Auto-tuning for Cellular Scheduling through Bandit-Learning and Low-Dimensional Clustering

Isfar Tariq, *Member, IEEE,* Rajat Sen, *Member, IEEE,* Thomas Novlan, *Member, IEEE,* Salam Akoum, *Member, IEEE,* Milap Majmundar, *Member, IEEE,* Gustavo de Veciana, *Fellow, IEEE,* and Sanjay Shakkottai, *Fellow, IEEE*

***Abstract*—We propose an online algorithm for clustering channel-states and learning the associated achievable multiuser rates. Our motivation stems from the complexity of multiuser scheduling. For instance, MU-MIMO scheduling involves the selection of a user subset and associated rate selection each time-slot for varying channel states (the vector of quantized channels matrices for each of the users) — a complex integer optimization problem that is different for each channel state. Instead, our algorithm clusters the collection of channel states to a much lower dimension, and for each cluster provides achievable multiuser capacity trade-offs, which can be used for user and rate selection. Our algorithm uses a bandit approach, where it learns both the unknown partitions of the channel-state space (channel-state clustering) as well as the rate region for each cluster along a pre-specified set of directions, by observing the success/failure of the scheduling decisions (e.g. through packet loss). We propose an epoch-greedy learning algorithm that achieves a sub-linear regret, given access to a class of classifying functions over the channel-state space. We empirically validate our approach on a high-fidelity 5G New Radio (NR) wireless simulator developed within AT&T Labs. We show that our epoch-greedy bandit algorithm learns the channel-state clusters and the associated rate regions. Further, adaptive scheduling using this learned rate-region model (map from channel-state to the set of feasible rates) outperforms the corresponding hand-tuned static maps in multiple settings. Thus, we believe that auto-tuning cellular systems through learning-assisted scheduling algorithms can significantly improve performance in real deployments.**

***Index Terms*—Online Learning, Bandit Algorithms, Wireless Networks, Scheduling, Capacity Region, Auto-tuning**

I. Introduction

Wireless cellular networks have become increasingly more complex to operate – the aggregate number of parameters available for optimization at various layers can range in the thousands (e.g. MIMO antenna weights, power levels, coding and modulation rates, and frequency/sub-frame allocation to users), and the choice of which depend on the channel-states of the users[1]. Thus, when scheduling users (e.g. in MU-MIMO scheduling [2]), a *channel-state dependent* combinatorial optimization problem needs to be solved each time-slot, where a subset of users need to be selected, and transmission rates and power levels *jointly* determined for each of these users from among the allowable parameters. This problem however has a latent low-dimensionality that can be exploited, namely that for channel-states that are "near" each other, the optimal solution (user and rate selection) is likely to be the same. Thus, if we cluster channel-states, and determine the effective rate region trade-offs for each cluster, these cluster-dependent rate regions can be used for user and rate selection, and thus significantly reduce the complexity of user and rate scheduling.

However, these clusters are unlikely to be universal, meaning that different scenarios (e.g. indoor, outdoor urban, outdoor rural) would lead to different channel-state clusterings. Indeed, it is also likely that the clusters and associated rate-regions will also vary with the time of day depending on different loading/use-case scenarios. This then, motivates an *online* clustering and multi-user rate region learning approach.

Such learned rate regions are useful in practice. In literature, scheduling algorithms typically assume that they have access to the set of available rates that are feasible for each channel-state. For instance, MaxWeight-like rules [3] that schedule based on the product of the queue-length and channel-rate implicitly assume that the map from the channel-state to the set of feasible channel rates is known, and thus solve an optimization each time-slot (over all feasible rates) resulting in the scheduling decision. These maps from channel-state to feasible rates, in reality, are hand-tuned by operators based on experiments, and these static maps are chosen such that they are good across several deployment scenarios. Instead, our approach of learning the feasible rate-region in each scenario, and thus having a *different* map for each scenario (effectively, an auto-tuning approach for the PHY/MAC scheduler) permits improved performance in deployments. We refer to Simulations Section VI and Section VI-D for more discussion in the setting of the AT&T Labs cellular network simulator.

Main Contributions: The contributions in this paper are two-fold. From a modeling and algorithm development perspective, we develop a clustering model for the wireless downlink, and develop an epoch-greedy bandit algorithm that learns the clusters, the associated capacity regions and schedules users using learned parameters to minimize regret. From a system simulation perspective, we study the benefits of auto-tuning cellular scheduling using such bandit learning, and demonstrate significant benefits on a high-fidelity cellular simulator developed by AT&T Labs.

We consider a system where the channel-state space $\mathcal{P}$ clusters

An earlier version of this paper appeared in the Proceedings of IEEE Infocom 2019 [1].

[1]In a MIMO setting, the channel-state for *each* of the users is the channel H matrix, and in practice the base-station would have access to an approximation of this (e.g. quantized version).

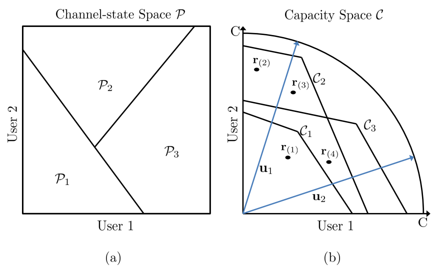


Fig. 1. An illustrative example of the channel-state space $\mathcal{P}$ and the corresponding capacity classes for $n = 2$ users, $K = 3$ capacity classes and $d = 1$. $\{\mathbf{r}_{(i)}\}_{i\in[4]}$ are different rate vectors that can be scheduled. The vectors $\{\mathbf{u}_i\}_{i\in[2]}$ correspond to the directions along which we need to maximize user rates.

into K (unknown) classes, with a corresponding multiuser rate-region for each class. Our goal is to develop *online* strategies that can learn clusterings of different channel-states that have similar multiuser rate regions along with the boundaries of these regions. Simultaneously while learning, we need to schedule users based on the observed channel-states to maximize the user rates along pre-specified directions (see Figure 1(a) precise definition is given in Section III). Our contributions are:

(i) We propose an *epoch-greedy* bandit algorithm for our problem setting. The algorithm assumes access to a class of experts/classifying functions $\hat{\Pi}$, where an expert in $\hat{\Pi}$ is a mapping from the channel-state space to $\{0, 1\}$. We also assume that the class of experts is rich enough, such that there exists a set of functions, which when composed together can yield a function from the channel-state space to $\{1, 2, ..., K\}$ which correctly identifies the class in which each channel-state belongs in. Similar assumptions have been made in the realizable setting in stochastic contextual bandits [4]. Our approach achieves a balance between three objectives: (i) *Class Explore*- learning the clustering of the channel-state space using the class of experts and feedback obtained by scheduling different rates in an exploratory manner (ii) *Capacity Explore*- learning the boundaries of the capacity regions in the specified directions for the different channel-state region clusters (iii) *Exploit* - finally, exploiting the knowledge learned, by scheduling the rate vector of maximum possible magnitude in the specified direction, that lies within the capacity region corresponding to the channel-state observed in a time-slot.

(ii) We consider a notion of *cumulative regret*, where the regret in our setting is the difference between the total effective rate obtained by a learning policy in T time-slots and the total rate obtained by a *genie policy* which knows the capacity clusters and the corresponding capacity regions and given a channel-state, always schedules the rate vector of maximum possible magnitude in the specified direction, which lies within the capacity region corresponding to the channel-state. We provide a rigorous definition of regret for our problem in Equation (4). We analyze our algorithm and prove that it has a regret scaling of $\mathcal{O}(T^{2/3} \log T)$ at time T.

(iii) We perform extensive simulations on a high-fidelity simulator – Wireless Next-Generation Simulation (WiNGS) – developed withing AT&T Labs. WiNGS includes a fully dynamic, event-driven system-level simulator which models both the 5G New Radio (NR) physical layer as well as the air interface protocols including the MAC, RLC, and PDCP sublayers. First, we note that the epoch-greedy bandit algorithm is able to learn channel-state clusters and the associated capacity regions over a short time-scale (30 seconds or less in our settings). Further, the associated scheduler using this learned model is able to match and/or outperform hand-tuned policies in multi-user MIMO settings (both with and without out-of-cell interference). This is because a learning-based algorithm is able to auto-tune to each specific scenario (user locations, interference environment, etc.), whereas static hand-tuned policies are chosen for good average performance across many scenarios. Thus, we believe that such bandit algorithms can play a significant role in auto-tuning wireless cellular systems in real deployments. We refer to Section VI-D for additional details.

Finally, we circle back to one of our motivations – understanding the channel-state-dependent capacity regions. Note that since our algorithms focus on optimizing along a pre-specified set of directions, the resulting capacity region that can be constructed for each channel-state class will be an approximation (because we can potentially miss some of the faces of the capacity region). However, if the capacity regions are "nice", then the direction vectors can be designed in order to get an almost exact estimate of the capacity regions. For instance in [5], it has been shown that convex polytopes formed by the intersection R half-spaces (the hyper-planes should have rational coefficients) and for which the vertex enumeration problem is efficient [6], can be learned with $O(\text{poly}(R, d'))$ *noiseless* membership queries, where d' is the dimension of the space.

II. Related Work

Over the last few decades, there has been a lot of work on opportunistic scheduling for wireless networks. This has led to a powerful framework of algorithms that utilize channel feedback and the queue lengths to achieve objectives like system stability, optimization of a utility function or average delay [3]. In the setting of multi-user MIMO wireless networks (MU-MIMO), scheduling algorithms need to optimize over user selection, beamforming (antenna weight selection), power allocations, physical layer modulation and coding parameters [2], [7], [8]. Here, the user selection sub-problem (choosing a subset of users for transmission from among all the possible users) renders leads to a combinatorial explosion in complexity, and several approximations have been used as guidelines for complexity reduction [9]–[11].

We approach dimensionality reduction through online clustering, and our algorithmic approach is related to the contextual multi-armed bandit problem [12]–[14]. The stochastic contextual bandits with experts problem [4], [13], [15], [16] is especially relevant to our problem. This problem has been studied in the literature starting with the epoch-greedy policy in [13] leading to the more powerful and essentially statistically optimal policies in [4], [15], [16]. Our problem is somewhat similar to this setting as the channel-states observed is analogous to the context and the feedback received after scheduling a rate vector is similar to the stochastic reward observed after pulling an arm. We also assume access to a class of experts that map the space of channel-states to $\{1, 2, ..., K\}$, where K is the number of capacity classes. However, it should be noted that the feedback received in our setting is much more challenging, as it does not provide direct information about the capacity classes unlike the rewards received from the arms in contextual bandits, which directly reflects the utility of that arm under the given context. Moreover, in our problem there is an additional task of learning the boundary of the capacity regions, even after the clustering of the channel-state region into K classes has been learned.

In the context of learning the capacity regions, there is a line of related work on learning convex polytopes which are formed by the intersection of a finite number of half-spaces, from noiseless membership queries [5], [17]. In [5] binary search type strategies have been used to provide efficient algorithms for learning a class of convex polytopes that are formed by the intersection of half-spaces defined by hyper-planes with rational coefficients and for which the vertex enumeration problem can be solved efficiently. Finally, an earlier version of this paper appeared in [1].

III. System Model and Definitions

We consider a discrete time scheduling system with n users and a single scheduler. At each time t, the scheduler observes a channel-state vector $\mathbf{q}(t) = \{\mathbf{q}_1(t), \mathbf{q}_2(t), \ldots, \mathbf{q}_n(t)\}$ where $\mathbf{q}_i(t) \in \mathcal{Q}^d$ is the channel-state for user $i \in [n]$, where $[n] \triangleq \{1, 2, .., n\}$. The set $\mathcal{Q}$ can be a bounded subset of $\mathbb{R}$ or a discrete alphabet set. We denote the set of all channel-state vectors as $\mathcal{P}(= (\mathcal{Q}^d)^n)$. At any time t we observe the channel-state vector $\mathbf{q}$ from a time-invariant distribution $f_{\mathcal{Q}}$ over $\mathcal{P}$ (this distribution depends on the wireless channel between the user and the base-station).

Scheduling a rate vector: Corresponding to each channel-state, there is a unique capacity region that the system can support. The capacity region corresponding to a channel-state is defined as the set of all user rate vectors $\mathbf{r} \in \mathbb{R}^n_+$ that can be achieved with probability close to one, potentially by time-sharing. Strictly speaking, we are really considering the rate region, i.e., the set of user rate vectors that are achievable using the available physical layer strategies at the base-station (convex hull of the data rates that be generated using the available physical layer coding/modulation/antenna-beamforming choices), as opposed to an information-theoretic characterization. We however use the term capacity region instead of rate region for clarity of description.

In our subsequent discussion, when using the phrase "schedule a rate vector $\mathbf{r}$", it means that we notify the PHY/MAC parameter selection algorithm that $\mathbf{r}$ needs to be scheduled. Then, this algorithm tries to achieve the rate $\mathbf{r}$ potentially by time-sharing among various allowable physical layer rates, and over a block of several physical layer time-slots, in which the channel-state remains the same. Finally, at the end of this time-share block, a notification is received which tells us whether the requested rate $\mathbf{r}$ is achieved or not. Therefore, in the subsequent discussion we use 'time-slot' as an abstraction for one trial by the PHY/MAC parameter selection algorithm to achieve a rate over a block of physical layer time-slots. Note that we use a finite length block of physical layer time-slots to judge whether a rate $\mathbf{r}$ can be achieved and therefore the notification is bound to be noisy. This noise is captured in our *noise model* which is described later in this section.

Channel-state Partitions and Capacity Regions: We assume that the channel-state space $\mathcal{P}$ can be partitioned into K *sets* denoted by $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_K$ with their corresponding unique capacity regions $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K$ respectively [2] such that for any $\mathbf{q} \in \mathcal{P}_i$, the capacity region is $\mathcal{C}_i$. In the case where $\mathcal{Q}$ is discrete and finite, it is reasonable to assume that $K \ll |\mathcal{P}| = |\mathcal{Q}|^{nd}$. The capacity regions $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K \subseteq \mathcal{C} \subset \mathbb{R}^n$ are convex polytopes that lie in the positive quadrant. Further, for non-negative vectors $\mathbf{x}, \mathbf{y}$, if $\mathbf{x} \leq \mathbf{y}$ (element-wise) and $\mathbf{y} \in \mathcal{C}_i$, then $\mathbf{x} \in \mathcal{C}_i$ for any $i \in [K]$. We also assume that all the capacity regions lie inside the positive quadrant of the ball with radius C centered at the origin, i.e $\mathcal{C}_i \subset \mathcal{B}(0, C)^+$ for all $i \in [K]$. Here, $\mathcal{B}(0, x) = \{\mathbf{u} \in \mathbb{R}^n : \|\mathbf{u}\|_2 \leq x\}$ and $\mathcal{A}^+$ denotes the subset of $\mathcal{A}$ that lies in the positive quadrant.

We provide an illustrative example in Fig. 1, with $n = 2$ users and $K = 3$ capacity classes. In our example each user provides a one-dimensional channel-state vector, therefore the dimensions of both $\mathcal{P}$ and $\mathcal{C}$ are two. The partitions of the channel-state space $\mathcal{P}$ is shown in Fig. 1(a), which correspond to K different capacity regions in Fig. 1(b). We shall define an index function relating any channel-state vector $\mathbf{q} \in \mathcal{P}$ to the channel-state partition and the capacity region as follows.

