CS 6840 Algorithmic Game Theory Oct 28, 2024

Lecture 26: Improved Learning: No Swap Regret

Instructor: Eva Tardos *Scribe: Andre Oganesian*

1 Swap Regret

Regret quantifies how much worse an algorithm performs compared to the best fixed strategy in hindsight. Traditional regret measures focus on comparing the cumulative utility of the algorithm to that of the best single strategy. However, this approach does not account for the potential benefits of adapting our strategy over time based on past performance.

To address this limitation, we introduce the notion of *swap regret*, which allows for more flexible comparisons. Instead of only considering switching to a single fixed strategy with hindsight, swap regret evaluates the benefit of swapping each of our played strategies to possibly different strategies.

Our goal is to avoid swap regret, ensuring that we cannot significantly improve our cumulative utility by swapping our strategies after observing the entire sequence of plays. Specifically, we aim to satisfy the following condition: for all pairs of strategies $s, s' \in S$,

$$\sum_{t=1}^{T} u^t(s^t) \geq \sum_{t:s^t \neq s} u^t(s^t) + \sum_{t:s^t=s} u^t(s') - Reg,$$

where:

- T is the total number of time steps.
- s^t is the strategy played at time t.
- $u^t(s)$ is the utility obtained by playing strategy s at time t.
- Reg is the regret term.

This inequality implies that our cumulative utility $\sum_{t=1}^{T} u^t(s^t)$ is at least as great as the utility we would have obtained by replacing every occurrence of strategy s with strategy s', up to the regret term Reg. In other words, the regret quantifies how much better we could have done by swapping strategy s for s' whenever we played s.

1.1 Alternative Formulation

A more general way to express swap regret, albeit with a slightly larger regret term, is to consider all possible mappings $\pi : S \to S$. We then aim to satisfy:

$$\sum_{t=1}^{T} u^t(s^t) \geq \sum_{t=1}^{T} u^t(\pi(s^t)) - Reg.$$

In this formulation, for every strategy $s \in S$, we consider swapping it to another strategy $\pi(s)$. The mapping π need not be a permutation, meaning it can map multiple strategies to the same strategy. However, in this second version, using $\pi(s^t)$, the regret term Reg may be larger than in the previous inequality.

If we restrict π to be a constant function, i.e., $\pi(s) = s'$ for all $s \in S$, then swap regret reduces to the standard regret (sometimes also called *external regret*) with respect to strategy s'.

2 Reducing External Regret to Swap Regret

Our objective is to design an algorithm that achieves low swap regret by leveraging external no-regret algorithms. Specifically, we aim to find a reduction that transforms any external no-regret algorithm into one that minimizes swap regret.

The key idea is to create a copy of the external no-regret algorithm for each strategy $s \in S$. For each strategy, we have a separate algorithm $\mathcal{A}_s$, which is responsible for managing the potential swapping from strategy s to other strategies.

2.1 Algorithm Description

The algorithm operates as follows:

1. **Initialize Algorithms**: For each strategy $s \in S$, run an instance of a classical no-regret algorithm $\mathcal{A}_s$.
2. **Obtain Recommendations**: At each time step t, each algorithm $\mathcal{A}_s$ outputs a probability distribution over strategies, denoted by $q_s^t = (q_{s,1}^t, q_{s,2}^t, \ldots, q_{s,k}^t)$, where $k = |S|$.
3. **Compute Overall Distribution**: We solve for the probability distribution $p^t = (p_1^t, p_2^t, \ldots, p_k^t)$ that satisfies:

$$p_i^t = \sum_{s=1}^{k} p_s^t q_{s,i}^t, \quad \text{for all } i = 1, \ldots, k.$$

   In matrix notation, this equation becomes:

$$p^t = Q^t p^t,$$

   where Q^t is the $k \times k$ matrix whose s-th row is q_s^t.
4. **Sampling Strategy**: We sample a strategy s^t according to the distribution p^t and play it.
5. **Feedback to Algorithms**: After observing the utility $u^t(s^t)$, we provide feedback to each algorithm $\mathcal{A}_s$. We feed algorithm $\mathcal{A}_s$ the scaled utilities:

$$\tilde{u}_s^t(s') = p_s^t u^t(s'), \quad \text{for all } s' \in S.$$

6. **Update Algorithms**: Each algorithm $\mathcal{A}_s$ updates its internal state based on the received utilities $\tilde{u}_s^t(s')$.

