a}^\rho - g_{t,x,a}| \leq c_{t,x,a}^\rho \quad \forall \rho \geq 2,\ \forall t \in [l_{\max}],\ \forall x \in \mathcal{X},\ \forall a \in \bar{\mathcal{A}}\}$$

be the event that the sample mean gains are within $c_{t,x,a}^\rho$ of the expected gains. The following lemma bounds the probability that $\mathcal{E}_{\text{conf}}$ happens.

Lemma 1. $\Pr(\mathcal{E}_{conf}) \geq 1 - \delta$.

The next lemma upper bounds the number of times each action can be selected on event $\mathcal{E}_{\text{conf}}$.

Lemma 2. *On event $\mathcal{E}_{conf}$ we have*

$$N_{t,x,a}^\rho \leq 3 + \frac{16\sigma^2}{\Delta_{t,x,a}^2} \log\left(\frac{16\sigma^2 K}{\Delta_{t,x,a}^2 \delta}\right) \quad \forall \rho \geq 1,\ \forall t \in [l_{\max}],\ \forall x \in \mathcal{X},\ \forall a \in \bar{\mathcal{A}}.$$

As a corollary of Lemma 2 we derive the following bound on the confidence of the actions selected by FeedBAL.

Corollary 1. *With probability at least $1 - \delta$*

$$\forall \rho \geq 2, \forall t \in [l_{\max}]\ \ g_{t,x_t^\rho}^* - g_{t,x_t^\rho,a_t^\rho} \leq 2\mathit{conf}_{t,x_t^\rho,a_t^\rho}^\rho.$$

Corollary 1 bounds the suboptimality of the action selected by FeedBAL in any step of any episode by $2\text{conf}_{t,x_t^\rho,a_t^\rho}^\rho$, which only depends on quantities δ, K, σ^2 and $N_{t,x_t^\rho,a_t^\rho}^\rho$, which are known by the learner at the time a_t^ρ is selected.

Consider any algorithm that deviates from the benchmark for the first time in step-state pair (t, x) by choosing action a that is different from the action that will be chosen by the benchmark at (t, x). Let $\mu_{t,x}^*$ be the maximum expected gain that can be acquired by the benchmark starting from step-state pair (t, x).[7] Let $\underline{\mu}_{t,x,a}$ be the minimum expected gain that can be acquired by any algorithm by choosing the worst-sequence of actions starting from step-state pair (t, x) after chosing action a. We define the *deviation gap* in step-state pair (t, x) as $\Omega_{t,x,a} := \mu_{t,x}^* - \underline{\mu}_{t,x,a}$. The following theorem shows that the regret of FeedBAL is bounded with probability at least $1 - \delta$.

Theorem 1. *With probability at least $1-\delta$, the regret of FeedBAL given in (2) is bounded by*

$$R(n) \leq \sum_{t=1}^{l_{\max}} \sum_{x \in \mathcal{X}} \sum_{a \notin \mathcal{O}_{t,x}} \Omega_{t,x,a} \left(3 + \frac{16\sigma^2}{\Delta_{t,x,a}^2} \log\left(\frac{16\sigma^2 K}{\Delta_{t,x,a}^2 \delta}\right)\right)$$

Proof. The proof directly follows by summing the result of Lemma 2 among all step-state-action triplets (t, x, a). □

The bound given in Theorem 1 does not depend on n. As given in the following theorem, this bound can be easily converted to a bound on the expected regret by setting $\delta = 1/n$.

Theorem 2. *When FeedBAL is run with $\delta = 1/n$, its expected regret given in (3) is bounded by*

$$\mathrm{E}[R(n)] \leq \Omega_{\max} + \sum_{t=1}^{l_{\max}} \sum_{x \in \mathcal{X}} \sum_{a \notin \mathcal{O}_{t,x}} \Omega_{t,x,a} \left(3 + \frac{16\sigma^2}{\Delta_{t,x,a}^2} \log\left(\frac{16\sigma^2 K n}{\Delta_{t,x,a}^2}\right)\right)$$

where $\Omega_{\max} = \max_{t,x,a} \Omega_{t,x,a}$.

Theorem 2 shows that the expected regret of FeedBAL is $O(\log n)$. Although the constant terms given in Theorems 1 and 2 depend on unknown parameters $\Delta_{t,x,a}$ and $\Omega_{t,x,a}$, FeedBAL does not require the knowledge of these parameters to run and to calculate its confidence bounds. From the expressions in Theorems 1 and 2, it is observed that the regret scales linearly with $\Omega_{t,x,a}/\Delta_{t,x,a}^2$, which is a term that indicates the *hardness* of the problem. If the suboptimality gap $\Delta_{t,x,a}^2$ is small, FeedBAL makes more errors by choosing $a \notin \mathcal{O}_{t,x}$ when it tries to follow the benchmark. This results in a loss in the expected gain that is bounded by $\Omega_{t,x,a}$.

Next, we consider problems in which deviations from the benchmark in early steps cost more than deviations from the benchmark at later steps.

7. In calculating $\mu_{t,x}^*$, we assume that in steps in which the benchmark needs to randomize between at least two actions, the action that maximizes the expected reward of the benchmark is selected.

Assumption 2. *$\Omega_{t,x,a} \leq (l_{\max} - t)\Delta_{t,x,a}$ for all $t \in [l_{\max}]$, $x \in \mathcal{X}$, $a \notin \mathcal{O}_{t,x}$.*

Using this assumption, the following result is derived for the expected regret of FeedBAL.