Definition 1 (Index function $\mathcal{I}(.)$)**.** *Given a channel-state vector* $\mathbf{q}$*,* $\mathcal{I}(\mathbf{q})$ *is the index of the element of the partition that contains* $\mathbf{q}$*, i.e.* $\mathbf{q} \in \mathcal{P}_{\mathcal{I}(\mathbf{q})}$.

The following assumption states that channel-state's from each element of the partition are observed sufficiently often.

Assumption 1 (Class Probabilities)**.** *We assume that* $\mathbb{P}(\mathcal{I}(\mathbf{Q}) = i) > \beta \; = O\left(\frac{1}{K}\right) \; \forall \; i \in [K]$*, where* $\mathbf{Q} \in \mathcal{P}$ *is a random variable with distribution* $f_{\mathcal{Q}}$ *capturing variability in the system.*

Separation of Capacity Regions: We assume that the capacity regions are sufficiently different from each other. For instance, in Fig. 1 if $\mathcal{C}_1$ and $\mathcal{C}_2$ were almost identical to each other, then it would be better to merge $\mathcal{P}_1, \mathcal{P}_2$ and treat it as a system with $K = 2$. The following assumption says that for any two capacity regions $\mathcal{C}_i, \mathcal{C}_j$ a sufficient fraction of the volume lies outside of their intersection.

[2]Our theoretical guarantees require the channel-state regions corresponding to the different capacity regions be disjoint, however our algorithm can also handle cases where the channel-state classes are not disjoint.

Assumption 2 (Separability)**.** *We assume the capacity regions are well separated, i.e., for all $i, j \in [K]$*

$$d(\mathcal{C}_i, \mathcal{C}_j) \triangleq \frac{|(\mathcal{C}_i \setminus \mathcal{C}_j) \cup (\mathcal{C}_j \setminus \mathcal{C}_i)|}{|\mathcal{B}(0, C)^+|} \geq \lambda > 0,$$

where $|\mathcal{A}|$ denotes the volume of the set $\mathcal{A}$.

Noise Model: Let $Y(\mathbf{q}, \mathbf{r}) \in \{0, 1\}$ denote a random variable modeling the observed notification when a rate $\mathbf{r} \in \mathbb{R}^n$ is scheduled when the observed channel-state vector is $\mathbf{q}$. Here, $Y(\mathbf{q}, \mathbf{r}) = 1$ signifies a successful transmission and $Y(\mathbf{q}, \mathbf{r}) = 0$ signifies a failure to achieve that rate vector . The success or failure to transmit a rate vector $\mathbf{r}$ under channel-state $\mathbf{q}$ is assumed to be an i.i.d random variable $Y(\mathbf{q}, \mathbf{r})$ with distribution given by

$$\mathbb{P}(Y(\mathbf{q}, \mathbf{r}) = 1) = \begin{cases} 1 - \rho(\mathbf{q}, \mathbf{r}), & \text{if } \mathbf{r} \in \mathcal{C}_{\mathcal{I}(\mathbf{q})}, \\ \rho(\mathbf{q}, \mathbf{r}), & \text{if } \mathbf{r} \in \mathcal{B}(0, C)^+ \setminus \mathcal{C}_{\mathcal{I}(\mathbf{q})}, \\ 0, & \text{otherwise.} \end{cases}$$

where $\rho(\mathbf{q}, \mathbf{r})$ can be viewed as a noise parameter (essentially the packet error rate) which depends on the channel-state $\mathbf{q}$ and the rate $\mathbf{r}$.

Assumption 3 (Noise Rate)**.** *We assume that $\rho(\mathbf{q}, \mathbf{r}) \leq \rho < 1/8$, $\forall$ $\mathbf{q}, \mathbf{r}$. We further assume that for all $\mathbf{p}, \mathbf{q}$ and i such that $\mathbf{p}, \mathbf{q} \in \mathcal{P}_i$, $\rho(\mathbf{p}, \mathbf{r}) = \rho(\mathbf{q}, \mathbf{r})$. For notational convenience, for all $\mathbf{q} \in \mathcal{P}_i$, let $\rho_i(\mathbf{r}) \triangleq \rho(\mathbf{q}, \mathbf{r}) = \rho_{\mathcal{I}(q)}(\mathbf{r})$.*

Given a channel-state and the corresponding capacity region, when $\mathbf{r}$ approaches the boundary of the capacity region (from inside) the probability of successful transmission is close to 1 but decreases slightly near the boundary. The success probability drops significantly after $\mathbf{r}$ crosses the boundary (there is a discontinuous jump in success probability at the boundary). After crossing the boundary, $\rho(\mathbf{q}, \mathbf{r})$ decreases till $|\mathbf{r}| = C$, beyond which $\rho(\mathbf{q}, \mathbf{r}) = 0$.

Bandit Feedback and Objectives: Let $\mathcal{U} = \{\mathbf{u}_1, ..., \mathbf{u}_D\}$ be a set of unit vectors such that $\mathbf{u}_i \in \mathbb{R}^n_+$. This set is fixed a priori. The broad objective is to discover the maximum possible service rates in these directions, given a particular channel-state. Since we use 'time-slot' as an abstraction for a block of several physical layer time-slots where a rate vector $\mathbf{r}$ is attempted to be scheduled potentially by time-sharing. Therefore, a wide-range of direction vectors within the capacity region can be supported.

Concurrently with the channel-state, a direction vector $\mathbf{u}$ is chosen uniformly at random from the set $\mathcal{U}$. The task is to schedule a rate vector within the capacity region $\mathcal{C}_{\mathcal{I}(\mathbf{q})}$, of maximum possible magnitude in the direction $\mathbf{u}$. In other words, we would ideally like to schedule a rate vector $c\mathbf{u}$ such that

$$c(\mathbf{q}) = \arg\max_{d} \{d | d\mathbf{u} \in \mathcal{C}_{\mathcal{I}(\mathbf{q})}\}.$$

The precise order of events at a given time-step is as follows:

- A channel-state $\mathbf{q}(t)$ from the distribution f_Q is observed. A direction $\mathbf{u}(t)$ drawn uniformly at random from $\mathcal{U}$ is also specified.
- The policy optionally selects a magnitude $c(\mathbf{q}(t), \mathbf{u}(t)) \in [0, C]$ to be scheduled in the direction $\mathbf{u}(t)$ and the rate vector $\mathbf{r}(t) = c(\mathbf{q}(t), \mathbf{u}(t))\mathbf{u}(t)$ is scheduled. On the other hand, the policy may choose any other rate vector $\mathbf{r}(t)$ that does not lie in the specified direction. In this case the reward obtained is zero in the time-step[3].
- A notification $Y(\mathbf{q}(t), \mathbf{r}(t)) \in \{0, 1\}$ is then observed.

Expected Reward Function: Recall $Y(\mathbf{q}, \mathbf{r})$ is the notification received for transmitting rate vector $\mathbf{r}$ when the observed channel-state was $\mathbf{q}$. Let us define the reward $r(\mathbf{q}, \mathbf{r}, \mathbf{u})$ for a rate vector $\mathbf{r}$, channel-state $\mathbf{q}$ and direction vector $\mathbf{u}$ to be

$$r(\mathbf{q}, \mathbf{r}, \mathbf{u}) = |\mathbf{r}| \mathbb{1}\{\mathbf{r}.\mathbf{u} = |\mathbf{r}|\} Y(\mathbf{q}, \mathbf{r}), \tag{1}$$

where $\mathbb{1}\{\}$ is the indicator function.

Note that for any $\mathbf{p}, \mathbf{q} \in \mathcal{P}_i$, we have $\mathbb{E}[r(\mathbf{q}, \mathbf{r}, \mathbf{u})] = \mathbb{E}[r(\mathbf{p}, \mathbf{r}, \mathbf{u})] \triangleq \mathbb{E}[r_i(\mathbf{r}, \mathbf{u})]$. Therefore, we define the expected reward function $f_{\mathbf{u},i}(c)$ for direction vector $\mathbf{u}$, capacity region $\mathcal{C}_i$ and magnitude c, as follows:

$$f_{\mathbf{u},i}(c) = \mathbb{E}[r_i(c\mathbf{u}, \mathbf{u})]. \tag{2}$$

The function $f_{\mathbf{u},i}(c)$ is the expected rate achieved if we schedule a rate vector $c\mathbf{u}$ when the channel-state observed belongs to capacity class i. It can be evaluated as follows,

$$f_{\mathbf{u},i}(c) = \begin{cases} c(1 - \rho_i(c\mathbf{u})), & \text{if } c\mathbf{u} \in \mathcal{C}_i, \\ c\rho_i(c\mathbf{u}), & \text{if } c\mathbf{u} \in \mathcal{B}(0, C)^+ \setminus \mathcal{C}_i, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Since we have assumed that $\rho_i(\mathbf{r}) < \frac{1}{8}$ $\forall$ $\mathbf{r}$ therefore $f_{\mathbf{u},i}(c)$ is a discontinuous function of c, and the discontinuity is located at the point where a ray in the direction $\mathbf{u}$ meets the boundary of $\mathcal{C}_i$. We make the following assumption on the expected rate function.

Assumption 4 (Maxima of Rate Function)**.** *Let us define*

$$\hat{c}_{\mathbf{u},i} = \arg\max_{c} f_{\mathbf{u},i}(c)$$

and $c^*_{\mathbf{u},i} = \max_c\{c | c\mathbf{u} \in \mathcal{C}_i\}$. We assume that the noise function $\rho_i(\mathbf{r})$ is such that $c^*_{\mathbf{u},i} = \hat{c}_{\mathbf{u},i}$.

This assumption basically implies that for all $i \in [K]$ the maximum of the rate function $f_{\mathbf{u},i}(c)$ is achieved at the point where a ray in the direction $\mathbf{u}$ meets the boundary of $\mathcal{C}_i$.

Class of Experts: We assume access to a class of binary experts/classifiers $\hat{\Pi}$, where each expert $\hat{\pi} \in \hat{\Pi}$ is a function mapping the space of channel-states to $\{0, 1\}$ i.e $\hat{\pi} : \mathcal{P} \to \{0, 1\}$.

Assumption 5 (Classifying Functions)**.** *Let κ be a proper subset of $[K]$. Let us define the following binary function $\hat{\mathcal{I}}_\kappa(\mathbf{q}) = \sum_{i \in \kappa} \mathbb{1}\{\mathbf{q} \in \mathcal{P}_i\}$. We assume that the set of binary*

[3]Note that this is a conservative estimate of the reward. In general, there is some non-zero value in scheduling any rate vector in the capacity region corresponding to the observed channel-state. However, our theoretical guarantees will be under this conservative reward model, and in practice the performance observed will only be better.

experts/classifiers $\hat{\Pi}$ is such that for all $\kappa \subset [K]$, $\hat{\mathcal{I}}_\kappa(.) \in \hat{\Pi}$. We further assume that the VC dimension [18] of our class of experts is V.

The above assumption states that the binary functions from $\mathcal{P}$ to $\{0, 1\}$ that are induced by labeling the channel-state's belonging to a set $\kappa \subset [K]$ of capacity classes as 1 and the rest as 0, are a part of our class of experts, for all such proper subsets κ. Consider the example exhibited in Figure 1. Suppose $\kappa = \{1, 2\}$. Then, $\hat{\mathcal{I}}_\kappa(\mathbf{q})$ divides $\mathcal{P}$ into two regions $\mathcal{P}_1 \cup \mathcal{P}_2$ and $\mathcal{P}_3$. Note that both these regions can be represented as the intersection of at most two half-spaces, as the boundaries of the partitions are linear. This is true for all such proper subsets κ. Therefore, if our class of binary classifiers contains all the separators that are intersections of at most two half-spaces, then Assumption 5 is valid.

Note that $\hat{\mathcal{I}}_\kappa(\mathbf{q})$ for different κ's can be composed together to recover $\mathcal{I}(\mathbf{q})$. In the example in Figure 1, $\hat{\mathcal{I}}_{[2,3]}(\mathbf{q})$ differentiates class 1 from 2, 3 and $\hat{\mathcal{I}}_{[3]}(\mathbf{q})$ separates class 3 from the rest. Given a channel-state $\mathbf{q}$, if for instance $\hat{\mathcal{I}}_{[3]}(\mathbf{q}) = 0$ and $\hat{\mathcal{I}}_{[2,3]}(\mathbf{q}) = 1$ then we can infer that $\mathcal{I}(\mathbf{q}) = 2$. Therefore, Assumption 5 basically implies that there exists a group of binary functions in $\hat{\Pi}$, which when composed together can yield the true index function. Note that this is similar to the *realizable setting* in the contextual bandits with experts problem [4], where it is assumed that the true behavior of the system can be represented by one of the expert function. However, finding the correct expert in an online setting is an algorithmic challenge.

Definition of Regret: The main objective is to minimize regret when compared to a genie strategy which knows the index function $\mathcal{I}$ and the capacity regions $\mathcal{C}_i$'s. Let $\mathbf{r}(t)$ be the rate vector selected by a policy, at time t. Then the regret of the policy till time T is given by:

$$R(T) = \sum_{t=1}^{T} \big(f_{\mathbf{u}(t),\mathcal{I}(\mathbf{q}(t))}(\hat{c}_{\mathbf{u}(t),\mathcal{I}(\mathbf{q}(t))}) \\ - \mathbb{E}\left[r_{\mathcal{I}(\mathbf{q}(t))}(\mathbf{r}(t), \mathbf{u}(t)) \right] \big) \qquad (4)$$

where $\mathbf{q}(t)$,$\mathbf{u}(t)$ are the channel-state vector and direction vector at time t, respectively. Note, that $f_{\mathbf{u}(t),\mathcal{I}(\mathbf{q}(t))}(\hat{c}_{\mathbf{u}(t),\mathcal{I}(\mathbf{q}(t))})$ is the maximum average rate that can be achieved in the direction $\mathbf{u}(t)$, by a *genie policy* that knows the capacity classes and the boundaries of the capacity regions. The regret measures the sub-optimality of the policy in question with respect to the *genie policy*, in an expected sense. The goal is to design a policy that yields $R(T)$ that is sub-linear in T, for all times T large enough. This basically implies that the policy keeps learning the system as time progresses.

IV. Algorithm

The algorithm is structured as an *epoch-greedy* strategy [13]. One key algorithmic idea is that if a rate vector $\mathbf{r}$ is scheduled for several different observed channel-state's $\mathbf{q}$, then the success notifications $Y(\mathbf{q}, \mathbf{r})$ provide useful information that can be leveraged using the class of binary experts $\hat{\Pi}$ to obtain a binary classifier that separates the channel-state space $\mathcal{P}$ into two regions $\mathcal{P}_*$ and $\mathcal{P}_*^c$, where $\mathcal{P}_* = \{\mathbf{q} \in \mathcal{P} : \mathbf{r} \in \mathcal{C}_{\mathcal{I}(\mathbf{q})}\}$. A carefully chosen set of rate points can then be used to form a group of binary classifying functions, which when composed together yields a mapping $\pi : \mathcal{P} \to [K]$, which is identical to $\mathcal{I}(\mathbf{q})$ with high probability.