2.2 Explanation of the Algorithm

The intuition behind the algorithm is that each $\mathcal{A}_s$ is responsible for evaluating whether swapping strategy s to another strategy s' would have been beneficial. By running these algorithms in parallel and combining their outputs, we aim to minimize swap regret.

The equation $p^t = Q^t p^t$ ensures consistency between the individual recommendations and the overall distribution. Importantly, a solution to $p^t = Q^t p^t$ always exists because Q^t is a stochastic matrix (each row sums to 1), so that p is simply an eigenvector of Q with an eigenvalue of 1. This also means that p^t is the stationary distribution of the Markov chain defined by Q^t.

3 Analysis of the Algorithm

3.1 No-Regret Condition for Each $\mathcal{A}_s$

Each algorithm $\mathcal{A}_s$ receives the utilities $\tilde{u}_s^t(s') = p_s^t u^t(s')$ and provides a distribution q_s^t. The expected utility for $\mathcal{A}_s$ at time t is:

$$\tilde{U}_s^t = p_s^t \sum_{i=1}^{k} q_{s,i}^t u^t(s_i).$$

The no-regret condition for $\mathcal{A}_s$ states that over T time steps, for any strategy $s' \in S$,

$$\sum_{t=1}^{T} \tilde{U}_s^t \geq \sum_{t=1}^{T} p_s^t u^t(s') - Reg_s,$$

where Reg_s is the regret of algorithm $\mathcal{A}_s$. This inequality ensures that $\mathcal{A}_s$ does not regret not consistently swapping to any particular strategy s'.

3.2 Aggregating the Utilities

Summing the expected utilities over all algorithms $\mathcal{A}_s$, we have:

$$\sum_{s=1}^{k} \sum_{t=1}^{T} \tilde{U}_s^t = \sum_{t=1}^{T} \sum_{s=1}^{k} p_s^t \sum_{i=1}^{k} q_{s,i}^t u^t(s_i).$$

Using the fact that $p_i^t = \sum_{s=1}^{k} p_s^t q_{s,i}^t$, this simplifies to:

$$\sum_{s=1}^{k} \sum_{t=1}^{T} \tilde{U}_s^t = \sum_{t=1}^{T} \sum_{i=1}^{k} u^t(s_i) p_i^t.$$

This expression represents the total expected utility of our algorithm over T time steps.

3.3 Bounding the Swap Regret

We have already defined the swap regret each individual algorithm $\mathcal{A}_s$. Now, for each algorithm $\mathcal{A}_s$, we apply the no-regret condition with $s' = \pi(s)$. Specifically, for each $s \in S$,

$$\sum_{t=1}^{T} \tilde{U}_s^t \geq \sum_{t=1}^{T} p_s^t u^t(\pi(s)) - Reg_s.$$

Summing this inequality over all $s \in S$, we obtain:

$$\sum_{s=1}^{k}\sum_{t=1}^{T} \tilde{U}_s^t \geq \sum_{s=1}^{k}\sum_{t=1}^{T} p_s^t u^t(\pi(s)) - \sum_{s=1}^{k} Reg_s.$$

Using the aggregated utilities from the previous subsection, we substitute:

$$\sum_{s=1}^{k}\sum_{t=1}^{T} \tilde{U}_s^t = \sum_{t=1}^{T}\sum_{i=1}^{k} u^t(s_i) p_i^t.$$

Therefore, the inequality becomes:

$$\sum_{t=1}^{T}\sum_{i=1}^{k} u^t(s_i) p_i^t \geq \sum_{t=1}^{T}\sum_{s=1}^{k} p_s^t u^t(\pi(s)) - \sum_{s=1}^{k} Reg_s.$$

where the RHS of the inequality is exactly the expected utility of swapping to $\pi(s)$ at each step, minus the total regret of the individual algorithms.