Corollary 2. *When Assumption 2 holds, and FeedBAL is run with $\delta = 1/n$, we have*

$$\mathrm{E}[R(n)] \leq \Omega_{\max} + l_{\max} \sum_{t=1}^{l_{\max}} \sum_{x \in \mathcal{X}} \sum_{a \notin \mathcal{O}_{t,x}} \left(3\Delta_{t,x,a} + \frac{16\sigma^2}{\Delta_{t,x,a}} \log(\frac{16\sigma^2 Kn}{\Delta_{t,x,a}^2}) \right)$$

Remark 1. *FeedBAL adaptively learns the expected gains of action and feedback sequences that correspond to stopping at various steps. Although our model allows at most $l_{\max}$ actions to be taken in each episode, the actual number of actions taken may be much lower than this value depending on the expected costs $c_{t,x,a}$. High costs implies a decrease in the marginal benefit of continuation, which implies that the benchmark may take the stop action earlier than the case when costs are low.*

6 Results on Medical Application Recommendation

We perform two sets of simulations using FeedBAL, two bandit algorithms: modified versions of ϵ_n-greedy and Thompson Sampling (TS), two collaborative filtering based algorithms, as well as a greedy algorithm (GA) on a statistical model generated by analyzing the data from a medical app suite. These simulations showcase our algorithm outperforming the others in different real-life scenarios, such as those where there are user dropouts. The simulations were coded in Python and the code is available at GitHub.[8]

6.1 Description of the Dataset

This app suite, whose data is used in our simulation, consists of one central 'Hub' app and 12 other apps, which are all medical apps intended to help users with their mental health issues [8]. During the experiment where this data was collected, users were free to choose among these apps by themselves or by following the recommendation of the 'Hub' app. The 12 apps, which help users with depression, anxiety, and other psychological disorders, were shown to users in a central hub app. These recommendations were made at random every week for the duration of the experiment, which was 16 weeks. In total, around 8000 users downloaded and used the hub app to download other apps throughout the experiment. As the users used the apps (excluding the hub app), their number of app sessions with each app was recorded, with one app session being defined as five minutes of being inside an app. In our simulations, we use a Poisson distribution, generated using the app session data of all users, to predict the total number of app sessions[9] that a user will have in a given week when recommended a given app.

8. The code is available at github.com/Bilkent-CYBORG/FeedBAL.

9. From now on, total app sessions will always denote the total app sessions the user had with all of the 12 medical apps. To maintain the anonymity of the IntelliCare participants, real-world data is used to extract a distribution for each (t, x, a) triplet. In simulations, we sample from the distributions that correspond to (t, x, a) triplets to obtain the corresponding feedback (i.e., number of app sessions the user had).

We use the weekly total number of app sessions of each user as an indicator of improvement in the user's condition since it can easily be measured on a weekly basis, unlike the outcome of the treatment, and empirical findings suggest a positive correlation between user engagement and patient outcomes. [8], [38].

6.2 Simulation Setup

The following parts of the setup are common across both simulations: In the beginning of each week in the first 16 weeks, a new group of K users arrive, where K is sampled from a Poisson distribution with mean 1250, hence the expected total number of users is 20,000. Then, these new users along with the old users still in the simulation are recommended apps by the learner. Once all active users have been recommended apps and one time step (week) passes, the learner observes the feedback and cost acquired from each recommendation. Note that each user goes through 16 weeks of app recommendations and new users only arrive in the first 16 weeks of the simulation; so, the overall simulation lasts 32 weeks. The state, x_t, is defined as the total number of app sessions before week t and the feedback, given the triplet (t, x, a), is defined as the total number of app sessions acquired in week t. Note that we define the feedback and state as such because we want them to be a proxy for user improvement and using any of the 12 medical apps, rather that just the app recommended, leads to user improvement [38].

Lastly, the expected cost is a function of t, x, and a and can be seen for all twelve apps in Table 2. As the costs should be comparable to the feedback in order to have an effect on the greedy decisions, the costs include a $\frac{x}{2t}$ term because $\frac{x}{t}$ is the average feedback acquired. We carefully picked these costs to increase the variety of the app with the highest feedback mean for each (t, x) pair. In other words, we made the simulations more challenging by ensuring that the same app is not always outperforming the other ones, which would have been the case had we chosen zero cost for all apps.

Below are the details for each simulation setup and how they differ from one another.

Simulation I: In this simulation, the system observes feedback and incurs cost after recommending each app. Fig. 7 illustrates this simulation setting.

Simulation II: This simulation is alike Simulation I, with the only difference being that users can drop out of the simulation if they are unsatisfied with the recommendations. In the case of medical app recommendations, the user may drop out if he feels that the apps he is being recommended are not improving his condition(s). We model this as follows: Starting from week 12, if the total number of app sessions the user had in weeks $t-2, t-1$, and t are not strictly monotonic increasing, then the user has a ten percent chance of dropping out. Formally, we denote the chance of dropping out by $p(t)$ and define it as

$$p(t) := \begin{cases} 0.10, & t \geq 12 \text{ and not } f_t > f_{t-1} > f_{t-2}, \\ 0, & \text{otherwise.} \end{cases}$$

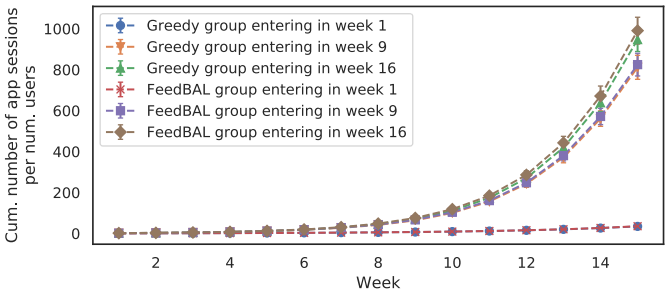
At the end of week t, a Bernoulli random variable with probability $p(t)$ is sampled and if the sample is equal to one, then the user will leave the experiment.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing

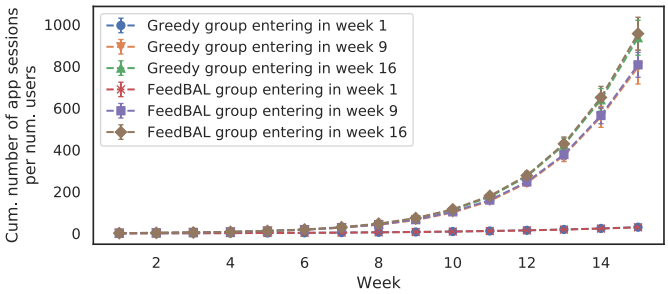
TABLE 2: The expected cost of apps with increasing costs, where $y_{\max} = \frac{x}{2t}$.