The algorithm starts with an initialization phase and then proceeds in *epochs*. In initialization phase the algorithm constructs π by building a tree of binary classifiers which is then used to classify the channel-state points into K different classes. This stage is referred to as *initializing classifier* π. After building π, the algorithm runs in epochs similar to epoch-greedy policies for contextual bandits. At the beginning of each epoch, there is a *class explore* stage corresponding to improving the accuracy of classifier π. This is followed by a *capacity explore* stage aimed at learning the capacity regions of the K different channel-state partitions, in the given directions $\mathcal{U}$. The last stage in an epoch is the *exploitation* stage where we deduce the correct capacity class of the observed channel-state vector using π and then schedule the optimal rate vector according to the current belief about the boundary of the corresponding capacity region. An illustrative pseudo-code of our algorithm is shown in Algorithm 1, while a more detailed pseudo-code can be found as Algorithm 4. We will explain each of the stages/phases in more detail in subsequent sections.

Algorithm 1 Epoch-greedy algorithm for online capacity class learning and rate allocation

```
1: Initialize classifier $\pi$, by observing $t_0$ channel-state's, scheduling corresponding carefully designed rate vectors and observing the notifications. *(Initializing Classifier)*
2: Epoch: $l = 1$. Time: $t = t_0$.
3: **while** $t \leq T$ **do**
4:     Update the classifier $\pi$ by observing channel-state $\mathbf{q}$, scheduling a carefully chosen rate point $\mathbf{r}$, and using the notification $Y(\mathbf{q}, \mathbf{r})$. This is repeated $K - 1$ times. *(Class explore)*
5:     Learn the boundaries of the $K$ capacity regions in the directions $\mathcal{U}$, by scheduling carefully chosen rate points and using the current $\pi$. A total of $\alpha(l)$ rate points are scheduled in this part of the epoch. *(Capacity explore)*
6:     Schedule next $s(l)$ rate points optimally using $\pi$ and the learned boundaries. *(Exploit)*
7:     Let $t = t + K - 1 + \alpha(l) + s(l)$ and $l = l + 1$.
8: **end while**
```

A. *Initializing Classifier π*

The first stage of the algorithm is to initialize the mapping (multi-class classifier) π used to classify the different channel-state's into the K different classes. This mapping consists of $K - 1$ binary experts from our class of experts, which are composed together in a *tree-like* structure, in order to yield the mapping π.

The detailed pseudo-code for this phase is provided as Algorithm 2. In the beginning of this phase, for several time-slots the channel-state's are observed and stored, while not making any scheduling decisions (for instance, the scheduler is allowed to proceed in its default behavior). This process is continued until we observe n_0 distinct channel-state vectors, which are essentially n_0 distinct i.i.d random variables sampled from f_Q.

Then we begin initializing the tree-structure which is detailed in steps 2-20 of Algorithm 2. Note that in each iteration of the while loop in step 2 of Algorithm 2, a rate point is randomly selected and then for the following l_0 time-slots irrespective of the channel-state observed, this rate point is scheduled. The feedback observed helps us in building a binary classification data-set that can be used to train a classifier $\hat{\pi} \in \hat{\Pi}$ which can differentiate all $\mathbf{q} \in \mathcal{P}$ such that $\mathbf{r} \in \mathcal{C}_{\mathcal{I}_\mathbf{q}}$ from the rest. We assume that the classifiers are trained in step 11 using empirical risk minimization (ERM) with the $0-1$ loss function. Therefore, we have that:

$$\hat{\pi}_i = \operatorname*{argmin}_{\hat{\pi} \in \hat{\Pi}} \frac{1}{|S_i|} \sum_{(\mathbf{q},y) \in S_i} \mathbb{1}\{\hat{\pi}(\mathbf{q}) \neq y\}.$$

At any point in time, an internal node $\mathcal{N}_i$ in the tree stores the triplet $(\hat{\pi}_i, \mathbf{r_i}, S_i)$ where $\hat{\pi}_i$ is the expert obtained by ERM over the examples $\{(\mathbf{q}, y)\}$ stored in S_i which were in turn obtained by scheduling the rate $\mathbf{r}_i$ for l_0 time-slots. A leaf of the tree $\mathcal{L}_i$ stores a subset of the initial n_0 channel-state points. In each iteration of the while loop, the classifier trained using the data collected by scheduling the current randomly chosen rate point, is only retained if it can split at least one of the current leaf nodes in the tree reliably into two distinct partitions. This is achieved by the check in step 14 of Algorithm 2. The while loop continues to iterate until the tree has $K-1$ internal nodes.

In order to illustrate this phase, let us consider a system as shown in Figure 1 with $r = 2$ users, such that the channel-states can be partitioned into $K = 3$ classes $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$ with capacity regions $\mathcal{C}_1$, $\mathcal{C}_2$ and $\mathcal{C}_3$ respectively.

For, simplicity let $\mathbf{r}_{(1)}, .., \mathbf{r}_{(4)}$ be the first four rate points that are randomly chosen in step 3 of Algorithm 2, in that order (see Fig. 1). Since, $\mathbf{r}_{(1)}$ is a rate point that lies in all the capacity regions, the corresponding classifier $\hat{\pi}_1$ formed using that data collected in step 8, will classify most of the n_0 channel-state points as 1. Therefore, this will not split the current leaf node (the root node with n_0 initial channel-state vectors) into any partitions. Hence, the classifier and the rate point is discarded and the value of the iterator i remains unchanged. The tree remains the same with one leaf node as shown in Fig. 2(a)-(b).

In the next iteration of the while loop, the randomly chosen rate point is $\mathbf{r}_{(2)}$. The data collected using $\mathbf{r}_{(2)}$ is used to train a classifier $\hat{\pi}_1$, which classifies most points in class $\mathcal{P}_2$ as 1, while classifying most points outside of $\mathcal{P}_2$ as zero [4]. This point splits the n_0 channel-state points in the current leaf node into two partitions. Therefore, the classifier is retained. An internal node $\mathcal{N}_1 = \{\mathbf{r}_1, \hat{\pi}_1, S_1\}$ is formed where $\mathbf{r}_1 = \mathbf{r}_{(2)}$.

[4]Note that this is just an initialization of the classifier and moreover the feedback received from scheduling is noisy. Therefore, the binary classifiers trained will not be fully accurate. However, n_0 and l_0 are designed to be large enough such that with high probability the tree structure is correct.

Algorithm 2 Initializing the Classifier Tree

```
1: Schedule arbitrary rate vectors for the first $n_0$ channel-state vectors observed. Let $i = 1$ and form a tree $\mathcal{T}$ where the root contains the $n_0$ initial channel-state points. There are no other nodes in the tree.
2: while $i < K - 1$ do
3:     Randomly select a rate point $\mathbf{r}$.
4:     $S_i = \{\}$
5:     for $l = 1 : l_0$ do
6:         Let $\mathbf{q}$ be the observed channel-state at time-step $t$.
7:         Schedule rate $\mathbf{r}$. (Class Explore)
8:         Let $y \in \{0, 1\}$ be the notification received. Add $(\mathbf{q}, y)$ to $S_i$.
9:         Set $t = t + 1$.
10:    end for
11:    Construct a binary classifier $\hat{\pi}_i$ by empirical risk minimization (ERM) over $S_i$, over the expert set $\hat{\Pi}$.
12:    for all leaves $j$ of $\mathcal{T}$ do
13:        Classify the channel-state at leaf $j$ according to the classifier $\hat{\pi}_i$. Let $n_j$ be the number of channel-state points at leaf $j$.
14:        if $\frac{n_0\beta}{2} <$ number of leaf channel-state classified as $0 < n_j - \frac{n_0\beta}{2}$ then
15:            Make leaf $j$ into a parent of two new leaves where the left leaf has all the channel-state's classified as 1 and the right has all the channel-state's classified as 0.
16:            $i = i + 1$
17:            Break
18:        end if
19:    end for
20: end while
```

Moreover, two leaf nodes are formed where $\mathcal{L}_1$ is a leaf corresponding to all the n_0 channel-state vectors that are labeled as 1 by $\hat{\pi}_1$ and $\mathcal{L}_2$ contains the rest. This is illustrated in Fig. 2(c).

In the next iteration, the rate point $\mathbf{r}_{(3)}$ is chosen, which will effectively yield the same classifier as the one corresponding to $\mathbf{r}_{(2)}$. Therefore, this classifier will be insufficient to split any of the leaves in Fig. 2(c). Thus the value of i is unchanged and the tree remains the same as shown in Fig. 2(d).

Finally, the rate point $\mathbf{r}_{(4)}$ is chosen. The classifier $\hat{\pi}_2$ corresponding to this point ideally distinguishes between points lying in $\mathcal{P}_1$ from those outside of $\mathcal{P}_1$. Thus, this new classifier can split the points in leaf $\mathcal{L}_2$ of the tree in Fig. 2(c), into two nodes, as shown in Fig. 2(d). This leads us to our final classifying tree π. Ideally (ignoring classification errors), a channel-state point belonging to $\mathcal{P}_1$,$\mathcal{P}_2$ and $\mathcal{P}_3$ will land in $\mathcal{L}_3$, $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively.

The parameters n_0, l_0 have been chosen in order to ensure that w.h.p a correct classifying tree is obtained. The following lemma formalizes this claim.

Lemma 1. *Let* $n_0 \geq \frac{24K}{\beta^2} \log\left(\frac{4\log(\frac{1}{\delta})+K}{\delta\lambda}\right)$ *and* l_0 *is large*

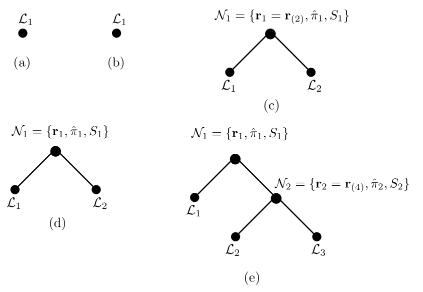

$\mathcal{L}_1$
(a)
$\mathcal{L}_1$
(b)
$\mathcal{N}_1 = \{\mathbf{r}_1 = \mathbf{r}_{(2)}, \hat{\pi}_1, S_1\}$
$\mathcal{L}_1$
$\mathcal{L}_2$
(c)
$\mathcal{N}_1 = \{\mathbf{r}_1, \hat{\pi}_1, S_1\}$
$\mathcal{L}_1$
$\mathcal{L}_2$
(d)
$\mathcal{N}_1 = \{\mathbf{r}_1, \hat{\pi}_1, S_1\}$
$\mathcal{N}_2 = \{\mathbf{r}_2 = \mathbf{r}_{(4)}, \hat{\pi}_2, S_2\}$
$\mathcal{L}_1$
$\mathcal{L}_2$
$\mathcal{L}_3$
(e)

Fig. 2. Construction of a classification tree which represents the final initial classifier π that maps $\mathcal{P} \to [K]$ corresponding to channel-state class structure in Fig. 1.

enough such that $\frac{1}{1-2\rho}\sqrt{\frac{V}{l_0}} + \sqrt{\frac{2\log\left(\frac{l_0^2}{\delta}\right)}{l_0}} < \frac{\beta}{4K}$ *and* $l_0 > \sqrt{\left(\frac{4\log(\frac{1}{\delta})+K-1}{K\lambda}\right)}$. *Then with probability at least* $1 - 3K\delta$, *the loop in step 2 of Algorithm 2 is terminated after at most* $\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ *iterations and further a correct classifying tree structure is obtained.*

B. Class explore

After the classification tree is initialized, the algorithm proceeds in epochs and the structure of the tree remains unchanged. The first few time-slots in each epoch are dedicated to improving the accuracy of the classifiers $\hat{\pi}_i$'s stored in the internal nodes of the tree $\mathcal{N}_i$'s. We name this part of an epoch *class explore*. The *class explore* phase in an epoch consists of $K-1$ time-steps $t_1, ..., t_{K-1}$. At time-step t_i, let the channel-state observed be $\mathbf{q}_i$. After the channel-state is observed, the rate vector $\mathbf{r}_i$ stored in the internal node $\mathcal{N}_i$ is scheduled and a notification y_i is received The data-sample $(\mathbf{q}_i, y_i)$ is added to the set S_i and $\hat{\pi}_i$ is updated through ERM over the updated set S_i. This is performed for all $i = 1, 2, ..., K-1$. This phase is detailed in steps 7 -14 of Algorithm 4. The basic idea is to obtain one more training sample for each of the classifiers stored in the internal nodes, at the beginning of each epoch, thereby improving the classification accuracy of the global classier $\pi : \mathcal{P} \to [K]$. The following lemma provides an upper bound for the classification error of the global classifier $\hat{\pi}$ at the beginning of epoch l.

Lemma 2. *At the end of the class explore phase in epoch* l *with probability at least* $1 - (K-1)\frac{\delta}{(l+l_0)^2}$ *we have*

$$\begin{aligned}&\mathbb{P}(\pi(\mathbf{Q}) \neq \mathcal{I}(\mathbf{Q}))\\ \leq&(K-1)\left(\left(\frac{1}{1-2\rho}\right)\sqrt{\frac{V}{l_0+l}} + \sqrt{\frac{2\log\left(\frac{(l_0+l)^2}{\delta}\right)}{l_0+l}}\right)\\ \triangleq&(K-1)\epsilon(l_0+l, \delta),\end{aligned}$$

where the probability is over the randomness in $\mathbf{Q} \sim f_Q$ *and the randomness in* π *due to the random samples in the training set.*

C. Capacity explore

In each epoch, the class explore phase is followed by a few time-slots dedicated to *capacity explore*. This phase is described as steps 16-22 in Algorithm 4. It is aimed towards learning the boundaries of the K capacity classes in the directions $\mathcal{U}$. Note that there are K possible capacity classes and $D = |\mathcal{U}|$ direction vectors to explore. In the capacity explore phase of epoch l, for $\alpha(l, \delta)$ time-slots we observe the channel-state vectors, direction vectors and schedule carefully designed rate vectors to learn the capacity region. We set $\alpha(l, \delta) = \frac{2D}{\beta}\left(\frac{16}{1-2\rho}\right)^2 \log\left(\frac{l^2}{\delta}\right)$.

We initialize $C_{k,\mathbf{u}}[0] = 0$ and $C_{k,\mathbf{u}}[1] = C$ for all $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$ at the start of the algorithm. $C_{k,\mathbf{u}}[0]$ is a lower bound for $c^*_{\mathbf{u},i}$ and $C_{k,\mathbf{u}}[1]$ is an upper bound for $c^*_{\mathbf{u},i}$, and these values are updated after the capacity explore phase in every epoch.

