CS272 - Theoretical Foundations of Learning, Decisions, and Games

Lecture 15: Reductions for Swap Regret

October 21, 2025

Lecturer: Nika Haghtalab *Readings: Chapter 18, Twenty Lectures on AGT (Roughgarden)*
Scribe: Denise Cerna, Sam Huang, Samuel Xu

1 Motivation

In previous lectures we introduced correlated equilibrium (CE) as one of the central solution concepts in learning in games. The main reason for starting with CE is that its definition is built around the idea of *swapping* actions, and we have already seen that no–swap-regret dynamics give correlated equilibria. Formally, recall the definition:

Definition 1.1 (Correlated Equilibrium)**.** A distribution π over $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ is a *correlated equilibrium* (CE) if and only if for all players $i \in [n]$ and for all *swap functions* $\delta : \mathcal{A}_i \to \mathcal{A}_i$,

$$\mathbb{E}_{a\sim\pi}[u_i(a)] \;\geq\; \mathbb{E}_{a\sim\pi}[u_i(\delta(a_i), a_{-i})].$$

An equivalent formulation using conditional probability is that for every player i and every action $a' \in \mathcal{A}_i$,

$$\mathbb{E}_{a\sim\pi}[u_i(a)] \;\geq\; \mathbb{E}_{a\sim\pi}[u_i(a', a_{-i}) \mid a_i].$$

Intuitively, then, a correlated equilibrium is a distribution over joint actions such that no player can profit by applying any mapping δ to their recommended action, even when these mappings depend arbitrarily on what they were told to play. This "no profitable swap" condition is exactly what no–swap-regret learning guarantees in the long run. We will use this perspective to motivate the focus of today's lecture: how to design learning algorithms whose long-run behavior satisfies these inequalities, and hence achieves no swap regret.

2 Swap Regret

Definition 2.1 (Swap Regret)**.** Given a sequence of utilities $u^1, \ldots, u^T$, the *swap regret* of an action sequence $a^1, \ldots, a^T$ is

$$\text{Swap-Regret}_T = \max_{\delta:\mathcal{A}\to\mathcal{A}} \sum_{t=1}^{T} \big(u^t(\delta(a^t)) - u^t(a^t)\big).$$

This equation defines the swap regret as the maximum, over all swap functions, of the difference between the utility obtained by deviating according to a swap function δ and the utility achieved by the algorithm. An algorithm is said to be *no-swap-regret* if

$$\lim_{T\to\infty} \frac{\text{Swap-Regret}_T}{T} = 0.$$

External Regret. Moving forward, we will refer to notion of regret we saw in previous lectures as the external regret (or static regret):

$$\text{Ext-Regret}_T = \max_{k \in \mathcal{A}} \sum_{t=1}^{T} \big(u^t(k) - u^t(a^t)\big).$$

Clearly, $\text{Swap-Regret}_T \geq \text{Ext-Regret}_T$, since a constant swap function $\delta(a) = k$ for all a recovers the external regret:

$$\max_{\delta:\text{constant}} \sum_{t=1}^{T} \big(u^t(\delta(a^t)) - u^t(a^t)\big) = \max_{k \in \mathcal{A}} \sum_{t=1}^{T} \big(u^t(k) - u^t(a^t)\big) = \text{Ext-Regret}_T.$$

Table 1 highlights why swap regret always upper bounds external regret. External regret considers only constant deviations, which is shown in the second column. Swap regret, however, allows much richer deviations. You can replace each action with a different one, as illustrated by the non-constant mappings in the remaining columns. Since constant mappings are just a special case of swap functions, the best swap-based deviation can only perform better (in hindsight) than the best constant deviation. Thus, swap regret is always at least as large as external regret.

t	a^t	$\delta(a^t) = D$	Swap: $A \to B$	Swap: $A \to B$, $B \to C$
1	A	D	B	B
2	B	D	B	C
3	A	D	B	B
4	C	D	C	C
5	A	D	B	B
6	B	D	B	C
7	D	D	D	D

Table 1: Example illustrating swap functions and how constant swaps recover external regret.