App name	iCope	Daily Feats	Day to Day	Purple Chill	Social Force	Aspire
Expected cost	$\frac{y_{\max}}{17}t^2$	$\frac{y_{\max}}{1024}t(t+10)^2$	$\frac{y_{\max}}{15.5}t^2$	$\frac{y_{\max}}{4.5}t\left(1-\exp\left(-\frac{5t}{32}\right)\right)$	$\frac{y_{\max}}{2}t\left(1-\exp\left(-3\frac{t-1}{16}\right)\right)$	$\frac{y_{\max}}{965.12}t(t+2)^2$

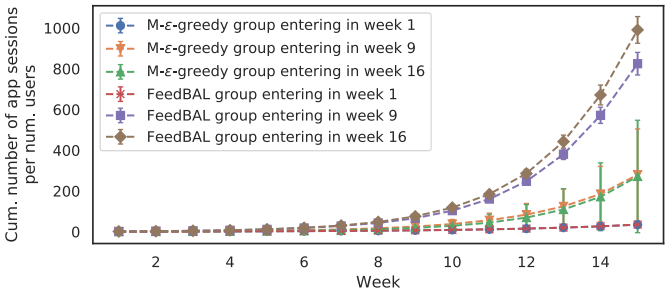
App name	Thought Challenger	Boost Me	My Mantra	MoveMe	Slumber Time	Worry Knot
Expected cost	$1.08y_{\max}t\left(1-\exp\left(-\frac{3t}{16}\right)\right)$	$1.41y_{\max}$	$\frac{y_{\max}}{30}t$	$\frac{y_{\max}}{150}t$	$\frac{y_{\max}}{1.7}t\left(1-\exp\left(-\frac{5t}{32}\right)\right)$	$\frac{y_{\max}}{37.8}t\exp\left(\frac{4-t}{48}\right)$



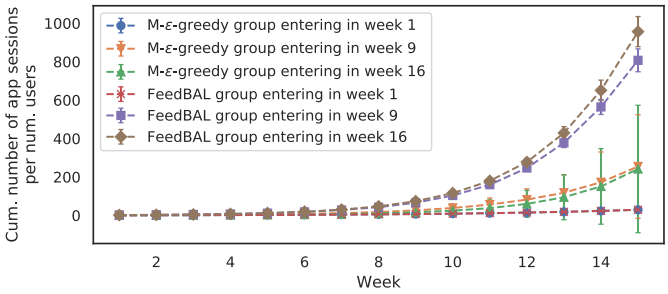
(a) FeedBAL and GA in simulation I



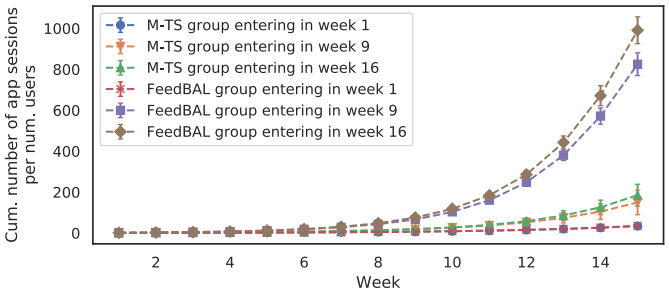
(b) FeedBAL and GA in simulation II



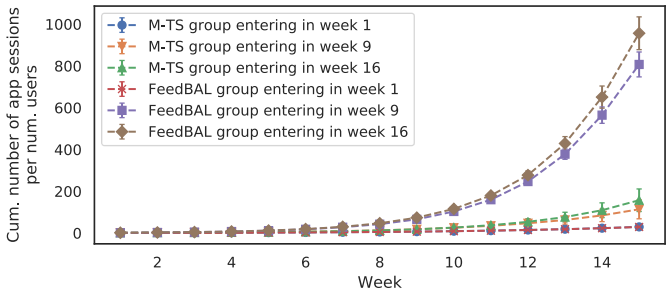
(c) FeedBAL and M-ϵ-greedy in simulation I



(d) FeedBAL and M-ϵ-greedy in simulation II



(e) FeedBAL and M-TS in simulation I



(f) FeedBAL and M-TS in simulation II

Fig. 3: Cumulative number of app sessions per number of users of groups 1, 3, and 9, for bandit-based algorithms in simulations I and II.

TABLE 3: The total number of app sessions of the last user group of the mini-simulation by end the last week, with the given scaling factor.

Scaling factor (α)	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$
Terminal reward of last user group	200	324	632	785	815	751

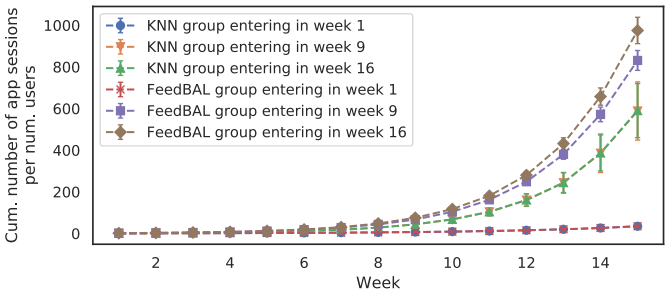
TABLE 4: The dropout rate of each algorithm in simulation II.