Algorithm 3 Capacity explore update

```
1: for ∀ $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$ do
2:     if $m_{k,\mathbf{u}} > \frac{1}{2}$ then
3:         $C_{k,\mathbf{u}}[0] = \frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}$
4:     else if $m_{k,\mathbf{u}} < \frac{1}{2}$ then
5:         $C_{k,\mathbf{u}}[1] = \frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}$
6:     end if
7: end for
```

Let $\tau_{l,k,\mathbf{u}}$ be the set of time-slots in which the channel-state $\mathbf{q}$ observed is such that $\pi(\mathbf{q}) = k$ and the direction vector specified is $\mathbf{u}$, in the capacity explore phase of epoch l. In all these time-slots, the rate $\frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}\mathbf{u}$ is scheduled. $m_{k,\mathbf{u}}$ denotes the empirical mean of the success rates in scheduling the above rate vectors. The lower and upper bounds $C_{k,\mathbf{u}}[0]$ and $C_{k,\mathbf{u}}[1]$ are then updated depending on the value of $m_{k,\mathbf{u}}$ for all $k, \mathbf{u}$. The update procedure is detailed in Algorithm 3, which is similar to a traditional binary search procedure for searching the boundary of the capacity regions in the given directions $\mathcal{U}$ (see also [5]).

D. Exploitation

In every epoch, after the *exploration* phases, the overwhelming majority of time-slots are dedicated to *exploitation*. The *exploitation phase* in epoch l consists of $s(l) = O(\sqrt{l})$ time-slots. In each of these time-slots, a channel-state $\mathbf{q}$ is observed and a direction vector $\mathbf{u}$ is specified. The class $k = \pi(\mathbf{q})$ is identified according to our current global classifier and the rate vector $C_{k,\mathbf{u}}[0]\mathbf{u}$ is scheduled. This phase is detailed as steps 24 - 29 in Algorithm 4.

Remark 1. *Algorithm 4 satisfies our regret bound in Theorem 1. However, there are few low-probability failure events that can affect the working of the algorithm in all future time-steps. For instance, the initial classifier tree-structure may be incorrect, which happens with low probability as shown in Lemma 1. Moreover, at any epoch the binary search can take an incorrect decision, which can also happen with a very low-probability. We can generalize the discussion to a more robust*

Algorithm 4 Online rate allocation from channel-state and service data

```
1: Initialize empty sets $S_i = \{\}$ for $i \in [K]$.
2: Initialize a single node tree $\mathcal{T}$ where the node contains $n_0$ different channel-state points.
3: Initialize capacity rate $C_{k,\mathbf{u}}[0] = 0$ and $C_{k,\mathbf{u}}[1] = C$ for all $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$.
4: Initialize classifier $\pi$ using Algorithm 2.
5: Set $t = t_0$ (time index) and $l = 1$ (epoch index).
6: while $t \leq T$ do
7:     for $i = 0 : K - 1$ do
8:         $\mathbf{r}_i$ is the rate vector stored in node $\mathcal{N}_i$.
9:         Let $\mathbf{q}$ be the channel-state observed at time step $t$.
10:        Schedule rate $\mathbf{r}_i$. (Class Explore)
11:        Let $y \in \{0, 1\}$ be the notification received. Add $(\mathbf{q}, y)$ to $S_i$.
12:        Set $t = t + 1$.
13:        Update the classifier $\hat{\pi}_i$ in $\mathcal{N}_i$.
14:    end for
15:    Let the empirical means of success rate be $m_{k,\mathbf{u}} = 0$ for all $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$.
16:    for $s = 1 : \alpha(\delta, l)$ do
17:        Observe $(\mathbf{q}, \mathbf{u})$.
18:        Let $k = \pi(\mathbf{q})$.
19:        Schedule rate vector $\left(\frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}\right)\mathbf{u}$. (Capacity Explore)
20:        Update $m_{k,\mathbf{u}}$ according to received notification $y$.
21:        Set $t = t + 1$.
22:    end for
23:    Update $C$ and $\hat{S}$ according to Algorithm 3.
24:    for $s = 1 : s(l)$ do
25:        Observe $(\mathbf{q}, \mathbf{u})$.
26:        Let $k = \pi(\mathbf{q})$.
27:        Schedule rate vector $C_{k,\mathbf{u}}[0]\mathbf{u}$. (Exploit)
28:        Let $t = t + 1$.
29:    end for
30:    $l = l + 1$.
31: end while
```

algorithm that can detect such low-probability failure states and correct them online (please see Supplementary material). In our simulations in Section VI we use the robust version of the algorithm.

V. Regret Bound

In this section, we provide our main theoretical result which provides a cumulative regret bound for Algorithm 4, when Assumptions 1-5 are satisfied.

Theorem 1. *Under Assumptions 1-5, with probability at least $1 - \xi KD\delta$, Algorithm 4 achieves a regret bound of,*

$$R(T) = \mathcal{O}\left(T^{2/3}\log\left(\frac{1}{\delta}\right)\left(D\log T + K + \sqrt{V}\right)\right),$$

at time T where $\xi < 13$.

Theorem 1 and its proof is available in greater detail in the Supplementary Material of this paper, where the explicit dependence on the various problem parameters has been specified.

Discussion: Theorem 1 states that the regret of Algorithm 4 scales as $\mathcal{O}(T^{2/3}\log T)$ as a function of time. The scaling is linear with respect to the number of classes K and the number of direction vectors D. It scales as $\sqrt{V}$ in terms of the VC dimension of the class of experts. For a finite class of experts $\hat{\Pi}$, the VC dimensions is $\mathcal{O}(\log N)$, where $N = |\hat{\Pi}|$ is the number of experts.

It should be noted that *epoch-greedy* algorithms in bandit settings generally have a regret scaling of $O(T^{2/3})$ in the problem independent setting, because of explicit exploration. For instance, the epoch-greedy strategy in [13] has a similar regret scaling for the problem of stochastic contextual bandits with experts. However, we would like to highlight that our problem setting is significantly more complicated than the usual contextual bandits with experts problem, as in a contextual bandit setting when an arm is pulled under a context, we get a direct feedback about the reward of that arm under that context. However, in our problem setting when a channel-state is observed and a rate vector is scheduled, the received feedback only gives us a partial noisy feedback about the possible capacity class in which the channel-state belongs. The quality of the feedback also depends on the choice of the rate points. Further in our problem setting, even after the capacity classes are learned there is an additional task to recover the boundaries of the corresponding capacity regions. Therefore, the epoch-greedy algorithm proposed in this paper is a first step towards analyzing this setting, and we leave the study of algorithms with implicit exploration that can potentially yield $O(\sqrt{T})$ regret bound as future work.

VI. Simulation Results

In this section we perform empirical simulation of our algorithm on the state-of-the-art Wireless Next-Generation Simulation (WiNGS) platform developed by AT&T Labs.

WiNGS includes a fully dynamic, event-driven system-level simulator which models large-scale cellular network deployments and the L3/L2/L1 protocols layers comprising the 5G New Radio (NR) air interface. Both planned and random deployments of base stations is supported, with users located indoors or outdoors generating traffic according to various finite and full-buffer traffic models. Packets generated by the traffic model pass through the PDCP (Packet Data Convergence Protocol), RLC (Radio Link Control), and MAC (Media Access Control) protocol sublayers where functions including segmentation, re-transmissions, and HARQ (Hybrid Automatic Repeat-reQuest) processes are implemented. The wireless channel is modeled with both long-term effects (e.g. log-normal shadowing, Line-of-Sight vs. Non Line-of-Sight pathloss) and short-term effects (e.g. fast fading due to the environmental scattering and user mobility). The physical layer functionality includes transport block formation based on link adaptation based on channel-state and ACK/NACK feedback. Codebook and channel reciprocity-based digital beamforming is used to generate linear precoders for both single user

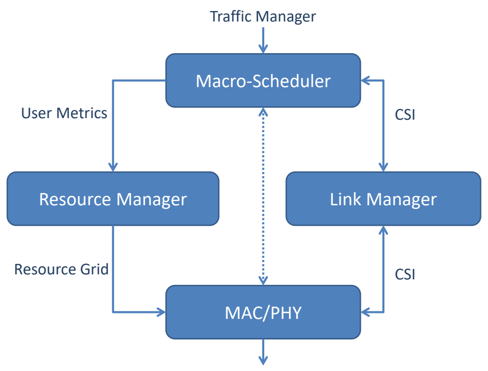


Fig. 3. Block diagram of scheduler module in WiNGS.

(SU) and multi-user MU-MIMO transmission on orthogonal or overlapping time/frequency resources. The probability of success for a transport block which is sent over the wireless channel is based on the post-MMSE receiver SINR to BLER (BLock Error Rate) mapping curves calculated from bit-precise link-level simulation.

Figure 3 provides a block diagram of the modules relevant to our simulations. The AT&T WiNGS scheduler runs in discrete time steps of size 1 ms. At the start of each time step, the traffic manager sends the set of schedulable[5] users and their user metrics to the macro-scheduler. The user metrics consist of information such as CQI (Channel Quality Index, generated every 10 ms), MIMO rank, NDI (New Data Indicator), set of pairable users (users that can be co-scheduled) and their corresponding MU-SINR, etc. A primary user is selected from the set of schedulable users by the macro-scheduler based on a round-robin policy. The macro-scheduler then picks a secondary user for MU-MIMO transmission from set of candidate pairable users. The macro-scheduler then passes the user-pair (primary user and secondary user) and their user-metrics to the resource manager. In the event the macro-scheduler is unable to find a secondary user to pair with primary user then only the primary user and its user metrics are sent to the resource manager. Furthermore it should be noted that for any failed MU transmission, the re-transmissions are sent in SU mode.[6]

For the selected user-pair (or user), the resource manager selects the MCS (Modulation and Coding Scheme) to be used according to the MU-SINR (or SU-SINR). For a given user the mapping of MU-SINR to MCS adapts according to the success/failure of the past transmissions. After selecting the MCS, resource manager fills the resource grid and sends it to the MAC/PHY layer to be scheduled.

To implement our algorithm, we have modified the default resource manager by selecting the MCS to be scheduled for the users according to our algorithm. We use a threshold

[5]Users that have data to be transmitted

[6]A failed MU transmission means either one or both users were unable to decode the packet. Every failed MU transmission can cause 1 or 2 SU re-transmissions based on if MU-transmission to one or both users was unsuccessful.

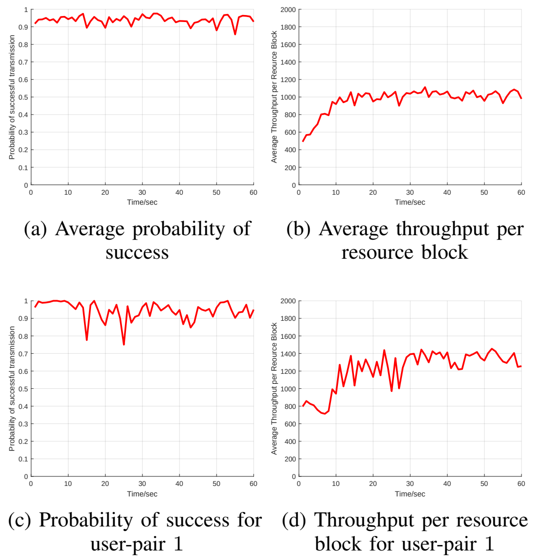


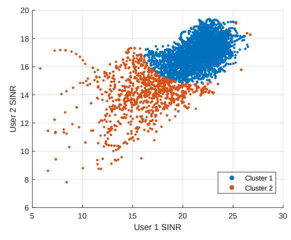
Fig. 4. Single run of our Algorithm for a system with 4 users.

value of $\frac{9}{10}$ instead of $\frac{1}{2}$ for $m_{k,\mathbf{u}}$ in Algorithm 3 since wireless carriers like AT&T requires probability of success to be greater than 90% for all users (desired service requirement). For all of our simulations, we use a robust version of our algorithm with few modifications. Specifically, the class of experts/classifying functions used in our simulations is the naive Bayes implemented in MATLAB. For building the classifier, we test 15 different MCS pairs (rate-vectors) for $l_0 = 50$. The value of n_0 is set to be 3000 (3 sec) and during this time we schedule according to AT&T resource manager. For sake of clarity in results we have excluded the first n_0 time steps from the results. Furthermore, the traffic model used in all simulations is full buffer i.e. all users have data to be transmitted at all time steps.

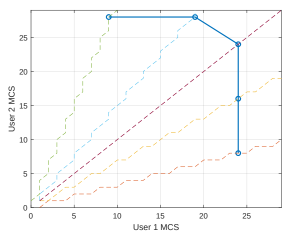
A. Clusters and associated capacity regions

We first plot the result for a single run of our algorithm for a system with one base-station and $n = 4$ users and $d = 5$ direction vectors. At any time-step the direction vector is selected uniformly at random from a set of 5 possible direction vectors. The probability of success and the throughput per resource block is plotted in Figure 4a and 4b respectively. For this system, there are 6 possible user-pairs therefore it take around 4.5 sec for the "build classifiers" phase to complete[7]. We observe that our algorithm is able to converge to optimum within 10 sec and achieve probability of success higher then 90%. In Figure 4c and 4d we plot the probability of success and throughput per resource block for user-pair 1 in the system. We observe that similar to before, our algorithm is able to learn the optimal throughput for user-pair 1 within 10 sec and achieve probability of success higher then 90%.

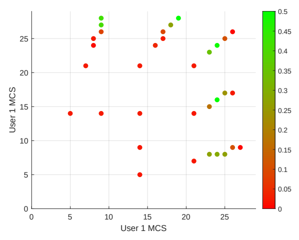
[7]The classifier will be built in 4.5 sec if all user pairs are observed with equal probability. However, it takes more then 4.5 sec for building classifier phase to be completed for all user-pairs since the user-pairs are selected with different probabilities by the system.



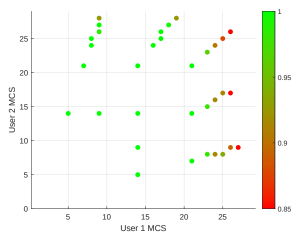
(a) Channel-state clusters for user-pair 1



(b) Acheiveable MCS region for cluster 1

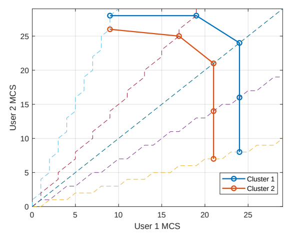


(c) Fraction of transmissions for cluster 1

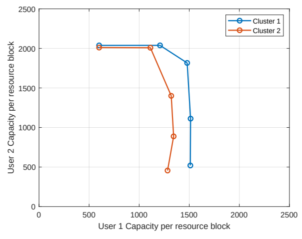


(d) Probability of success for cluster 1

Fig. 5. Single cell results for our Algorithm for user-pair 1.



(a) Acheiveable MCS region



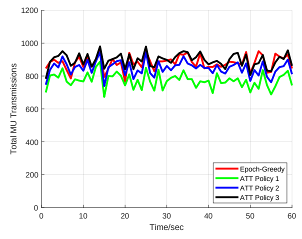
(b) Capacity region

Fig. 6. Achievable MCS region and Capacity region for the different cluster of user-pair 1.