3 Reduction from Swap to External Regret [Blum and Mansour, 2007]

3.1 Construction Idea

We instantiate n copies of a no-external-regret algorithm, denoted $M_1, \ldots, M_n$.

(i) Each M_j is responsible for protecting against deviations from action j to others.

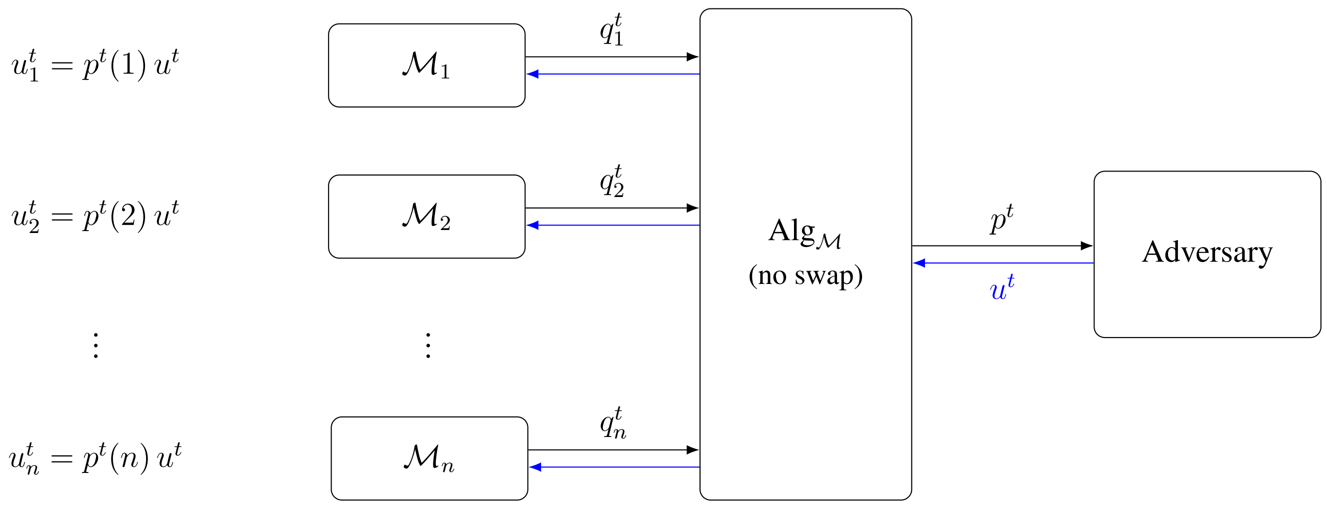


Figure 1: Reduction Overview

(ii) At time t, each M_j outputs a distribution q_j^t over actions.

(iii) With the information of q_1^t, ..., q_n^t, the master algorithm Alg_M updates its mixed strategy

$$p^t = \big(p^t(1), \dots, p^t(n)\big), \qquad p^t(i) = \text{probability of playing action } i \text{ at time } t.$$

(iv) The reward for each action i at time t is given by $u^t(i)$.

There are two questions we will explore. The first is how to feed back rewards to the sub-algorithms M_j so that they collectively simulate a no-swap-regret learner. The second is how to design p_t as a function of $q_1 \dots q_n$.

3.2 Reward Passing

Each sub-algorithm M_j receives a scaled utility vector:

$$u_j^t = p^t(j) \cdot u^t,$$

so that its feedback depends on how often the master algorithm chose j. These dynamics in shown in Figure 1.