Algorithm	Dropout rate (%)	Standard deviation
Benchmark	4.16	0.135
FeedBAL	16.6	0.845
GA	16.6	0.930
M-ϵ_n-greedy	28.7	8.21
M-TS	28.0	4.06
KNN	22.2	2.10
SVD	26.3	3.83

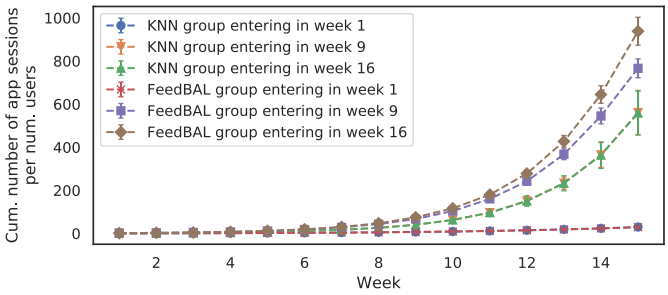
6.3 Algorithms Used

We compare FeedBAL with both bandit-based and collaborative filtering-based algorithms. The bandit algorithms

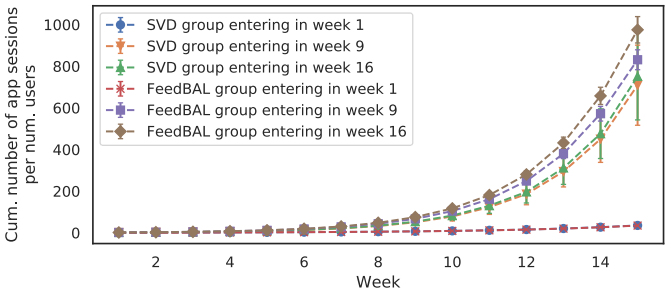
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing



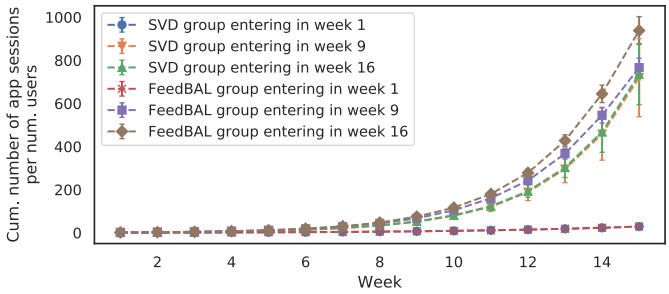
(a) FeedBAL and the KNN in simulation I



(b) FeedBAL and the KNN in simulation II



(c) FeedBAL and SVD in simulation I



(d) FeedBAL and SVD in simulation II

Fig. 4: Cumulative number of app sessions per number of users of groups 1, 3, and 9, for CF-based algorithms in simulations I and II.

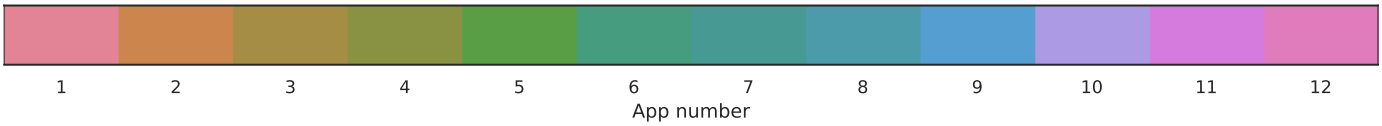


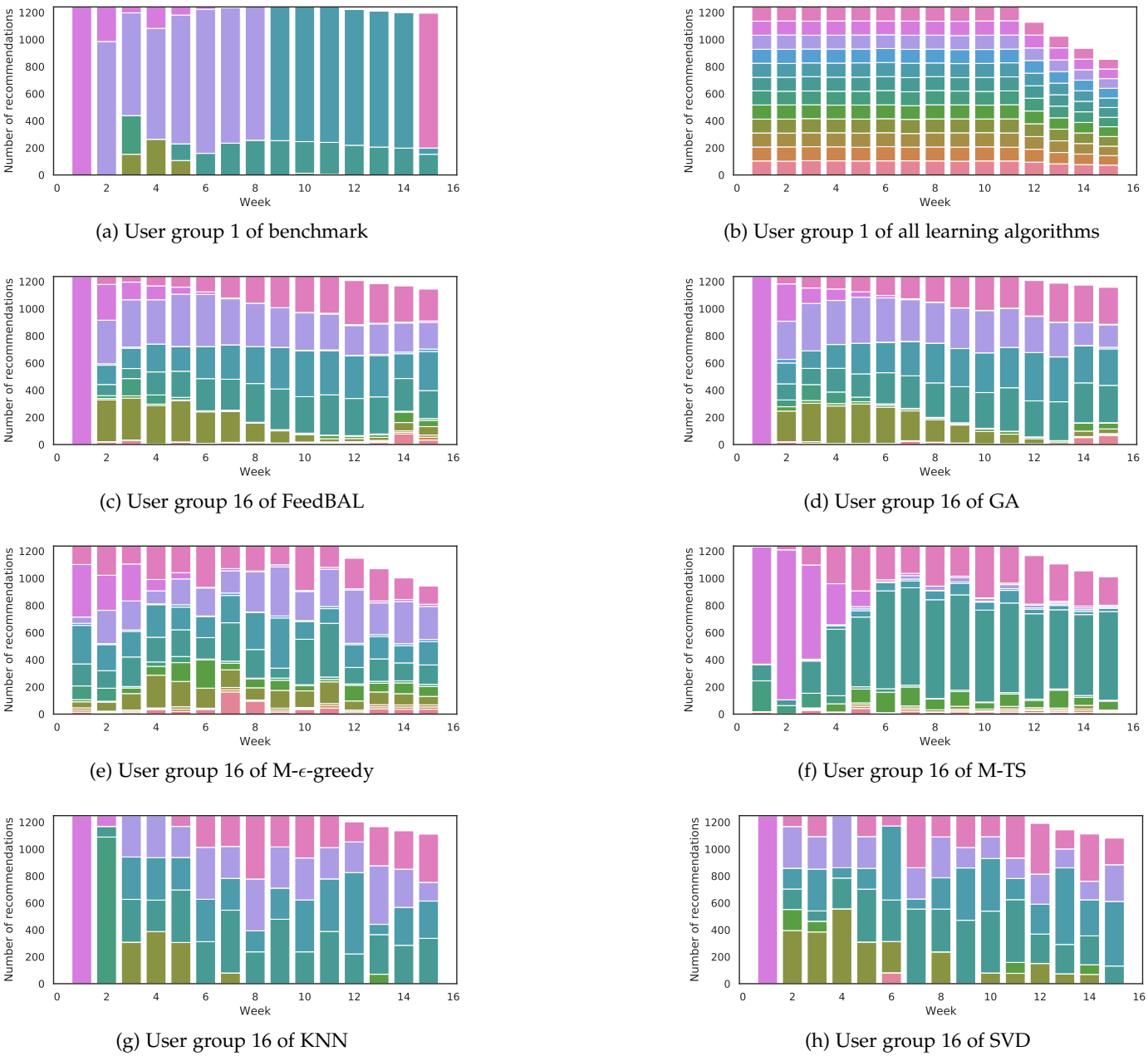
Fig. 5: The colors representing each app. The apps are numbered in the following order: Aspire, Boost Me, Daily Feats, iCope, My Mantra, Day to Day, MoveMe, Purple Chill, Slumber Time, Social Force, Thought Challenger, and Worry Knot.