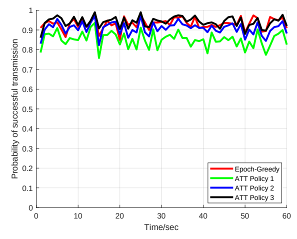
In Figure 5a, we plot the channel-state clusters determined by our algorithm for user-pair 1. We observe that channel-state clusters determined by our algorithm are disjoint and cluster 1 contains better channel-states values then cluster 2. We plot the achievable MCS values of cluster 1 in Figure 5b where the dotted lines show the 5 different direction vectors. The fraction of time different MCS values are scheduled and their corresponding success rate are plotted in Figure 5c and 5d respectively. We observe that while our algorithm explores several different MCS values for any given direction vector, the MCS value that lies on the boundary of achievable MCS region[8] is selected most often.

In Figure 6a, we plot the achievable MCS region for the two channel-state clusters. We observe that the achievable MCS region belonging to cluster 2 is subset of achievable MCS region of cluster 1 because cluster 1 contains better channel-state values as compared to cluster 2. In Figure 6b we plot the capacity regions of both clusters. It should be noted that the capacity region are slightly different from achievable MCS regions since the two users can have different probability of success for any given MCS value.

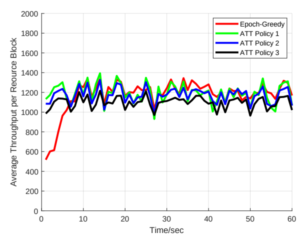
[8]Those having a probability of success exceeding 90%.



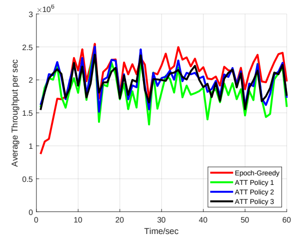
(a) Number of transmissions



(b) Probability of success



(c) Throughput per resource block



(d) Throughput per second

Fig. 7. Performance of different algorithms for Scenario 1 with 4 users. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 9.5%.

B. Single Cell

We consider a single base-station setting and analyze the performance of our algorithm against the state of the art AT&T scheduler. The parameter settings for the AT&T scheduler needs to be manually optimized for every scenario which is impractical in real deployments. Therefore, following practice in real deployments, for the simulations we consider 3 different parameter settings denoted as Policy 1, Policy 2 and Policy 3 respectively, where theses policies are hand-tuned to provide good results for a majority of scenarios. For our algorithm we have a set of $d = 11$ direction vectors and at every time step, the direction vector selected by the scheduler is the direction vector which is closest to the MCS selected by the default AT&T scheduler[9].

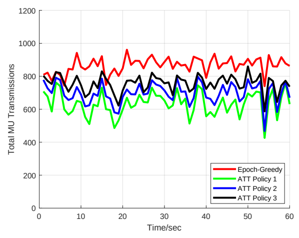
For this setting we first consider 3 different scenarios with a single active base-station and $n = 4$ users each. These scenarios differ in that the user locations are different (and thus, the channel and interference environments differ). For each scenario, we provide plot results for number of MU transmissions, probability of success, throughput per resource block and throughput per second.

The results for scenario 1 are shown in Figure 7. We observe that our algorithm is able to learn the channel-state clusters and suitable MCS values for different direction vectors within 10 seconds. Furthermore when compared to the best performing AT&T policy (Policy 2), our algorithm is able to achieve 9.5% more throughput per second and improve the probability of success from 90.8% to 93.6%. Table I summarizes the

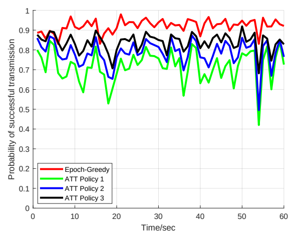
[9]The MCS value selected by the AT&T scheduler is only used to select the direction vector. Our algorithm determines the MCS value to be scheduled along the selected direction vector.

TABLE I
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 1 FOR LAST 30 SEC.

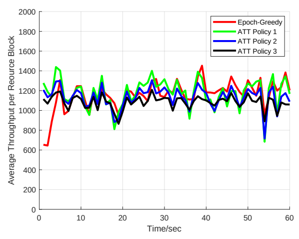
	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	887.7	767.8	845.9	892.6
Average probability of success (%)	93.57	84.74	90.78	94.10
Average throughput per resource block	1221	1184	1173	1100
Average throughput per second ($\times 10^6$)	2.170	1.824	1.988	1.976



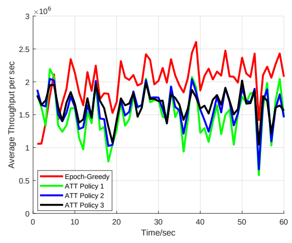
(a) Number of transmissions



(b) Probability of success



(c) Throughput per resource block



(d) Throughput per second

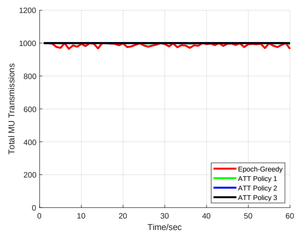
Fig. 8. Performance of different algorithms for Scenario 2 with 4 users. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 30% .

TABLE II
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 2 FOR LAST 30 SEC.

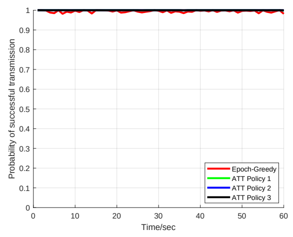
	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	876.0	634.0	698.5	756.8
Average probability of success (%)	92.82	71.68	78.37	83.85
Average throughput per resource block	1219	1172	1127	1083
Average throughput per second ($\times 10^6$)	2.139	1.508	1.588	1.646

performance from 30 to 60 sec for the different algorithms in scenario 1.

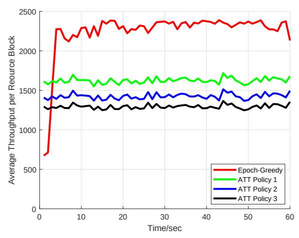
The results for the scenario 2 are shown in Figure 8. Similar to scenario 1 our algorithm is able to learn the channel-state clusters and optimum MCS values for different direction vectors within 10 sec. However unlike scenario 1 the AT&T policy which provides the best average throughput per second is Policy 3. Once again our algorithm outperforms the best AT&T Policy and achieves 30% more throughput per second and improve probability of success from 83.9% to 92.8%. Table II summarizes the performance from 30 to 60 sec for different algorithms in scenario 2.



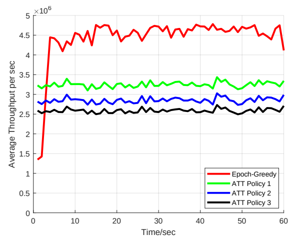
(a) Number of transmissions



(b) Probability of success



(c) Throughput per resource block



(d) Throughput per second

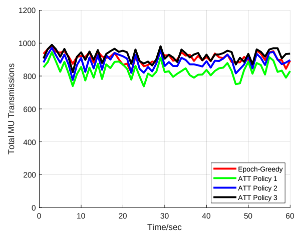
Fig. 9. Performance of different algorithms for Scenario 3 with 4 users. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 41%.

TABLE III
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 3 FOR LAST 30 SEC.

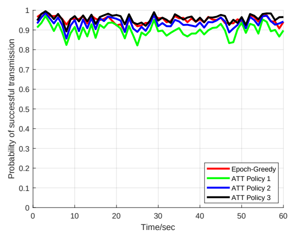
	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	988.2	999.7	999.9	1000
Average probability of success (%)	99.4	100	100	100
Average throughput per resource block	2335	1633	1435	1299
Average throughput per second ($\times 10^6$)	4.615	3.265	2.869	2.597

Figure 9 provides the results for the scenario 3. Similar to previous two scenarios, our algorithm is quickly able to learn the channel-state clusters and optimum MCS values for different direction vectors within 10 sec. For this scenario the AT&T policy which provides the best average throughput per second is Policy 1. Similar to the previous two scenarios, our algorithm significantly outperforms Policy 1 and achieves 41% more throughput per second while still providing probability of success of 99.4%. Table III summarizes the performance of different algorithms in scenario 3 for last 30 sec.

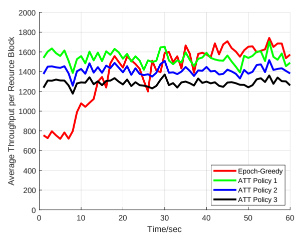
For scenario 4 we consider a single base-station with $n = 7$ users. The results for the scenario 4 are shown in Figure 10. For this scenario, our algorithm takes longer time to learn the channel-state clusters and optimum MCS values for different direction vectors since there are 21 user-pairs in scenario 4 (as compared to 6 for scenarios $1 - 3$). However our algorithm is able to learn optimum value of MCS for different direction vectors within 30 sec. Among the static policies, AT&T Policy



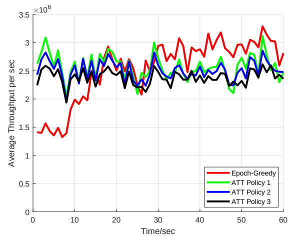
(a) Number of transmissions



(b) Probability of success



(c) Throughput per resource block



(d) Throughput per second

Fig. 10. Performance of different algorithms for Scenario 4 with 7 users. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 14.5%.

TABLE IV
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 4 FOR LAST 30 SEC.

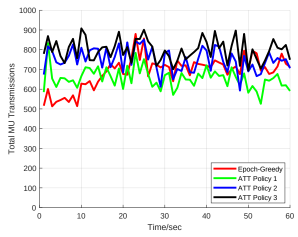
	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	908.4	829.8	886.4	926.9
Average probability of success (%)	94.91	89.62	93.52	96.01
Average throughput per resource block	1603	1530	1408	1287
Average throughput per second ($\times 10^6$)	2.913	2.543	2.497	2.386

1 provides the best average throughput per second for scenario 4, however our algorithm is able to achieves 14.5% more throughput per second and improve the probability of success from 89.6% to 94.9%. Table IV summarizes the performance of different algorithms in scenario 4 during last 30 sec.

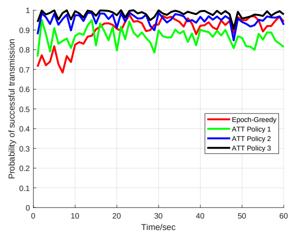
C. Multiple cell

We now consider the multiple base-station setting and analyze the performance of our algorithm against the state of the art AT&T scheduler. For this setting we activate 4 base-stations in the neighborhood of primary base-station. For the AT&T scheduler, we have 3 new parameter setting[10] denoted as Policy 1, Policy 2 and Policy 3. For the simulations with AT&T scheduler we use the same policy on all 5 base-stations. For simulations with our algorithm we only run our algorithm on primary base-station and run AT&T Policy 2 on the 4 neighboring base-stations. In the following, we compare the performance of primary base-station for the different algorithms. Furthermore for our algorithm we have set of $d = 11$ direction vectors, where the direction vectors are

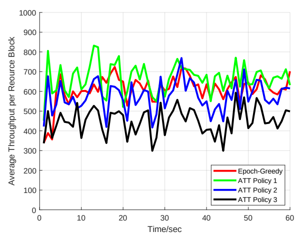
[10]These policies are different from the previous policies since there is interference from the neighboring base-stations.



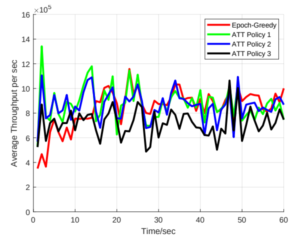
(a) Number of transmissions



(b) Probability of success



(c) Throughput per resource block



(d) Throughput per second

Fig. 11. Performance of different algorithms for Scenario 5 with 4 users connected to primary base-station. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 5%.

TABLE V
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 5 FOR LAST 30 SEC.

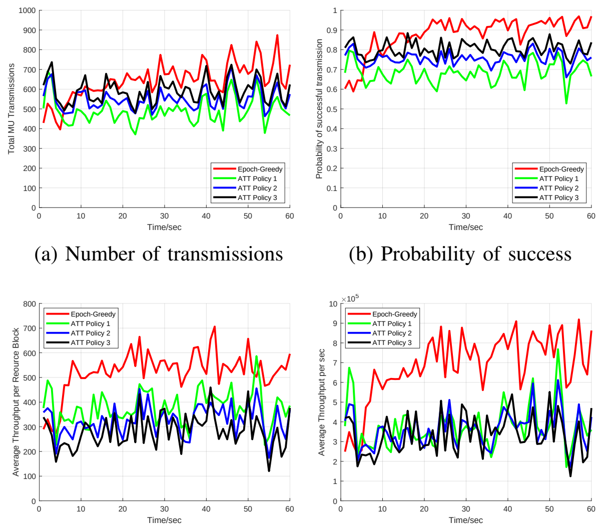
	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	718.9	641.8	731.7	791.0
Average probability of success (%)	93.74	86.17	94.97	98.08
Average throughput per resource block	629.2	674.5	589.3	461.5
Average throughput per second ($\times 10^5$)	9.041	8.601	8.604	7.286

selected using the same method as described for single cell simulations.

We present results for 3 different scenarios denoted by scenario 5, scenario 6 and scenario 7 with 4, 4 and 3 users connected to the primary base-station respectively. For all these scenarios we observe that the average throughput is significantly reduced due to inter-cell interference as compared to average throughput for single cell scenarios.

The results for scenario 5 are shown in Figure 11. We observe that our algorithm is able to learn the channel-state clusters and learn optimum MCS values for different direction vectors within 15 sec. Furthermore, the AT&T policy which provides the best average throughput per second is Policy 2, however our algorithm is able to achieve 5% more throughput per second and have high probability of success of 93.74%. Table V summarizes the performance from 30 to 60 sec for the different algorithms in scenario 5.

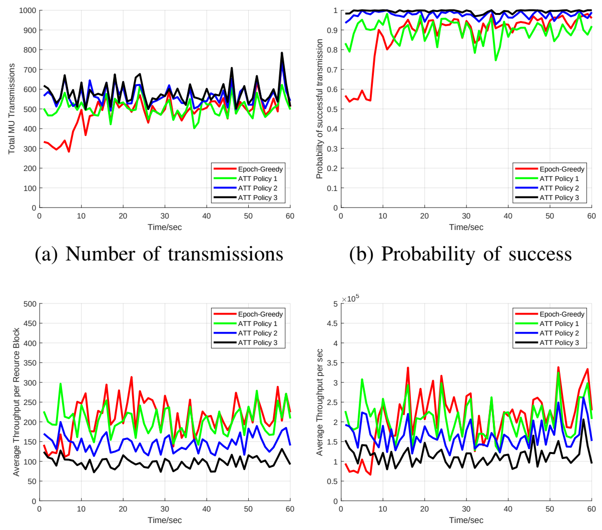
The results for scenario 6 are presented in Figure 12. We observe that similar to scenario 5, our algorithm is able to learn the channel-state clusters and optimum MCS values for different direction vectors within 15 sec. Furthermore our



(a) Number of transmissions (b) Probability of success

(c) Throughput per resource block (d) Throughput per second

Fig. 12. Performance of different algorithms for Scenario 6 with 4 users connected to primary base-station. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 90%.



(a) Number of transmissions (b) Probability of success

(c) Throughput per resource block (d) Throughput per second

Fig. 13. Performance of different algorithms for Scenario 7 with 3 users connected to primary base-station. Our algorithm outperforms the current state-of-art and improves the average throughput per sec by more then 10%.

TABLE VI
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 6 FOR LAST 30 SEC.