Utility of the master algorithm.

$$\mathcal{U}(\mathsf{Alg}_M) = \sum_{t=1}^{T} \mathop{\mathbb{E}}_{i \sim p^t} [u^t(i)] = \sum_{t=1}^{T} \sum_{i=1}^{n} p^t(i) u^t(i). \tag{1}$$

Utility under a swap function δ.

$$\mathcal{U}(\mathsf{Alg}_M, \delta) = \sum_{t=1}^{T} \mathop{\mathbb{E}}_{i \sim p^t}[u^t(\delta(i))] = \sum_{t=1}^{T} \sum_{i=1}^{n} p^t(i) u^t(\delta(i)). \tag{2}$$

Our goal is to ensure that $\text{Alg}_{\mathcal{M}}$ has no swap regret. In other words:

$$\mathcal{U}(\mathsf{Alg}_M) \geq \mathcal{U}(\mathsf{Alg}_M, \delta) - o(T) \qquad \forall \delta : \mathcal{A} \to \mathcal{A}.$$

3.3 Analysis from the Perspective of M_j

Algorithm M_j observes utilities scaled by $p^t(j)$, that is,

$$u_j^t(i) = p^t(j)\, u^t(i),$$

and thus its cumulative utility is

$$\begin{aligned} \mathcal{U}_j(\mathsf{Alg}_{M_j}) &= \sum_{t=1}^{T} \mathop{\mathbb{E}}_{i \sim q_j^t}[u_j^t(i)] = \sum_{t=1}^{T} \mathop{\mathbb{E}}_{i \sim q_j^t}[u^t(i)\, p^t(j)] \\ &= \sum_{t=1}^{T} \sum_{i=1}^{n} q_j^t(i)\, u^t(i)\, p^t(j). \end{aligned} \tag{3}$$

Since M_j has no external regret, no deviation to arm k is beneficial. For any fixed deviation to k, we have

$$\begin{aligned} \mathcal{U}_j(\mathsf{Alg}_{M_j}) &\geq \sum_{t=1}^{T} u_j^t(k) - \text{Regret}_T(M_j) \\ &= \sum_{t=1}^{T} p^t(j)\, u^t(k) - \text{Regret}_T(M_j), \quad \forall k \in [n]. \end{aligned} \tag{4}$$

Given a swap function δ, define $k_j = \delta(j)$. Summing (4) over all $j = 1, \dots, n$ gives

$$\sum_{j=1}^{n} \mathcal{U}_j(\mathsf{Alg}_{M_j}) \geq \sum_{j=1}^{n} \sum_{t=1}^{T} p^t(j) u^t(\delta(j)) - \sum_{j=1}^{n} \text{Regret}_T(M_j). \tag{5}$$

Then, by Equation 2, we have

$$\sum_{j=1}^{n} \sum_{t=1}^{T} \sum_{i=1}^{n} q_j^t(i) p^t(j) u^t(i) \geq \mathcal{U}(\mathsf{Alg}_M, \delta) - \sum_{j=1}^{n} \text{Regret}_T(M_j).$$

We want the left-hand side to match $\mathcal{U}(\mathsf{Alg}_M)$. To achieve that, it suffices to ensure that

$$\sum_{j=1}^{n} p^t(j) q_j^t(i) = p^t(i), \forall i \in [n].$$

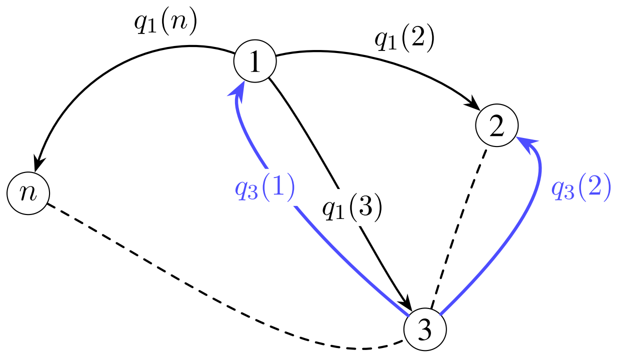


Figure 2: Interpretation as a Markov Chain

In matrix form, this requires

$$\boldsymbol{p}^{t\top} Q^t = \boldsymbol{p}^{t\top}.$$

This means $\boldsymbol{p}^t$ is a *stationary distribution* of the stochastic matrix Q^t whose (j,i)-th entry is $q_j^t(i)$.