Thus, this demonstrates that our algorithm's cumulative utility is at least as much as what we would have obtained by applying the mapping π to swap strategies, up to the total regret of the individual algorithms.

3.4 Regret Bound

Since each $\mathcal{A}_s$ is a classical no-regret algorithm with regret $Reg_s = O(\sqrt{T \ln k})$, the total regret is:

$$Reg_{\text{total}} = \sum_{s=1}^{k} Reg_s = O(k\sqrt{T \ln k}).$$

Thus, the swap regret of our algorithm is bounded by $O(k\sqrt{T \ln k})$, which grows sublinearly with T.

3.5 Improving the Regret Bound

Recall that each algorithm $\mathcal{A}_s$ is a classical no-regret algorithm over the k strategies. Looking back to our analysis of the Hedge algorithm, the regret for $\mathcal{A}_s$ over the T time steps is bounded by

$$\sum_{t=1}^{T} \tilde{U}_s^t \geq (1-\varepsilon) \sum_{t=1}^{T} \tilde{u}_s^t(s') - \frac{\ln k}{\varepsilon}$$

where $\varepsilon > 0$ is a parameter of the algorithm, and $\tilde{U}_s^t$ is the expected utility received by $\mathcal{A}_s$ at time t

Using this bound for each $\mathcal{A}_s$ and mapping $\pi : S \to S$ we have:

$$\sum_{t=1}^{T} \tilde{U}_s^t \geq (1-\varepsilon) \sum_{t=1}^{T} \tilde{u}_s^t(\pi(s)) - \frac{\ln k}{\varepsilon}$$

where summing over all s we get, using our previously calculated expressions,

$$\sum_{t=1}^{T} \sum_{s=1}^{k} u^t(s_i) p_i^t \geq (1-\varepsilon) \sum_{t=1}^{T} \sum_{s=1}^{k} p_s^t u^t(\pi(s)) - \frac{k \ln k}{\varepsilon}$$

Since our goal is to bound the difference between the cumulative utility of our algorithm and that of the best swap strategy in hindsight, the swap comparators utility is given by

$$U_{swap} = \sum_{t=1}^{T} \sum_{s=1}^{k} p_s^t u^t(\pi(s))$$

Plugging this in and rearranging we get

$$Reg_{total} = U_{swap} - \sum_{t=1}^{T} \sum_{s=1}^{k} u^t(s_i) p_i^t \leq \varepsilon U_{swap} + \frac{k \ln k}{\varepsilon}$$

We can then choose the clear bound $U_{swap} \leq T$ to get

$$Reg_{total} \leq \varepsilon T + \frac{k \ln k}{\varepsilon}$$

Solving this involves basic calculus, by differentiating to get the optimal epsilon as $\varepsilon^* = \sqrt{\frac{k \ln k}{T}}$ which gives us the final bound

$$Reg_{total} \leq 2\sqrt{kT \ln k}$$

Thus, our algorithm also achieves a swap regret with a sublinear dependence on k, of the order $O(\sqrt{kT \ln k})$.

4 Extension to Partial Information

In the previous analysis, we assumed a full-information setting where each algorithm $\mathcal{A}_s$ knows the utilities $u^t(s')$ for all strategies $s' \in S$. To adapt our algorithm to the partial-information setting, we modify the feedback provided to the algorithms $\mathcal{A}_s$.

4.1 Modified Feedback

Specifically, at each time step t, the estimated utility for strategy s' is defined as:

$$\hat{u}^t(s') = \begin{cases} 0, & \text{if } s' \neq s^t, \\ \dfrac{u^t(s')}{p^t(s')}, & \text{if } s' = s^t. \end{cases}$$

Each algorithm $\mathcal{A}_s$ receives a scaled version of this estimated utility, where the utility provided to $\mathcal{A}_s$ is:

$$\tilde{u}_s^t(s') = p^t(s) \cdot \hat{u}^t(s').$$

Notice that when $s = s'$, the scaling factor $p^t(s)$ cancels out.

Thus, each algorithm $\mathcal{A}_s$ still operates as if in the full-information setting, while simply receiving these scaled utilities.