	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	694.0	501.6	558.9	598.5
Average probability of success (%)	92.35	69.19	75.92	80.87
Average throughput per resource block	548.0	385.6	331.1	290.3
Average throughput per second ($\times 10^6$)	7.598	3.925	3.733	3.502

TABLE VII
PERFORMANCE OF DIFFERENT POLICIES IN SCENARIO 7 FOR LAST 30 SEC.

	Epoch-greedy	AT&T Policy 1	AT&T Policy 2	AT&T Policy 3
Average number of MU transmissions	525.5	503.6	563.5	580.6
Average probability of success (%)	93.35	88.97	97.31	99.27
Average throughput per resource block	212.2	199.3	146.3	97.5
Average throughput per second ($\times 10^6$)	2.243	2.025	1.657	1.139

algorithm is able to achieve 90% more throughput per second than the best performing AT&T policy while having a high probability of success of 92.35%. Table VI summarizes the performance from 30 to 60 sec for the different algorithms in scenario 6.

Finally we present the results for scenario 7 in Figure 13. Unlike scenario 5 and scenario 6 our algorithm is able to learn the channel-state clusters and optimum MCS values for different direction vectors within 10 sec because there are fewer users connected to primary base-station. For this scenario the AT&T policy which provides the best average throughput per second is Policy 2, however our algorithm achieves 10% more throughput per second and have a high probability of success of 93.35%. The performance from 30 to 60 sec for the different algorithms in scenario 7 is summarized in Table VII.

D. Summary of Results and Discussion

We observe that in all scenarios our algorithm is able to learn the channel-state clusters and their corresponding capacity regions in a short amount of time (10 to 15 sec for a 4 user system, 30 sec for a 7 user system). Furthermore for both single cell and multi cell scenarios our algorithm is able to match if not outperform the current state of art MU scheduling algorithm. It should be noted that the *best performing static policy for the AT&T scheduler to maximize the throughput per sec differs across scenarios.* Further, these policies would need to be continuously hand-tuned if one desires scenario-specific optimal results; additionally we expect that the policy will dynamically change as users enter and leave the system. However, our algorithm is able to learn the correct channel-state to MCS mapping for different user-pairs, which allows it to match if not outperform the current state-of-art. We should also note that even in cases like scenario 3 and scenario 6 shown in Figures 9 and 12 where all policies for the AT&T scheduler policies failed to achieve good throughput, our algorithm was able to learn the system and achieve significantly better results.

For a practical setting, an important metric used by wireless carriers like AT&T is the probability of success of a transmission, where an ideal policy should achieve more then 90% probability of success for all users. Our algorithm is able to ensure a high probability of success within 92%

to 95% for all scenarios. Scenario 3 is an outlier with a high probability of success at 99.4% because the channel-state of users is very good and we are able to transmit the maximum value of MCS successfully with a high probability. We achieve the desired success probability by defining the capacity region as set of MCS values with probability of success of more then 90% and changing the threshold value for $m_{k,\mathbf{u}}$ in Algorithm 3 to $\frac{9}{10}$ (instead of $\frac{1}{2}$) for updating the boundary of capacity regions. Furthermore, we can achieve a higher or lower overall probability of success of system, by increasing or decreasing the threshold value for $m_{k,\mathbf{u}}$ in Algorithm 3. For any fixed AT&T policy the probability of success varies significantly, and cannot be easily controlled. By selecting a static policy (like Policy 3) having more then 90% probability of success for most scenarios, we end up significantly lowering the throughput. Furthermore there will still be cases like scenario 2 or scenario 6 where we have probability of success significantly below 90%. In conclusion, our algorithm is able to provide high throughput as well as ensuring that the probability of success exceeds 90%, thus meeting the desired service requirement (probability of success) goal.

Acknowledgements

This work was supported by NSF grants CNS-1731658, CNS-1718089 and CNS-1910112, Army Futures Command Grant W911NF1920333, the Wireless Networking and Communications Group Industrial Affiliates Program, and the the US DoT supported D-STOP Tier 1 University Transportation Center. The authors would like to sincerely thank Dr. Arunabha Ghosh for discussions on the model as well as providing extensive access to the AT&T WiNGS platform.

References

[1] I. Tariq, R. Sen, G. de Veciana, and S. Shakkottai, "Online channel-state clustering and multiuser capacity learning for wireless scheduling," in *Proceedings of IEEE Infocom*, Paris, France, May 2019.

[2] Y. Du and G. de Veciana, "Wireless networks without edges: Dynamic radio resource clustering and user scheduling," in *Proc. IEEE INFOCOM*, Apr. 2014, pp. 1–9.

[3] R. Srikant and L. Ying, *Communication Networks: An Optimization, Control, and Stochastic Networks Perspective*. Cambridge University Press, 2014.

[4] A. Agarwal, M. Dudík, S. Kale, J. Langford, and R. Schapire, "Contextual bandit learning with predictable rewards," in *Artificial Intelligence and Statistics*, 2012, pp. 19–26.

[5] P. W. Goldberg and S. Kwek, "The precision of query points as a resource for learning convex polytopes with membership queries.," in *Conference on Learning Theory (COLT)*, 2000, pp. 225–235.

[6] D. Avis and K. Fukuda, "A pivoting algorithm for convex hulls and vertex enumeration of arrangements and polyhedra," *Discrete & Computational Geometry*, vol. 8, no. 3, pp. 295–313, 1992.

[7] G. Wunder, M. Kasparick, A. Stolyar, and H. Viswanathan, "Self-organizing distributed inter-cell beam coordination in cellular networks with best effort traffic," in *Proc. 8th Intl. Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, IEEE, 2010, pp. 295–302.

[8] W. Yu, T. Kwon, and C. Shin, "Multicell coordination via joint scheduling, beamforming, and power spectrum adaptation," *IEEE Transactions on Wireless Communications*, vol. 12, no. 7, pp. 1–14, 2013.

[9] T. Yoo and A. Goldsmith, "On the optimality of multiantenna broadcast scheduling using zero-forcing beamforming," *IEEE Journal on selected areas in communications*, vol. 24, no. 3, pp. 528–541, 2006.

[10] X. Xie and X. Zhang, "Scalable user selection for mu-mimo networks," in *INFOCOM, 2014 Proceedings IEEE*, IEEE, 2014, pp. 808–816.

[11] D. Gesbert, M. Kountouris, R. W. Heath, C.-B. Chae, and T. Salzer, "From single user to multiuser communications: Shifting the mimo paradigm," *IEEE signal processing magazine*, vol. 24, no. 5, pp. 36–46, 2007.

[12] A. Slivkins, "Contextual bandits with similarity information," in *Proceedings of the 24th annual Conference On Learning Theory*, 2011, pp. 679–702.

[13] J. Langford and T. Zhang, "The epoch-greedy algorithm for multi-armed bandits with side information," in *Advances in neural information processing systems*, 2008, pp. 817–824.

[14] S. Bubeck, N. Cesa-Bianchi, *et al.*, "Regret analysis of stochastic and nonstochastic multi-armed bandit problems," *Foundations and Trends® in Machine Learning*, vol. 5, no. 1, pp. 1–122, 2012.

[15] A. Agarwal, D. Hsu, S. Kale, J. Langford, L. Li, and R. Schapire, "Taming the monster: A fast and simple algorithm for contextual bandits," in *Intl. Conf. on Machine Learning*, 2014, pp. 1638–1646.

[16] M. Dudik, D. Hsu, S. Kale, N. Karampatziakis, J. Langford, L. Reyzin, and T. Zhang, "Efficient optimal learning for contextual bandits," *arXiv preprint arXiv:1106.2369*, 2011.

[17] S. Kwek and L. Pitt, "Pac learning intersections of halfspaces with membership queries," *Algorithmica*, vol. 22, no. 1-2, pp. 53–75, 1998.

[18] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," in *Measures of complexity*, Springer, 2015, pp. 11–30.

[19] A. K. Menon, B. van Rooyen, and N. Natarajan, "Learning from binary labels with instance-dependent corruption," *arXiv preprint arXiv:1605.00751*, 2016.

Supplementary Material

Appendix A
Robust Algorithm

Algorithm 4 satisfies our regret bound in Theorem 1. However, Algorithm 4 have few *low-probability* failure events that can affect the working of the algorithm in all future time-steps. In this section, we present robust version of our algorithm which detects these low-probability failure events and corrects them online. The pseudo-code for robust algorithm is presented in Algorithm 5. We will discuss the changes made by Algorithm 5 in different phases separately.

Initialization Classifier π: The initialization of classifier π in Algorithm 2 have a low probability failure event, if the set of n_0 initial channel-states used to build the classifier was "bad" i.e. the set of n_0 initial channel-states does not contain sufficient channel-state points from all classes. The robust algorithm, fixes this failure event by reinitializing the set of n_0 channel-state points for the classifier π and rebuilding the classifier π from scratch in the event that classifier π was unable to be built by testing $\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ rate vectors. The pseudo code for the robust initialization of the Classifier Tree is given in Algorithm 6.

Class Explore: We can have a failure event where the classification tree π is incorrect due to a wrong sub classifier present in it. A wrong sub-classifier could be added to classifier π during initialization due to non-zero misclassification error. However, the misclassification error of the sub-classifiers (and also classifierπ) decreases as the epoch number l increases. Therefore in order to detect if the classifier π contains any incorrect sub-classifier, we reclassify the initial n_0 channel-states with the classifier π at each epoch. If the number of channel-states at any leaves node are fewer than $\frac{n_0}{2\beta}$, we will reinitialize the epoch number l, set S_i and $C_{k,u}$ and rebuild the classifier π. The pseudo code for this phase is described in step $7-25$ in Algorithm 5.

Capacity Explore: A low probability failure event that can occur during any epoch, is the binary search making an incorrect decision. This failure event can be solved by testing the upper and lower bounds for capacity region during the capacity explore phase. With this approach the capacity explore will be able to detect and correct for incorrect past decisions in addition learning the boundaries of the K capacity classes in the directions $\mathcal{U}$. The capacity explore phase for the robust algorithm is described as steps $26-34$ in Algorithm 5.

We initialize $C_{k,\mathbf{u}}[0]=0$ and $C_{k,\mathbf{u}}[1]=C$ for all $k\in[K]$ and $\mathbf{u}\in\mathcal{U}$ at the start of the algorithm (same as previous algorithm). However, we will also test the rate vectors $C_{k,\mathbf{u}}[0]\mathbf{u}$ and $C_{k,\mathbf{u}}[1]\mathbf{u}$ in addition of $\frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}\mathbf{u}$. Since we using binary search to update capacity regions, the rate vectors $C_{k,\mathbf{u}}[0]\mathbf{u}$ and $C_{k,\mathbf{u}}[1]\mathbf{u}$ corresponds to the capacity region estimate for previous epochs. Therefore, by scheduling $C_{k,\mathbf{u}}[0]\mathbf{u}$ and $C_{k,\mathbf{u}}[1]\mathbf{u}$ in epoch l we can detect and correct any incorrect decision made in past epochs.

Algorithm 5 Robust online rate allocation from channel-state and service data

```
1: Initialize empty sets $S_i=\{\}$ for $i\in[K]$
2: Initialize a single node tree $\mathcal{T}$ where the node contains $n_0$ different channel-state points
3: Initialize capacity rate $C_{k,\mathbf{u}}[0]=0$, $C_{k,\mathbf{u}}[1]=C$ and stage number $\hat{S}_{k,n}=1$ for all $k\in[K]$ and $\mathbf{u}\in\mathcal{U}$.
4: Initialize classifier $\pi$ using Algorithm 6
5: Set $t=t_0$ (time index) and $l=1$ (epoch index)
6: while $t\le T$ do
7:     for $i=1:K-1$ do
8:         $\mathbf{r}_i$ is the rate vector stored in node $\mathcal{N}_i$
9:         Let $\mathbf{q}$ be the channel-state observed at time step $t$.
10:        Schedule rate $\mathbf{r}_i$. (Class Explore)
11:        Let $y\in\{0,1\}$ be the notification received. Add $(\mathbf{q},y)$ to $S_i$.
12:        Set $t=t+1$.
13:        Update the classifier $\hat{\pi}_i$ in $\mathcal{N}_i$
14:    end for
15:    Reclassify the $n_0$ channel-state points at the root of classifier according to the updated sub classifiers.
16:    for $i=1:K$ do
17:        if number of channel-state point that classify to leaf $i<\frac{n_0}{2\beta}$ then
18:            Reinitialize the single node tree $\mathcal{T}$ where the node contains $n_0$ different channel-state points
19:            Reinitialize $S_i=\{\}$ for $i\in[K]$
20:            Reinitialize $C_{k,\mathbf{u}}[0]=0$, $C_{k,\mathbf{u}}[1]=C$ and stage number $\hat{S}_{k,n}=1$ for all $k\in[K]$ and $\mathbf{u}\in\mathcal{U}$.
21:            Rebuild classifier $\pi$ using Algorithm 6
22:            Epoch number $l=1$
23:            Break
24:        end if
25:    end for
26:    Let the empirical means of success rate be $m_{i,k,\mathbf{u}}=0$ for all $i\in\{-1,0,1\}$, $k\in\{1,2\}$ and $\mathbf{u}\in\mathcal{U}$.
27:    for $s=1:3\alpha(\delta,l)$ do
28:        Observe $(\mathbf{q},\mathbf{u})$.
29:        Let $k=\pi_l(\mathbf{q})$ and $j=Uniform(\{-1,0,1\})$.
30:        Schedule rate vector (Capacity Explore) $\left(\frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}+j\times\left(\frac{C_{k,\mathbf{u}}[0]-C_{k,\mathbf{u}}[1]}{2}\right)\right)\mathbf{u}$.
31:        Update $m_{j,k,\mathbf{u}}$.
32:        Set $t=t+1$.
33:    end for
34:    Update $C$ and $\hat{S}$ according to Algorithm 7
35:    $l=min(\hat{S})$
36:    for $s=1:s(l)$ do
37:        Observe $(\mathbf{q},\mathbf{u})$.
38:        Let $k=\pi_l(\mathbf{q})$.
39:        Schedule rate vector $C_{k,\mathbf{u}}[0]\mathbf{u}$. (Exploit)
40:        Let $t=t+1$.
41:    end for
42:    $l=l+1$.
43: end while
```

Algorithm 6 Robust initialization of the Classifier Tree

```
1: Let $i = 1$ and form a tree $\mathcal{T}$ where the root contains $n_0$ initial channel-state points. There are no other nodes initially.
2: Let $x = 0$.
3: while $i < K$ do
4:     if $x > \frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ then
5:         Schedule arbitrary rate vectors for the next $n_0$ channel-state vectors observed. Reinitialize $i = 1$, $x = 0$ and tree $\mathcal{T}$ where the root contains $n_0$ new channel-state points. There are no other nodes initially.
6:         $t = t + n_0$
7:     end if
8:     Randomly select a capacity point $\mathbf{r}$
9:     $S_i = \{\}$
10:    for $l$ = $1 : l_0$ do
11:        Let $\mathbf{q}$ be the channel-state observed at this time-step.
12:        Schedule rate $\mathbf{r}$. (Class Explore)
13:        Let $y \in \{0, 1\}$ be the notification received. Add $(\mathbf{q}, y)$ to $S_i$.
14:        Set $t = t + 1$.
15:    end for
16:    Construct a binary classifier $\hat{\pi}_i$ by empirical risk minimization (ERM) over $S_i$, over the expert set $\hat{\Pi}$.
17:    for all leaves $j$ of $\mathcal{T}$ do
18:        Classify the channel-state at leaf $j$ according to the classifier $\hat{\pi}_i$. Let $n_j$ be the number of channel-state points at leaf $j$.
19:        if $\frac{n_0\beta}{2} <$ number of leaf channel-state classified as $0 < n_j - \frac{n_0\beta}{2}$ then
20:            Make leaf $j$ into a parent of two new leaves where the left leaf has all the channel-state's classified as 1 and the right has all the channel-state's classified as 0.
21:            $i = i + 1$
22:            Break
23:        end if
24:    end for
25:    $x = x + 1$
26: end while
```

For the robust algorithm, we initialize $m_{i,k,\mathbf{u}} = 0$ for all $i \in \{-1, 0, 1\}$, $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$ at the start of each epoch where $m_{i,k,\mathbf{u}}$ is empirical mean success rate and $i = \{-1, 0, 1\}$ corresponding to rate vectors $C_{k,\mathbf{u}}[1]\mathbf{u}$, $\frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}\mathbf{u}$ and $C_{k,\mathbf{u}}[0]\mathbf{u}$ respectively. In the capacity explore phase we schedule $3\alpha(l, \delta)$ channel-state for epoch l. For a given channel-state vector $\mathbf{q}$ and direction vector $\mathbf{u}$ we use the classifier to find the class $k = \pi(\mathbf{q})$ of the channel-state point. We select $j = Uniform(\{-1, 0, 1\})$ and transmit the rate vector $\left(\frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2} + j \times \left(\frac{C_{k,\mathbf{u}}[0]-C_{k,\mathbf{u}}[1]}{2}\right)\right)\mathbf{u}$ and update $m_{j,k,\mathbf{u}}$. We update the $C_{k,\mathbf{u}}[0]$, $C_{k,\mathbf{u}}[1]$ and $\hat{S}_{k,u}$for all $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$ according to Algorithm 7, which is similar to a traditional binary search procedure.