3.4 Interpretation as a Markov Chain

Each node j "recommends" another action according to q_j^t. If we think of Q^t as defining a Markov chain over $\mathcal{A}$, then $\boldsymbol{p}^t$ is its stationary distribution:

$$\boldsymbol{p}^{t\top} Q^t = \boldsymbol{p}^{t\top}.$$

Existence is guaranteed since any stochastic matrix has at least one stationary distribution; moreover, we can compute $\boldsymbol{p}^t$ as the left eigenvector of Q^t corresponding to eigenvalue 1.

4 Reduction For Large n

This section aims to provide some intuition for dealing with the case for large n, but not the detail.

4.1 From External to Swap Regret

It is well-known that using the multiplicative weights update (MWU) algorithm, the *external regret* after T rounds satisfies

$$\text{Ext-Regret}_T = O(\sqrt{T \log n}).$$

Equivalently, to achieve ϵ-external-regret, we require a horizon

$$T^{\text{ext}} \geq O\left(\frac{\log n}{\epsilon^2}\right).$$

In the reduction of section 3, the total swap regret bound accumulates the external regret across all local modules M_j (or layers). Hence, to guarantee ϵ-swap-regret, it suffices to have

$$T^{\text{swap}} \geq O\left(\frac{n \log n}{\epsilon^2}\right).$$

This bound is undesirable when n is large; indeed, when $n = \infty$ (continuous or combinatorial actions), the dependence is unbounded. This undesirable bound comes from the fact that each algorithm in the support of the stationary distribution is contributing its regret.

When the action space $\mathcal{A}$ has finite Littlestone dimension LDim, prior work on external regret ensures that

$$T^{\text{ext}} \geq O\left(\frac{\mathrm{LDim}}{\epsilon^2}\right)$$

suffices for ϵ-external-regret. However, a naive reduction to swap regret via independent external learners yields no finite bound. Thus, our question is

> Can we get q^t's such that the resulting stationary distribution (across all t) has small support while q^t's are still no external regret?

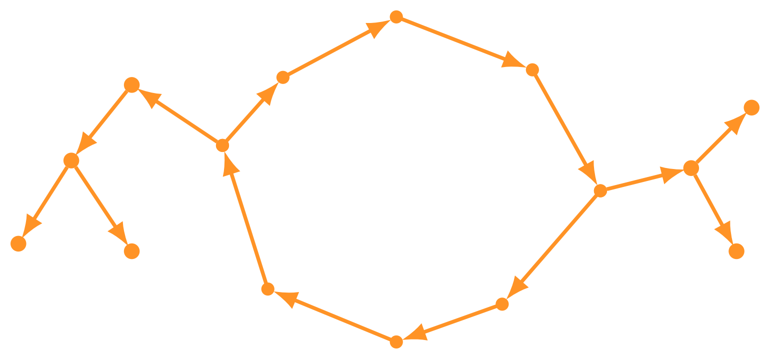
4.2 High-Level Intuition

The Tree-Swap algorithm provides a structural reduction from external to swap regret. As before, we can view the algorithm through the lens of Markov chains. Each node j corresponds to a base learner M_j that maintains a local distribution over actions and recommends another node $\delta(j)$. Below we illustrate the key challenges behind the construction of the Tree-Swap algorithm.

Key Challenges.

1. Solving $\boldsymbol{p}^\top Q = \boldsymbol{p}^\top$ relates to the adversary's action; it's hard to know whether we can guarantee small-support stationary distributions.

2. Constructing stationary distributions is also quite abstract. So let's make it less abstract by understanding what they look like. If the out-degree of every node is one, the directed graph $(j, \delta(j))$ decomposes into cycles plus trees rooted in the cycles like the following,



and the stationary distribution is uniform over the nodes of each cycle.