include variations of Thompson Sampling and ϵ_n-greedy, while the collaborative filtering algorithms are a neighborhood model based on K-nearest neighbors (KNN with baseline), and a factorization model based on singular value decomposition (SVD). Since our data only includes past interactions of users and apps, context-based approaches such as [11], [12], [13] are not suitable for our model. We would also like to note that collaborative filtering has been extended to use more complex techniques such as neural networks [39], however these models would not be suitable for our online setup due to the success of such models stemming from being trained on hundreds of thousands of training data.

FeedBAL: FeedBAL is used with the states, feedbacks, costs, and actions defined in the previous subsection. Note that apps are continuously recommended and the stop action is only taken when it is the 16th week for the user, at which point the episode (user) is ended. Following the end of an episode, the learner observes the total number of app sessions during those 16 weeks as the terminal reward. The learner does not stop until the end because it makes more sense to collect the terminal reward at the end of 16 weeks, when it is maximum. Therefore, app recommendations are made in the first 15 weeks while the stop action is taken in week 16 and terminal reward is collected. Lastly, since the total number of app sessions can reach up to 2000 for most users, FeedBAL needs at least $2000 \times 15 = 30000$ feedback/cost averages to keep track of. This not only drastically increases regret, but it also causes FeedBAL to use a lot of space. To address this issue, we discretize the states as follows: let the discretized version of x be denoted by x_d. Then, $x_d := \begin{cases} x, & x \leq 100, \\ \lfloor \frac{x}{50} \rfloor + 99, & \text{otherwise.} \end{cases}$ Note that this discretization does not change the states of the users, instead it changes the states seen by FeedBAL.

FeedBAL presented in Section 4 assumes that the sub-Gaussian parameter of the reward and cost distribution, σ, is known. However, as this may not be true when performing online recommendations, we make a small modification to FeedBAL so that it learns the σ parameter during the simulation. To do this, we assume that the reward distribution is σ-sub-Gaussian with standard deviation σ, meaning that to estimate σ, FeedBAL needs to keep track of the rewards it received and compute the sample standard deviation. Note that although the rewards in our simulations, which correspond to number of app sessions coming from the IntelliCare suite, follow a Poisson distribution, which is not sub-Gaussian, FeedBAL can still learn and recommend apps successfully because a Poisson distribution with mean λ can be approximated by a Gaussian with mean and variance λ, especially for large λ (see Theorem I of [40]). Lastly, even

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2020.3037224, IEEE Transactions on Services Computing



(a) User group 1 of benchmark

(b) User group 1 of all learning algorithms

(c) User group 16 of FeedBAL

(d) User group 16 of GA

(e) User group 16 of M-ϵ-greedy

(f) User group 16 of M-TS

(g) User group 16 of KNN

(h) User group 16 of SVD

Fig. 6: The number of times each app was recommended in each week of simulation II for the first and last user groups of the benchmark, bandit algorithms and CF-based algorithms.

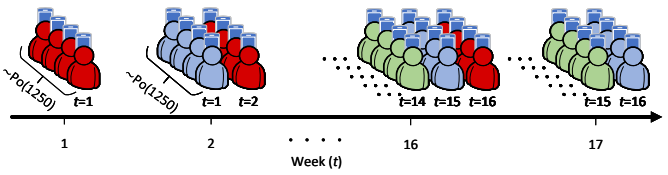


Fig. 7: Batch user arrivals without user dropout. Note that t that is used for each user group is relative to when they entered the experiment. Also, observe that each user group stays for 16 weeks. For instance, the red users, who entered in week 1, leave at the end of week 16.

though the mean (and so standard deviation) of the rewards is a function of (t, x, a), we chose to only keep track of a different σ for each t.

Since the cardinality of $[l_{\max}] \times \mathcal{X} \times \mathcal{A}$ is large, the number of observations of each triplet (t, x, a) is small. In order to avoid having too many explorations, we multiply the confidence term by a scaling factor. In other words, $u^{\rho}_{t,x^{\rho}_t,a} := \hat{g}^{\rho}_{t,x^{\rho}_t,a} + \alpha \cdot \text{conf}^{\rho}_{t,x^{\rho}_t,a}$, where $0 < \alpha \leq 1$ in our simulations.

GA: GA stores the feedback and cost of recommending app a in week t in (discretized) state x, and recommends the app with the highest sample mean feedback minus cost. In other words, GA is just like the FeedBAL algorithm with its confidence term ($\text{conf}^{\rho}_{t,x^{\rho}_t,a}$) set to zero.

M-ϵ_n-greedy and M-TS: Typical MAB algorithms like ϵ_n-greedy [34] or TS [35] cannot directly be used in our setting, as the set of all sequences of app recommendations has

cardinality $12^{15} \approx 1.5 \times 10^{16}$. If each sequence is regarded as an arm, then these algorithms would always be exploring. Therefore, We use modified versions of ϵ_n-greedy and TS, called M-ϵ_n-greedy and M-TS, respectively, that make sense in the context of our episodic app recommendation problem. In these two algorithms, 15 instances[10] of ϵ_n-greedy (with $c = 0.8$ and $d = 0.1$) and 15 instances of TS[11] are used, respectively, with one assigned to each week. For example, to recommend an app in week t using M-TS, the respective M-TS instance for that week will be used and once the feedback and cost arrive, only that TS agent's mean reward will be updated.