We will modify the way epoch is computed since Algorithm 7 can detect past incorrect decisions (corresponding to previous epochs) and correct them. The set $\hat{S} = \{\hat{S}_{k,\mathbf{u}}\}$, is initialized to be equal to epoch number for all $k \in [K]$ and $\mathbf{u} \in \mathcal{U}$. We update this set according to the Algorithm 7 and the new epoch number is $l = \min(\hat{S})$.

Algorithm 7 Robust capacity explore update

```
1: for $\forall k \in [K]$ and $\mathbf{u} \in \mathcal{U}$ do
2:     if $m_{-1,k,\mathbf{u}} > \frac{9}{10}$ then
3:         $C_{k,\mathbf{u}}[1] = 2 \times C_{k,\mathbf{u}}[1] - C_{k,\mathbf{u}}[0]$
4:         $\hat{S}_{k,\mathbf{u}} = \hat{S}_{k,\mathbf{u}} - 1$
5:     else if $m_{1,k,\mathbf{u}} < \frac{9}{10}$ then
6:         $C_{k,\mathbf{u}}[0] = 2 \times C_{k,\mathbf{u}}[0] - C_{k,\mathbf{u}}[1]$
7:         $\hat{S}_{k,\mathbf{u}} = \hat{S}_{k,\mathbf{u}} - 1$
8:     else if $m_{0,k,\mathbf{u}} > \frac{9}{10}$ then
9:         $C_{k,\mathbf{u}}[0] = \frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}$
10:        $\hat{S}_{k,\mathbf{u}} = \hat{S}_{k,\mathbf{u}} + 1$
11:    else if $m_{0,k,\mathbf{u}} < \frac{9}{10}$ then
12:        $C_{k,\mathbf{u}}[1] = \frac{C_{k,\mathbf{u}}[0]+C_{k,\mathbf{u}}[1]}{2}$
13:        $\hat{S}_{k,\mathbf{u}} = \hat{S}_{k,\mathbf{u}} + 1$
14:    end if
15: end for
```

Remark 2. *The Algorithm 5 is constructed for the system model presented in section III and accounts for the low probability failure events present in Algorithm 4. However, under additional information such as a discrete capacity region, further improvements can be made to the algorithm. For instance the binary search in Algorithm 7 can be stopped when the upper and lower bound for the capacity regions are within a quantization step apart from each other.*

*Furthermore, similar adjustments can be made to the algorithm in a practical setting where some assumptions presented in section III may no longer be valid. For instance, our system model assumes an i.i.d channel. In a practical system this assumption may not hold, which can potentially cause failure events in the initialization of classifier π. The probability of these failure events can however be reduced by testing the different rate vectors **r** in Algorithm 6 over interleaved time-steps instead of consecutive time-steps.*

Appendix B
Initialization Phase

Before we provide the proof of Lemma 1, we state and prove two lemmas used for proof of Lemma 1.

Lemma 3. *With probability at least $1 - \delta$, the classifier π is constructed by testing $\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ different rate vectors.*

Proof. A rate vector $\mathbf{r}$ is defined to be **good** if the classifier $\hat{\pi}$ built using it splits a leaf in the tree $\mathcal{T}$. Let R be the set of different channel-state classes at leaf $\mathcal{L}$ in the tree and i and j be two channel-state classes in set R. Then a classifier $\hat{\pi}$ constructed using a good **good** rate vector will partition the K classes into set $\mathcal{K}$ and $\mathcal{K}^c$ such that $i \in \mathcal{K}$ and $j \in \mathcal{K}^c$ therefore classifier will split the set R at leaf $\mathcal{L}$. Since $d(C_i, C_j) \geq \lambda$, $\forall\ i,\ j$ and the rate vectors are selected uniformly at random so the probability of selecting a **good** rate vector that can split the set of channel-state classes R into I and J such that $i \in I$, $j \in J$, $I \cup J = R$ and $I \cap J = \emptyset$ is at least λ i.e.

$$\begin{aligned}
&\mathbb{P}(\text{rate vector } \mathbf{r} \text{ is a "good" rate vector})\\
\geq &\mathbb{P}(\hat{\pi} \text{ can split leaf } \mathcal{L} \text{ that contain muliple classes})\\
\geq &\mathbb{P}(\hat{\pi} \text{ can split the set of channel-state } R \text{ at leaf } \mathcal{L} \text{ into } I \text{ and } J)\\
\geq &\lambda
\end{aligned}$$

It should be noted that after $K-1$ such splits, all the channel-state regions can be distinguished since there will be K leaf node and the set of channel-state classes at each leaf node is distinct (due to construction of tree). Therefore each leaf node corresponds to a single and unique channel-state class.

Let $\mathbf{r}_i$ be the i^{th} rate vector tested and let X_i be the event that $\mathbf{r}_i$ can potentially split some leaf in the tree $\mathcal{T}$. Then X_i is a Bernoulli random variable with mean greater or equal than λ. Using multiplicative Chernoff bound for $m > 1$ we have

$$\mathbb{P}\left(\sum_{i=1}^{\frac{(K-1)(m+1)}{\lambda}} X_i < K-1\right) < e^{-\frac{m(K-1)}{4}} \tag{5}$$

Therefore with probability[11] $1-\delta$, in $\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ randomly selected rate vectors there at least $K-1$ **good** rate vectors. □

For, any rate point $\mathbf{r}$ let us define a mapping $\hat{\mathcal{I}}^{(\mathbf{r})}$ from $\mathcal{P} \to \{0,1\}$ such that,

$$\hat{\mathcal{I}}^{(\mathbf{r})}(\mathbf{q}) = \begin{cases} 1, & \text{if } \mathbf{r} \in \mathcal{C}_{\mathcal{I}(\mathbf{q})} \\ 0, & \text{otherwise} \end{cases}$$

.

Thus, if outputs observed by scheduling rate point $\mathbf{r}$ are used to build a binary classifier, an ideal classifier would try to learn the above mapping.

Lemma 4. *Let $\hat{\pi}_i$ be the classifying function returned by the oracle for a training set of l_0 channel-state's for rate vector $\mathbf{r}_i$. Note that $\hat{\mathcal{I}}_i \triangleq \hat{\mathcal{I}}^{(\mathbf{r}_i)}$ is the ideal classification function, that this classifier aims to approximate. Then with probability at least $1-\frac{\delta}{l_0^2}$ we have,*

$$\mathbb{P}(\hat{\pi}_i(\mathbf{Q}) \neq \hat{\mathcal{I}}_i(\mathbf{Q})) \leq \frac{1}{1-2\rho}\sqrt{\frac{V}{l_0}} + \sqrt{\frac{2\log\left(\frac{l_0^2}{\delta}\right)}{l_0}} \triangleq \epsilon(l_0,\delta), \tag{6}$$

where the probability is over the randomness in $\mathbf{Q} \sim f_{\mathcal{Q}}$ and the randomness in $\hat{\pi}_i$ which was trained on the random training set $\mathcal{N}_i$.

Proof. Let $\mathcal{N}_i = \{(\mathbf{q}_1, y_1), ..., (\mathbf{q}_{l_0}, y_{l_0})\}$ be the set of channel-state's and observations from scheduling the point $\mathbf{r}_i$. Let us first define a classification problem in the absence of scheduling noise. Let us define a random variable $\mathbf{Q} \sim f_{\mathcal{Q}}$. Let us define $Y = \mathbb{1}\left(\hat{\mathcal{I}}_i(\mathbf{Q}) = 1\right)$. Let $\mathcal{D}$ be the joint distribution of $(\mathbf{Q}, Y)$. Note that $\mathcal{N}_i$ contains i.i.d samples from a distribution $\mathcal{D}$, however the label for a channel-state point $\mathbf{q}$ is flipped with probability $\rho(\mathbf{q}, \mathbf{r}_i)$. Let us name the distribution with flipped labels by $\tilde{\mathcal{D}}$. For any function $\hat{\pi} \in \hat{\Pi}$, let us define the risk with respect to $\mathcal{D}$ as follows:

$$\mathcal{R}_{\mathcal{D}}(\hat{\pi}) = \mathbb{E}_{\mathcal{D}}\left[\mathbb{1}\left(\hat{\pi}(\mathbf{q}) \neq Y\right)\right]. \tag{7}$$

Let $\hat{\pi}^*_{\tilde{\mathcal{D}}}$ be defined as follows:

$$\hat{\pi}^*_{\tilde{\mathcal{D}}} = \underset{\hat{\pi} \in \hat{\Pi}}{\operatorname{argmin}} \frac{1}{l_0} \sum_{(\mathbf{q},y) \in \mathcal{N}_i} \mathbb{1}\left(\hat{\pi}(\mathbf{q}) \neq y\right).$$

A well-known result from noisy classification [19] states that even though we minimize the above loss function over samples drawn from the distribution $\tilde{\mathcal{D}}$, the function resulting from the minimization has good risk guarantees with respect to the non-noisy distribution $\mathcal{D}$. By Proposition 4 in [19], with at least probability $1-\delta$ we have:

$$\mathcal{R}_{\mathcal{D}}(\hat{\pi}^*_{\tilde{\mathcal{D}}}) - \mathcal{R}_{\mathcal{D}}(\hat{\pi}^*) \leq \frac{1}{1-2\rho}\sqrt{\frac{V}{l_0}} + \sqrt{\frac{2\log\left(\frac{l_0^2}{\delta}\right)}{l_0}} \tag{8}$$

Note $\mathcal{R}_{\mathcal{D}}(\hat{\pi}^*) = 0$ by assumption. This yields the required result. □

Remark 3. *Note that the above result from noisy classification is only true if the Bayes optimal classifier is in our class of classifying functions. This has been assumed in our experts setting.*

Proof of Lemma 1. Let E_1 be the event that in the step 2 in Algorithm 2 the classifier π is built by testing at most $\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ different rate vectors $\mathbf{r}_i$ i.e $K-1$ of these rate vectors are sufficient to split the channel-state region into K correct partitions.

Let $\hat{\pi}_i$ be the classifier built using the rate vector $\mathbf{r}_i$. Let $E_2(i)$ be the event that the classifier $\hat{\pi}_i$ satisfies Eq. (6) in Lemma 4. Given E_1 and $E_2(i)$ for all $i \in \left[\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right]$, at most $\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}$ different classifiers are constructed and for each of the classifiers we have $\mathbb{P}(\hat{\pi}_i(q) \neq \hat{\mathcal{I}}_i(q)) < \epsilon(l_0,\delta)$.

Let $\hat{\pi}_i$ be the classifier used for classifying $\hat{n}$ channel-state points (at some leaf in the classification tree), where the channel-state points are selected at random from $\mathcal{F}$ according to distribution $f_{\mathcal{Q}}$. Let X_j be the indicator random variable that $\hat{\pi}_i(q_j) \neq \hat{\mathcal{I}}_i(q_j)$. Then given $E_2(i)$ the probability of misclassifying more then $2\hat{n}\epsilon(l_0,\delta)$ channel-state is

$$\mathbb{P}(\sum_{j=1}^{\hat{n}} X_j > 2\hat{n}\epsilon(l_0,\delta)) < e^{-\frac{\hat{n}\epsilon(l_0,\delta)}{3}}$$

Let $\hat{A}_i$ be the event that fewer than $n_0\beta\left(1-\frac{1}{2K}\right)$ channel-state point out of n_0 channel-state points at the root node belong to channel-state class i. Then according to multiplicative Chernoff bound

$$\mathbb{P}(\hat{A}_i) < e^{-\frac{n_0\beta}{8K^2}} \qquad \forall\ j \in [K]$$

[11] Where $\delta < e^{-\frac{K}{4}}$

Therefore by union bound

$$\mathbb{P}(\cup_i \hat{A}_i) < \sum_i \mathbb{P}(\hat{A}_i) < Ke^{-\frac{n_0\beta}{8K^2}}$$

Let $E_3(i,\hat{n})$ be the event that the classifier $\hat{\pi}_i$ classified $\hat{n}$ points with less then $2\hat{n}\epsilon(l_0,\delta)$ misclassification and E_4 be the event that all the channel-state classes have at least $n_0 c\left(1-\frac{1}{2K}\right)$ points sampled, then

$$\begin{aligned}\mathbb{P}(E_3(i,\hat{n})|E_1,E_2(i)) >&1-e^{-\frac{\hat{n}\epsilon(l_0,\delta)}{3}}\\ \mathbb{P}(E_4|E_1,E_2(i)\ \forall\ i) >&1-Ke^{-\frac{n_0\beta}{8K^2}}\end{aligned} \tag{9}$$

Let E_5 be the event that all the classifier $\hat{\pi}_i\ \forall\ i \in \left[\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right]$ classify $\hat{n}$ channel-state points in leaf $\mathcal{L}$ with less then $2\hat{n}\epsilon(l_0,\delta)$ misclassification for all leaves in the tree. It should be noted that the classification tree π has at most K leaves. Therefore

$$\begin{aligned}&\mathbb{P}(\bar{E}_5|E_1,E_2(i)\ \forall\ i)\\ <&\mathbb{P}\left(\bigcup_{i=1}^{\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}}\bigcup_{leaves}\bar{E}_3(i,\hat{n})|E_1,E_2(i),E_4\ \forall\ i\right)\\ <&\sum_{i=1}^{\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}}\sum_{leaves}\mathbb{P}(\bar{E}_3(i,\hat{n})|E_1,E_2(i),E_4\ \forall\ i)\\ <&\sum_{i=1}^{\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}}\sum_{leaves}e^{-\frac{\hat{n}\epsilon(l_0,\delta)}{3}}\\ <&K\left(\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right)e^{-\frac{\hat{n}\epsilon(l_0,\delta)}{3}}\end{aligned} \tag{10}$$

The above equations follow owing to the fact that at any point the number of leaves in the tree is less than K.