The Tree-Swap construction [Dagan et al., 2024] overcomes this barrier by implicitly constructing cycles with length that is far less than n, so the stationary distribution has small support and the Swap-regret only blows up by the length of cycles, not n.

4.3 Lazy MWU and Layered Regret Decomposition

To reduce computational overhead, the Tree-Swap algorithm employs *lazy multiplicative weights updates*, grouping rounds into blocks of size B:

$$\underbrace{a^1, \dots, a^1}_{\text{block 1}}, \underbrace{a^2, \dots, a^2}_{\text{block 2}}, \dots, \underbrace{a^M, \dots, a^M}_{\text{block } M}.$$

Each block corresponds to one update step of the local MWU learners, allowing amortized updates across B rounds. Since there are fewer blocks of time than T, the average external regret

$$\frac{1}{T}\text{Ext-Regret}_T^{\text{lazy}} = O(\sqrt{B \log n / T}) = O\Big(\sqrt{\frac{\log n}{M}}\Big).$$

Specifically, Tree-Swap organizes d layers of no-external-regret learners $\{M^{(1)}, \dots, M^{(d)}\}$ in a hierarchical tree. Each learner $M^{(i)}$ is a *lazy MWU* instance, updated only once every M^{d-i} rounds, producing a sequence $\{a_t^{(i)}\}$. At time t, the overall algorithm Alg_M chooses the mixture

$$a_t = \frac{1}{d} \sum_{i=1}^{d} a_t^{(i)},$$

where each $a_t^{(i)} \in \Delta(\mathcal{A})$ is the current distribution of layer i. And we choose iterations $T = M^d$.

Regret Decomposition. The Tree-Swap construction interprets the swap mapping as a collection of local decisions made along a depth-d tree: each layer i learns a mapping $\delta^{(i)}$ from its current node to a next node, defining a directed graph where edges represent local swaps.

At time step t, the i-th layer maintains a distribution $a_t^{(i)} \in \Delta(\mathcal{A})$, and the global algorithm Alg_M plays the averaged action

$$a_t = \frac{1}{d} \sum_{i=1}^{d} a_t^{(i)}.$$

Each layer observes a scaled version of the reward vector $u_t \in [0, 1]^n$, defined as

$$u_t^{(i)} = \frac{1}{d} u_t,$$

so that the total reward of the full algorithm can be written recursively as

$$\text{Reward} = \sum_{t=1}^{T} \frac{1}{d} \sum_{i=1}^{d} \langle a_t^{(i)}, u_t \rangle = \sum_{i=1}^{d} \sum_{t=1}^{T} \langle a_t^{(i)}, u_t^{(i)} \rangle.$$