KNN with baseline: KNN with baseline, or KNN for short, is a neighborhood model that uses an item-based approach that predicts a user's rating for an app based on other apps that the user has rated. Rating prediction is made using the closest 40 user-item pairs and with a weighted sum with a baseline. We refer readers to Section 2.2 and more specifically Equation 3 of [41] for the details of how the rating is generated. The closest 40 pairs are determined using Pearson's correlation coefficient as the similarity measure. We use the Python library Surprise[12] and more specifically its KNNBaseline class for the implementation of this algorithm.

Matrix factorization with SVD: In matrix factorization-based approaches, the user-app-rating matrix is broken down into a product of smaller matrices, in this case using SVD. The basic idea is that the broken down matrices can provide implicit and hidden information about user-app interactions that are otherwise unused when directly using the user-app-rating matrix. We use the matrix factorization based algorithm of [41], described in Section 3 and Equation 5. We also use the Surprise library for this algorithm's implementation; more specifically the SVD class.

For both CF-based algorithms, we use feedback minus cost, scaled to $[0, 1]$, as the rating. The user-app-rating matrix is filled throughout the experiment and training/data-fitting using the matrix is done at the end of each week. Moreover, when a new user arrives who has no recorded app interactions, he is recommended an app at random.

6.4 Results

For the simulations, we set $l_{\max} = 16$ and $\delta = 0.01$ in FeedBAL. Furthermore, all simulations were ran and averaged on five different user arrival samples, with each simulation being repeated eight times per sample for a total of 40 runs. For FeedBAL, we first ran mini-simulations on 2000 users without dropouts with $\alpha \in \{1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{32}\}$ and recorded the total number of app sessions of the last user group by end the last week (i.e, terminal reward of last user group); this data can be seen in Table 3. We then picked the α corresponding to the highest terminal reward, which was $\frac{1}{16}$, and performed simulations I and II.

10. We use 15 instances because the action taken in week 16 is always the stop action.

11. The update rule is as follows: reward, i.e., feedback minus cost, is observed and scaled to $[0, 1]$. Then, a Bernoulli distribution with probability equal to the scaled reward is sampled. Finally, the a/b parameter is incremented by one if the sample is 1/0. The scaling factor is learned by each TS agent (i.e., when a new reward that is higher than the current scaling factor is observed, the scaling factor is set to that reward).

12. Link: http://surpriselib.com.

Fig. 3a-3f show the cumulative number of app sessions observed by three user groups entering in week 1, 9, and 16, up to and including week t, achieved by the bandit-based algorithms in simulations I and II, and Fig. 4a-4d show the same for CF-based algorithms. The x-axis is the relative week number of the user. For instance, if a user enters the experiment in the 13th week and has y app sessions, then y will be added to app sessions of week 1 as he observed y sessions in his first week of entering the experiment.

When compared with other bandit algorithms, FeedBAL manages to outperform the other algorithms and achieve substantial improvement over M-ϵ_n-greedy and M-TS as the experiment proceeds (e.g. from week 9 to week 16) in both simulations. The GA performs the closest to FeedBAL, having done so as a result of the nature of the data, where the difference between the user app session counts of the top apps in each week is not small. Hence, in other settings where the difference between the reward of the optimal action and the reward of the next optimal action is small, we expect the difference between GA and FeedBAL's performance to grow.

FeedBAL also manages to outperform CF-based algorithms. Notice that both KNN and SVD do not provide any major improvement from week 9 to week 16, which indicates that they have maximized the information coming solely from user-app interactions at the end of week 9. In other words, FeedBAL outperforms them because it takes into account in which week and with how many previous app sessions (state) a user was given an app recommendation when making decisions for future users. On the other hand, both KNN and SVD do not take into account week or state.

When it comes to simulation II, M-ϵ_n-greedy and M-TS also perform poorly as they cannot learn how to recommend the best apps fast enough, which results in a significant drop in the cumulative number of app sessions. This stems from their failure to exploit the state information. KNN and SVD perform slightly better with user dropouts and SVD even gets very close to FeedBAL, as seen in Fig. 4d, but FeedBAL still manages to outperform them all.

To demonstrate that the actions taken by FeedBAL approach those taken by the benchmark that picks apps with the highest expected feedback, we look at the number of times each of the twelve apps were recommended by the benchmark and all four algorithms during simulation II. We expect that FeedBAL will recommend apps randomly (exploration) for the first user group, while it will recommend the apps it has learned to give the highest reward minus cost (exploitation) for the last user group. Our expectations were accurate and the results can be seen in Fig. 6a to 6h. These figures show the number of times each app was recommended in each week using stacked bar charts, where each color represents an app. Fig. 5 shows the app to color mapping. Notice that all learning algorithms recommend all apps the same number of times for the first user group (Fig. 6b), with the reason being that when recommending apps to the users in the first user group, each bandit algorithm is seeing each week for the first time and each CF-based algorithm is seeing the users for the first time and so they recommend apps randomly. Furthermore, the benchmark recommends the same apps for each user group, hence the

illustration of apps recommended for user group 1 (Fig. 6a) is representative of all user groups.

While all bandit algorithms learn something by the time the last user group enters the experiment, only GA and FeedBAL manage to learn the correct apps (i.e., those with the highest feedback minus cost mean, which are picked by the benchmark). Visually, notice that not only did the the number of colors, each of which represents an app, decreased from user group one to user group sixteen of FeedBAL (Fig. 6b to Fig. 6c), but so did the drop in column heights. These two visual changes, which represent the learning of app rewards/costs and decrease in user dropouts, respectively, are hardly present in the figures of M-ϵ_n-greedy and M-TS.