Let us select l_0 such that $\epsilon(l_0,\delta) < \frac{\beta}{4K}$ (i.e. l_0 is large enough so that $\frac{1}{1-2\rho}\sqrt{\frac{V}{l_0}}+\sqrt{\frac{2\log\left(\frac{l_0^2}{\delta}\right)}{l_0}} < \frac{\beta}{4K}$) i.e. for classifying $\hat{n}$ points, at most $\frac{\hat{n}\beta}{2K}$ points are mis-classified with high probability.

Let us suppose that event E_4 and E_5 hold. Since the tree has height of at most K (as there are only $K-1$ internal nodes) which means that the channel-state points of any channel-state class have to be classified by at most $K-1$ classifiers. As the largest value of $\hat{n}$ is n_0 at the root node so the number of misclassified points of channel-state class j after doing classification with $K-1$ different classifiers is at most $n_0\beta(\frac{K-1}{2K})$. Since the total number of channel-state points for channel-state class j is at least $n_0\beta(1-\frac{1}{2K})$ so after misclassification of $n_0\beta(\frac{K-1}{2K})$ points at least $\frac{n_0\beta}{2}$ points will be correctly classified.

Since the above statement hold for all channel-state classes therefore a leaf j is split into two if and only if the number of channel-state classified as 0 are more then $\frac{n_0\beta}{2}$ and less than $n_j-\frac{n_0\beta}{2}$ where n_j are number of channel-state points at the leaf.

Since the minimum value of $\hat{n}$ is $\frac{n_0\beta}{2}$ i.e. minimum number of channel-state points that can be at any leaf. Therefore

$$\begin{aligned}&\mathbb{P}(\bar{E}_4,\bar{E}_5|E_1,E_2(i)\ \forall\ i)\\ <&\mathbb{P}(\bar{E}_4|E_1,E_2(i)\ \forall\ i)+\mathbb{P}(\bar{E}_5|E_1,E_2(i)\ \forall\ i)\\ <&K\left(\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right)e^{-\frac{n_0\beta^2}{24K}}+Ke^{-\frac{n_0\beta}{8K^2}}\\ <&K\left(\frac{4\log(\frac{1}{\delta})+K}{\lambda}\right)e^{-\frac{n_0\beta^2}{24K}}\\ <&K\delta \qquad \text{for} \quad n_0 \geq \frac{24K}{\beta^2}\log\left(\frac{4\log(\frac{1}{\delta})+K}{\delta\lambda}\right)\end{aligned} \tag{11}$$

Let E_6 be the event that the classifier π is successfully build then

$$\begin{aligned}\mathbb{P}(\bar{E}_6)=\mathbb{P}(\bar{E}_4,\bar{E}_5) <&K\delta+\delta+\left(\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right)\frac{\delta}{l_0^2}\\ <&3K\delta\end{aligned} \tag{12}$$

since $\mathbb{P}(\bar{E}_1)=\delta$ and $\mathbb{P}(\bar{E}_2(i))=\frac{\delta}{l_0^2}$ according to Lemma 3 and Lemma 4 respectively and $l_0 > \sqrt{\left(\frac{4\log(\frac{1}{\delta})+K-1}{K\lambda}\right)}$. □

Appendix C
Class Explore Phase

Outline of proof of lemma2. The proof is pretty straightforward. The classifier π is build using $K-1$ sub-classifiers each of which is trained on l_0+l points. So by using the result of Lemma 4 and taking union bound over all sub classifiers inside classifier π we get the above lemma. □

Appendix D
Capacity Explore Phase

The value of $\alpha(l,\delta)$ is set to be $\frac{2D}{\beta}\left(\frac{16}{1-2\rho}\right)^2\log\left(\frac{l^2}{\delta}\right)$. Let $T_{\mathbf{u},k,l}$ be the number of samples in step 18 of the Algorithm 4 in phase l, such that capacity class observed is k and the direction observed is $\mathbf{u}\in\mathcal{U}$. A simple application of multiplicative Chernoff bound yields,

$$\mathbb{P}\left(T_{\mathbf{u},k,l} \leq \left(\frac{16}{1-2\rho}\right)^2\log\left(\frac{l^2}{\delta}\right)\right) \leq \frac{\delta}{l^2}.$$

Appendix E
Putting it together: Regret Bound

Let $E_7(l)$ be the success event in lemma 2 for phase l. Let $E_8(l)$ be the event $\left\{T_{\mathbf{u},k,l} > \left(\frac{16}{1-2\rho}\right)^2\log\left(\frac{l^2}{\delta}\right)\right\}$ for all k and $\mathbf{u}$. Also, recall the event E_6 which is the success event in Lemma 1.

Lemma 5. *The events $E_7(l)$ and $E_8(l)$ are mutually independent. Moreover, we have,*

$$\begin{aligned}&\mathbb{P}\left(E_6\cap\left(\cap_{l=1}^{\infty}(E_7(l)\cap E_8(l))\right)\right)\\ \geq&1-K(D+1)\sum_{l=1}^{\infty}\frac{\delta}{l^2}-3K\delta.\end{aligned} \tag{13}$$

Note that $m_{\mathbf{u},k}$ are the means collected in step 20 of the algorithm 4. In stage l, we name $S_{\mathbf{u},k,l}$ as a success event which signifies whether $m_{\mathbf{u},k} > \frac{1}{2}$ when $1/2(C_{\mathbf{u},k}[0] + C_{\mathbf{u},k}[1])$ is within $\mathcal{C}_k$ or whether $m_{\mathbf{u},k} < \frac{1}{2}$ when $1/2(C_{\mathbf{u},k}[0] + C_{\mathbf{u},k}[1])$ is outside $\mathcal{C}_k$.

Lemma 6. *We have,*

$$\mathbb{P}\left(S^c_{\mathbf{u},k,l} | E_6 \cap E_7(l) \cap E_8(l)\right) \leq \frac{\delta}{l^2}. \tag{14}$$

This further implies that,

$$\mathbb{P}\left(E_6 \cap (\cap_{l=1}^{\infty}(E_7(l) \cap E_8(l) \cap_{k,\mathbf{u}} S_{k,\mathbf{u},l})\right) \geq 1 - \kappa K D \delta \tag{15}$$

where κ is a suitable constant.

Proof. Consider the scheduling instances $s = \{1, .., \alpha(l, \delta)\}$. Let $s_{(1)}, ..., s_{(T_{\mathbf{u},k,l})}$ be the indices of the time-slots where class is k and direction is $\mathbf{u}$. Let $d_{k,\mathbf{u}}$ be set as $(C_{k,\mathbf{u}}[0] + C_{k,\mathbf{u}}[1])/2$ and assume that $d_{k,\mathbf{u}}$ lies within $\mathcal{C}_k$. We have the following bounds,

$$m_{k,\mathbf{u}} \geq \frac{1}{T_{\mathbf{u},k,l}} \sum_{r=1}^{T_{\mathbf{u},k,l}} Y_{s(r)} \mathbb{1}\left(\pi_l(\mathbf{q}_{s(r)}) = k\right).$$

Now conditioned $\mathbb{E}\left[m_{k,\mathbf{u}} | E_7(l)\right] \geq (1-(K-1)\epsilon(l_0+l, \delta))(1-\rho)$. Note that the random variables in the above summation are i.i.d conditioned on $E_7(l)$. Therefore, by Chernoff's bound we have,

$$\mathbb{P}\left(m_{k,\mathbf{u}} < \mathbb{E}\left[m_{k,\mathbf{u}} | E_7(l)\right] - \gamma | E_7(l)\right) \leq \exp\left(-\gamma^2 \frac{T_{\mathbf{u},k,l}}{2}\right)$$

This yields the following,

$$\begin{aligned}\mathbb{P}\left(m_{k,\mathbf{u}} < (1-(K-1)\epsilon(l_0+l,\delta))(1-2\rho) - \gamma | E_7(l), E_8(l)\right) \\ \leq \exp\left(-\gamma^2 \left(\frac{8}{1-2\rho}\right)^2 \log\left(\frac{l^2}{\delta}\right)\right)\end{aligned}$$

Since $\epsilon(l_0 + l, \delta) \leq \epsilon(l_0, \delta) \leq \frac{\beta}{4K} \leq \frac{1}{5(K-1)}$, therefore setting $\gamma = (1-2\rho)/8$ yields the following,

$$\mathbb{P}\left(m_{k,\mathbf{u}} < \frac{1}{2} \middle| E_7(l), E_8(l)\right) \leq \frac{\delta}{l^2}.$$

Similar results hold for the case when the scheduled rate vector lies outside the capacity region. Therefore, we have the following:

$$\mathbb{P}\left(\cup_{k,\mathbf{u}} S^c_{\mathbf{u},k,l} \middle| E_7(l), E_8(l), E_6\right) \leq \frac{KD\delta}{l^2}.$$

Therefore,

$$\begin{aligned}\sum_l \mathbb{P}\left(\cup_{k,\mathbf{u}} S^c_{\mathbf{u},k,l} | E_6\right) &\leq \sum_l \left(\frac{KD\delta}{l^2} + \frac{\delta}{l^2} + \frac{(K-1)\delta}{(l_0+l)^2}\right) \\ &\leq \sum_l \left(\frac{KD\delta}{l^2} + \frac{K\delta}{l^2}\right) \\ \implies \mathbb{P}\left(\cup_{k,\mathbf{u}} S^c_{\mathbf{u},k,l}\right) &\leq \frac{\pi^2}{6} K(D+1)\delta + 3K\delta.\end{aligned}$$

Combining this with lemma 5 we get the following,

$$\begin{aligned}&\mathbb{P}\left(E_6 \cap \left(\cap_{l=l_0}^{\infty}(E_7(l) \cap E_8(l) \cap_{k,\mathbf{u}} S_{k,\mathbf{u},l}\right)\right) \\ &\geq 1 - \left(\frac{\pi^2}{3} K(D+1)\delta + 6K\delta\right) \\ &\geq 1 - \kappa K D \delta.\end{aligned}$$

□

Theorem 2. *With probability at least $1 - \kappa K D \delta$ the regret of the algorithm is $\mathcal{O}\Big(C T^{2/3} \left(K + \frac{D}{\beta} \log\left(\frac{T}{\delta}\right) + \sqrt{V}\right) + \left(\frac{K^2}{\beta^2}\left(V + \log\left(\frac{1}{\delta}\right)\right)\right)\left(\frac{\log(\frac{1}{\delta}) + K}{\lambda}\right)\Big)$.*

Proof. Let $E = E_6 \cap \left(\cap_{l=l_0}^{\infty}(E_7(l) \cap E_8(l) \cap_{k,\mathbf{u}} S_{k,\mathbf{u},l}\right)$. Given E, at time l we have the following:

$$|C_{k,\mathbf{u}}[0] - c^*_{\mathbf{u},k}| \leq \frac{2C}{2^l}. \tag{16}$$

Let us choose $s(l) = \sqrt{l}$. Let $\mu(l)$ be the expected regret on the exploit slots at time l. It is easy to see that given E,

$$\begin{aligned}\mu(l) \leq &\left((1-\epsilon(l_0+l,\delta)) \times \frac{2C}{2^l} + \epsilon(l_0+l,\delta)C\right) \times \sqrt{l} \\ \leq &2\epsilon(l,\delta)C\sqrt{l} = \frac{C}{1-2\rho}\sqrt{V} + C\sqrt{3\log\left(\frac{l^2}{\delta}\right)}\end{aligned}$$

Let $\gamma(l)$ be the total expected regret at phase l. Then we have the following:

$$\begin{aligned}\gamma(l) \leq &(K-1)C + \alpha(l,\delta)C + \frac{C}{1-2\rho}\sqrt{V} + C\sqrt{3\log\left(\frac{l^2}{\delta}\right)} \\ \leq &C\Big((K-1) + \frac{2D}{\beta}\left(\frac{16}{1-2\rho}\right)^2 \log\left(\frac{l^2}{\delta}\right) \\ &+ \frac{1}{1-2\rho}\sqrt{V} + \sqrt{3\log\left(\frac{l^2}{\delta}\right)}\Big) \\ \leq &C\left((K-1) + \frac{997D}{\beta}\log\left(\frac{l^2}{\delta}\right) + \frac{4}{3}\sqrt{V} + \sqrt{3\log\left(\frac{l^2}{\delta}\right)}\right)\end{aligned} \tag{17}$$

since $\rho \leq \frac{1}{8}$.

Let L^* be the epoch after the end of T time-slots. It is easy to see that $L^* \leq T^{2/3}$. Therefore given E, the expected regret for T time-slots is given by,

$$\begin{aligned}
R(T) \leq & \sum_{l=0}^{L^*} C\left((K-1)+\frac{997D}{\beta}\log\left(\frac{l^2}{\delta}\right)+\frac{4}{3}\sqrt{V}+\sqrt{3\log\left(\frac{l^2}{\delta}\right)}\right) \\
& + n_0 + l_0\left(\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right) \\
\leq & \sum_{l=0}^{T^{\frac{2}{3}}} C\left(K+\frac{997D}{\beta}\log\left(\frac{T^{\frac{4}{3}}}{\delta}\right)+\frac{4}{3}\sqrt{V}+\sqrt{3\log\left(\frac{T^{\frac{4}{3}}}{\delta}\right)}\right) \\
& + \frac{24K}{\beta^2}\log\left(\frac{2\log(\frac{1}{\delta})+K}{\delta\lambda}\right)+l_0\left(\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right) \\
\leq & \sum_{l=0}^{T^{\frac{2}{3}}} C\left(K+\frac{1000D}{\beta}\log\left(\frac{T^{\frac{4}{3}}}{\delta}\right)+\frac{4}{3}\sqrt{V}\right) \\
& + \frac{24K}{\beta^2}\log\left(\frac{2\log(\frac{1}{\delta})+K}{\delta\lambda}\right)+l_0\left(\frac{4\log(\frac{1}{\delta})+K-1}{\lambda}\right) \\
= & \mathcal{O}\Big(CT^{2/3}\left(K+\frac{D}{\beta}\log\left(\frac{T}{\delta}\right)+\sqrt{V}\right) \\
& + \left(\frac{K^2}{\beta^2}\left(V+\log\left(\frac{1}{\delta}\right)\right)\right)\left(\frac{\log(\frac{1}{\delta})+K}{\lambda}\right)\Big) \qquad (18)
\end{aligned}$$

□