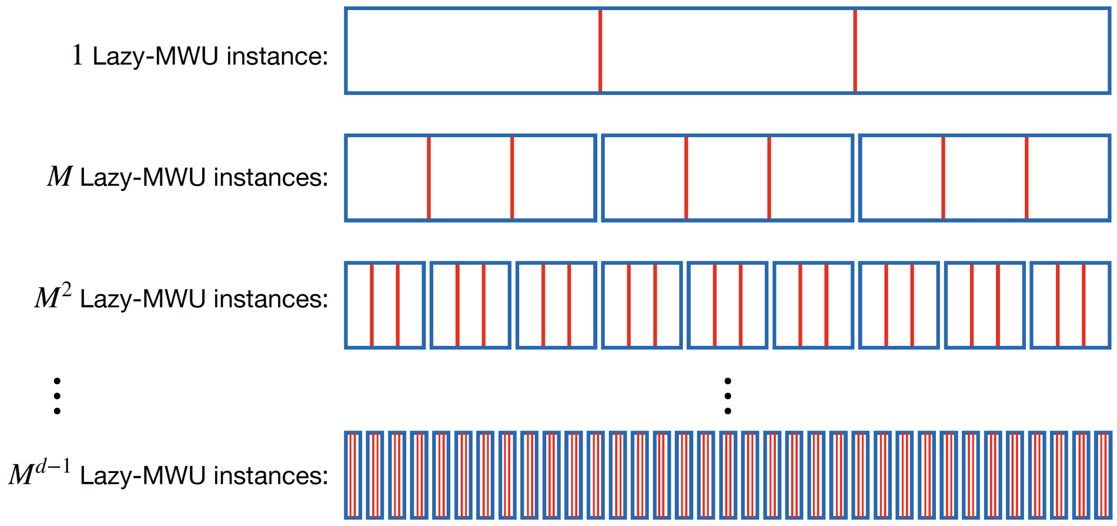


Figure 3: Tree Swap [Dagan et al., 2024]

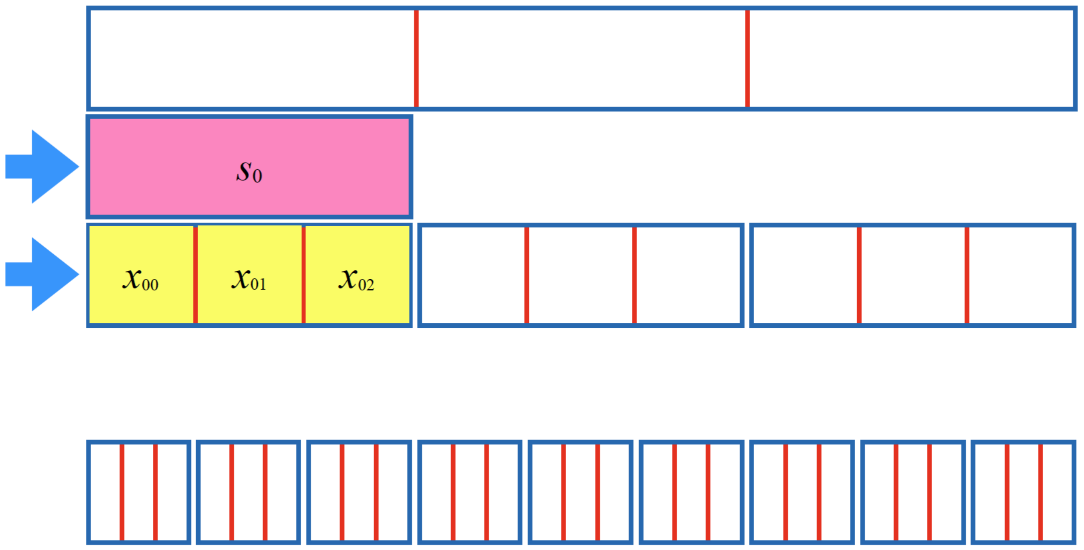


Figure 4: Regret Decomposition [Dagan et al., 2024]

The Swap-Reward corresponds to the best achievable reward if we could independently replace each local mapping $\delta^{(i)}$ by its best fixed alternative in hindsight.

We next prove that

$$\text{Swap-Reward} - \text{Reward} \le \sum_{i=1}^{d-1} \text{Ext-Regret}_{\text{layer } i} + \frac{T}{d}.$$

Step 1: From Global Swap to Layerwise Swaps. The global swap deviation δ can be interpreted as a collection of local mappings $\delta^{(1)}, \ldots, \delta^{(d)}$, one for each layer. Each mapping $\delta^{(i)}$ takes the current node or action index and outputs the next node in the directed structure. Applying all $\delta^{(i)}$ together defines a directed graph on layers; its stationary distribution describes the "swap policy" that achieves Swap-Reward.