The CF-based algorithms did fare better than the bandit ones (excluding GA and FeedBAL), but their downfalls are not using week/state information and their lack of exploration. Visually, notice that only a few apps are recommended by KNN and SVD (few distinct colors in Fig. 6g and 6h), while FeedBAL recommends nearly every app in most weeks (many distinct colors in Fig. 6c).

We present the dropout rates of each algorithm, defined as the number of users that dropped out divided by total number of users that entered the experiment, in Table 4. Observe that FeedBAL and GA achieve the lowest dropout rate, which is 12.1, 11.4, and 9.7 percent lower than those of M-ϵ_n-greedy, M-TS, and SVD respectively. This result is also visible in Fig. 6a to 6c, where FeedBAL and GA achieve the lowest drop in bar height after week 12 of the last user group. KNN performs the second best after FeedBAL and GA, but it is still significantly behind at 22.2 percent, versus the 16.6 percent dropout rate of FeedBAL and GA.

Lastly, the apps chosen by FeedBAL for the last user group resemble those chosen by the benchmark much more than M-ϵ_n-greedy and M-TS, as evident by the similar colors in Fig. 6c and 6a. KNN and SVD recommend similar apps when compared with FeedBAL, with the main difference being a lack of exploration, which as mentioned previously hinders their performance.

Note that although overall GA performs just under FeedBAL, it does not mean it will always do so in every simulation setting and on every dataset. Rather, it highlights the fact that the user data was skewed and users of IntelliCare mostly used one or two of the twelve apps. Had the data have been better distributed with all apps equally used, then we would expect to see GA get stuck in recommending a suboptimal app.

7 Conclusion

We proposed a new class of online learning methods for application recommendation. Although the number of possible sequences of actions increases exponentially with the length of the episode, we proved that an efficient online learning algorithm that has expected regret that grows polynomially in the number of steps and states, and logarithmically in the number of episodes exists. We also showed that the proposed algorithm outperforms ϵ_n-greedy and TS based learning methods and CF based learning methods.

Acknowledgments

The authors would like to thank Ken Cheung and Xinyu Hu of Columbia University for providing us with IntelliCare data that was used in the simulations. The work of Cem Tekin was supported by BAGEP 2019 Award of the Science Academy.

References

[1] Z. Zheng, H. Ma, M. R. Lyu, and I. King, "QoS-aware web service recommendation by collaborative filtering," *IEEE Trans. Services Comput.*, vol. 4, no. 2, pp. 140–152, 2010.
[2] L. Song, C. Tekin, and M. van der Schaar, "Online learning in large-scale contextual recommender systems," *IEEE Trans. Services Comput.*, vol. 9, no. 3, pp. 433–445, 2014.
[3] R. Liu, J. Cao, K. Zhang, W. Gao, J. Liang, and L. Yang, "When privacy meets usability: unobtrusive privacy permission recommendation system for mobile apps based on crowdsourcing," *IEEE Trans. Services Comput.*, vol. 11, no. 5, pp. 864–878, 2016.
[4] O. Khalid, M. U. S. Khan, S. U. Khan, and A. Y. Zomaya, "OmniSuggest: A ubiquitous cloud-based context-aware recommendation system for mobile social networks," *IEEE Trans. Services Comput.*, vol. 7, no. 3, pp. 401–414, 2013.
[5] B. Liu, Y. Wu, N. Z. Gong, J. Wu, H. Xiong, and M. Ester, "Structural analysis of user choices for mobile app recommendation," *ACM Trans. Knowl. Discovery Data*, vol. 11, no. 2, pp. 1–23, 2016.
[6] M. Terry, "Medical apps for smartphones," *Telemedicine and e-Health*, vol. 16, no. 1, pp. 17–23, 2010.
[7] K. Hirsh-Pasek, J. M. Zosh, R. M. Golinkoff, J. H. Gray, M. B. Robb, and J. Kaufman, "Putting education in "educational" apps: Lessons from the science of learning," *Psychol. Sci. Public Interest*, vol. 16, no. 1, pp. 3–34, 2015.
[8] D. C. Mohr, K. N. Tomasino, E. G. Lattie, H. L. Palac, M. J. Kwasny, K. Weingardt, C. J. Karr, S. M. Kaiser, R. C. Rossom, L. R. Bardsley *et al.*, "Intellicare: an eclectic, skills-based app suite for the treatment of depression and anxiety," *J. Med. Internet Res.*, vol. 19, no. 1, p. e10, 2017.
[9] Y. K. Cheung, B. Chakraborty, and K. W. Davidson, "Sequential multiple assignment randomized trial (SMART) with adaptive randomization for quality improvement in depression treatment program," *Biometrics*, vol. 71, no. 2, pp. 450–459, 2015.
[10] E. M. Luef, B. Ghebru, and L. Ilon, "Language proficiency and smartphone-aided second language learning: A look at english, german, swahili, hausa and zulu." *Electronic Journal of e-Learning*, vol. 17, no. 1, pp. 25–37, 2019.
[11] H. Zhu, E. Chen, H. Xiong, K. Yu, H. Cao, and J. Tian, "Mining mobile user preferences for personalized context-aware recommendation," *ACM Trans. Intell. Syst. Technol.*, vol. 5, no. 4, pp. 1–27, 2014.
[12] T. Liang, L. Zheng, L. Chen, Y. Wan, S. Y. Philip, and J. Wu, "Multi-view factorization machines for mobile app recommendation based on hierarchical attention," *Knowledge-Based Systems*, vol. 187, p. 104821, 2020.
[13] C. Liu, J. Cao, and S. Feng, "Leveraging kernel-incorporated matrix factorization for app recommendation," *ACM Trans. Knowl. Discovery Data*, vol. 13, no. 3, pp. 1–27, 2019.
[14] S. A. Murphy, "Optimal dynamic treatment regimes," *J. R. Stat. Soc.: Series B (Stat. Methodol.)*, vol. 65, no. 2, pp. 331–355, 2003.
[15] J. M. Robins, "Optimal structural nested models for optimal sequential decisions," in *Proc. 2nd Seattle Symposium in Biostatistics*, 2004, pp. 189–326.
[16] S. A. Murphy, "An experimental design for the development of adaptive treatment strategies," *Statistics in Medicine*, vol. 24, no. 10, pp. 1455–1481, 2005.
[17] D. Blatt, S. A. Murphy, and J. Zhu, "A-learning for approximate planning," *Ann Arbor*, vol. 1001, pp. 48 109–2122, 2004.
[18] P. J. Schulte, A. A. Tsiatis, E. B. Laber, and M. Davidian, "Q-and A-learning methods for estimating optimal dynamic treatment regimes," *Stat. Sci.*, vol. 29, no. 4, p. 640, 2014.
[19] Y. Gai, B. Krishnamachari, and R. Jain, "Combinatorial network optimization with unknown variables: Multi-armed bandits with linear rewards and individual observations," *IEEE/ACM Trans. Netw.*, vol. 20, no. 5, pp. 1466–1478, 2012.
[20] N. Cesa-Bianchi and G. Lugosi, "Combinatorial bandits," *J. Comput. Syst. Sci.*, vol. 78, no. 5, pp. 1404–1422, 2012.