Step 2: Layer-by-Layer Replacement Argument. Let $\Pi^{(0)}$ denote the actual Tree-Swap policy (using all online learners), and let $\Pi^{(i)}$ denote the hypothetical policy where the first i layers have been replaced by their best fixed mappings in hindsight. Then

$$\text{Swap-Reward} - \text{Reward} = \mathsf{Reward}(\Pi^{(d)}) - \mathsf{Reward}(\Pi^{(0)}) = \sum_{i=1}^{d} \left[\mathsf{Reward}(\Pi^{(i)}) - \mathsf{Reward}(\Pi^{(i-1)})\right].$$

For layers $i = 1, \ldots, d-1$: only the i-th learner changes when moving from $\Pi^{(i-1)}$ to $\Pi^{(i)}$, so by the external regret guarantee of $M^{(i)}$ with respect to its local payoffs $u_t^{(i)}$,

$$\mathsf{Reward}(\Pi^{(i)}) - \mathsf{Reward}(\Pi^{(i-1)}) \le \text{Ext-Regret}_{\text{layer } i}.$$

Summing across the first $d-1$ layers gives

$$\sum_{i=1}^{d-1} \big(\mathsf{Reward}(\Pi^{(i)}) - \mathsf{Reward}(\Pi^{(i-1)})\big) \le \sum_{i=1}^{d-1} \text{Ext-Regret}_{\text{layer } i}.$$

Step 3: The $\frac{T}{d}$ Mixing Term. The final layer replacement (from $\Pi^{(d-1)}$ to $\Pi^{(d)}$) adds an additional error because the overall action x_t is an average:

$$a_t = \frac{1}{d} \sum_{i=1}^{d} a_t^{(i)}.$$

When only the d-th layer changes, its contribution to the mixture has weight $\frac{1}{d}$. Since the per-round reward $\langle a_t, u_t \rangle$ is bounded in $[0, 1]$, this averaging causes at most $\frac{1}{d}$ deviation per round, resulting in

$$\mathsf{Reward}(\Pi^{(d)}) - \mathsf{Reward}(\Pi^{(d-1)}) \le \frac{T}{d}.$$

Combining these two parts yields

$$\text{Swap-Reward} - \text{Reward} \leq \sum_{i=1}^{d-1} \text{Ext-Regret}_{\text{layer } i} + \frac{T}{d}.$$

Hence,

$$\begin{aligned}\text{Swap-Regret}_T &\leq \text{Swap-Reward} - \text{Reward} \\ &\leq \sum_{\text{layer}=1}^{d-1} \text{Ext-Regret}_{\text{layer}} + \frac{T}{d}.\end{aligned}$$

Applying the MWU bound $\text{Ext-Regret}_{\text{layer}} = O(\sqrt{\log n/M})$ and summing across d layers yields

$$\text{Swap-Regret}_T \leq O\left(T\Big(\sqrt{\frac{\log n}{M}} + \frac{1}{d}\Big)\right).$$

Choosing

$$M \geq \frac{\log n}{\epsilon^2} \qquad \text{and} \qquad d \geq \frac{1}{\epsilon}$$

ensures $\text{Swap-Regret}_T \leq \epsilon T$, provided

$$T = M^d \geq \left(\frac{\log n}{\epsilon^2}\right)^{1/\epsilon}.$$

For infinite action spaces with finite Littlestone dimension LDim, replacing $\log n$ by LDim yields the analogous requirement:

$$T \geq \left(\frac{\mathrm{LDim}}{\epsilon^2}\right)^{1/\epsilon}.$$

Thus, Tree-Swap guarantees ϵ-swap-regret with polynomial dependence on LDim and subexponential dependence on $1/\epsilon$.

References

Avrim Blum and Yishay Mansour. From external to internal regret. *Journal of Machine Learning Research*, 8(47):1307–1324, 2007. URL http://jmlr.org/papers/v8/blum07a.html.

Yuval Dagan, Constantinos Daskalakis, Maxwell Fishelson, and Noah Golowich. From external to swap regret 2.0: An efficient reduction for large action spaces. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 1216–1222, 2024.