[21] B. Kveton, Z. Wen, A. Ashkan, and C. Szepesvari, "Tight regret bounds for stochastic combinatorial semi-bandits," in *Proc. AISTATS*, 2015, pp. 535–543.
[22] B. Kveton, Z. Wen, A. Ashkan, H. Eydgahi, and B. Eriksson, "Matroid bandits: Fast combinatorial optimization with learning," in *Proc. UAI*, 2014.
[23] D. Golovin and A. Krause, "Adaptive submodularity: A new approach to active learning and stochastic optimization." in *Proc. COLT*, 2010, pp. 333–345.
[24] V. Gabillon, B. Kveton, Z. Wen, B. Eriksson, and S. Muthukrishnan, "Adaptive submodular maximization in bandit setting," in *Proc. NIPS*, 2013, pp. 2697–2705.
[25] A. Tewari and P. Bartlett, "Optimistic linear programming gives logarithmic regret for irreducible MDPs," in *Proc. NIPS*, 2008, pp. 1505–1512.
[26] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, pp. 1563–1600, 2010.
[27] A. Zimin and G. Neu, "Online learning in episodic Markovian decision processes by relative entropy policy search," in *Proc. NIPS*, 2013, pp. 1583–1591.
[28] M. Azar, I. Osband, and R. Munos, "Minimax regret bounds for reinforcement learning," in *Proc. ICML*, 2017, pp. 263–272.
[29] S. Agrawal and R. Jia, "Optimistic posterior sampling for reinforcement learning: Worst-case regret bounds," in *Proc. NIPS*, 2017, pp. 1184–1194.
[30] D. Zhao and Y. Zhu, "MEC: A near-optimal online reinforcement learning algorithm for continuous deterministic systems," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 2, pp. 346–356, 2015.
[31] C. J. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3-4, pp. 279–292, 1992.
[32] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, 1988.
[33] T. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Adv. Appl. Math.*, vol. 6, pp. 4–22, 1985.
[34] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, pp. 235–256, 2002.
[35] S. Agrawal and N. Goyal, "Analysis of Thompson sampling for the multi-armed bandit problem," in *Proc. COLT*, 2012, pp. 39.1–39.26.
[36] K. Cheung, W. Ling, C. J. Karr, K. Weingardt, S. M. Schueller, and D. C. Mohr, "Evaluation of a recommender app for apps for the treatment of depression and anxiety: an analysis of longitudinal user engagement," *J. Am. Med. Inform. Assoc.*, vol. 25, no. 8, pp. 955–962, 2018.
[37] N. R. Vempaty, V. Kumar, and R. E. Korf, "Depth-first versus best-first search." in *Proc. AAAI*, 1991, pp. 434–440.
[38] J. Greene, J. H. Hibbard, R. Sacks, V. Overton, and C. D. Parrotta, "When patient activation levels change, health outcomes and costs change, too," *Health Aff.*, vol. 34, no. 3, pp. 431–437, 2015.
[39] H.-J. Xue, X. Dai, J. Zhang, S. Huang, and J. Chen, "Deep matrix factorization models for recommender systems," in *Proc. IJCAI*, 2017, pp. 3203–3209.
[40] T. T. Cheng, "The normal approximation to the Poisson distribution and a proof of a conjecture of Ramanujan," *Bull. Am. Math. Soc.*, vol. 55, no. 4, pp. 396–401, 1949.
[41] Y. Koren, "Factor in the neighbors: Scalable and accurate collaborative filtering," *ACM Trans. Knowl. Discov. Data*, vol. 4, no. 1, Jan. 2010.
[42] Y. Abbasi-Yadkori, D. Pál, and C. Szepesvári, "Improved algorithms for linear stochastic bandits," in *Proc. NIPS*, 2011, pp. 2312–2320.
[43] A. Antos, V. Grover, and C. Szepesvári, "Active learning in heteroscedastic noise," *Theor. Comput. Sci.*, vol. 411, no. 29, pp. 2712–2728, 2010.

Cem Tekin (M'13-SM'20) is currently an Assistant Professor with the Department of Electrical and Electronics Engineering, Bilkent University, Ankara, Turkey. He received the Ph.D. degree in electrical engineering: systems from the University of Michigan, Ann Arbor, MI, USA, in 2013. His research interests include cognitive communications, reinforcement learning, multiarmed bandit problems and multiagent systems.

Sepehr Elahi (STM'18) is currently an undergraduate student in the Department of Electrical and Electronics Engineering and Department of Mathematics, Bilkent University, Ankara, Turkey. His research interests include reinforcement learning, bandit problems and recommender systems.

Mihaela van der Schaar (F09) is currently the John Humphrey Plummer Professor of machine learning, artificial intelligence, and medicine with the University of Cambridge, and a Turing Fellow with The Alan Turing Institute, London. She was a recipient of the NSF Career Award, three IBM Faculty Awards, the IBM Exploratory Stream Analytics Innovation Award, the Philips Make a Difference Award, and several best paper awards, including the IEEE Darlington Award.