CS6550 Matroid Secretary Project

Mihir Kadiwala, Meena Nagarajan

April 2024

Abstract

Recent advancements in the Matroid Secretary Problem have highlighted the complexity and potential of competitive online algorithms. Notably, the development of a simplified $O(\log\log(\text{rank}))$-competitive algorithm provides a significant leap over prior methods that offered $O(\log(\text{rank}))$ competitive ratios. This paper aims to dissect the effectiveness of the $O(\log\log(\text{rank}))$ approach and explore potential avenues for further enhancing the competitive ratio. Through a detailed analysis, we evaluate the underlying mechanisms of the algorithm, focusing on its simplification and the reduction in the hidden constants involved in its competitive ratio. Additionally, we propose novel approaches that may potentially improve the competitive ratio, contributing to more efficient solutions for the Matroid Secretary Problem.

Overview

The Classical Secretary Problem

The Secretary Problem, a well-studied model in optimal stopping theory, involves sequentially interviewing candidates and deciding immediately whether to hire a candidate or continue with the interviews. The challenge is to maximize the probability of selecting the best candidate among a known total number, with each candidate appearing in a random order. A classical solution to this problem allows for the selection of the best candidate with a probability of at least $\frac{1}{e}$, a result that is known to be asymptotically optimal.

Extensions to the Matroid Secretary Problem

The Matroid Secretary Problem extends the classical framework to scenarios where multiple selections are permissible under certain constraints defined by a matroid structure. In this context, each element (akin to a secretary)

is associated with a weight, and the goal is to maximize the total weight of the selected set while adhering to the independent set constraints of the matroid. This problem formulation encapsulates a wide range of practical applications, from resource allocation to auction design, where decisions must be made online and optimally.

Introduction to Competitive Algorithms

Competitive analysis is a critical tool for evaluating online algorithms, where the performance of an online algorithm is compared against an optimal offline algorithm that has complete future knowledge. The focus of our study, the $O(\log\log(\text{rank}))$-competitive algorithm, represents a significant theoretical advancement in this area, simplifying previous approaches while maintaining desirable competitive ratios. The algorithm leverages a bucketing technique and randomization to achieve its performance, reducing the complexity involved in earlier methods that required intricate bucketing and careful analysis.

Simplification and Analysis

The $O(\log\log(\text{rank}))$-competitive algorithm simplifies the execution and analysis by employing a distribution over simple types of matroid secretary algorithms, which are more straightforward to analyze. Our paper delves into the mathematical underpinnings of this simplification, elucidating how the algorithm minimizes the hidden constants in its competitive ratio, a factor crucial for practical implementations.

Proposing Improvements

Building on the foundational work of the $O(\log\log(\text{rank}))$-competitive algorithm, our research explores theoretical enhancements that could further improve the competitive ratio. By analyzing the algorithm's performance under varying conditions and matroid structures, we identify potential modifications and novel techniques that could yield superior results. These advancements could pave the way for more efficient algorithms capable of handling a broader array of problems within the matroid framework.

This comprehensive analysis not only solidifies understanding of current competitive algorithms but also sets the stage for future breakthroughs in the field of online optimization and decision-making.

1 The Log(rank) Approach

Once we have an understanding of the problem, it is simple to see that there exists an approach that is able to get us feasible solution without much involvement. We have a problem that comes to us like this: The Matroid Secretary Problem, which consists of the matroid M= $(N, \mathcal{I})$. the naive approach, as we have seen for the original problem is quite simple and is able to find a $O\hat{P}T$ and choose the best element that satisfies. Now, this approach does not translate entirely to the matroid problem because the structure of a matroid is fundamentally different than that of the regular secretary and a weight, however, we can still use a simple algorithm to find a feasible solution with a competitive ratio that is equal to the log of the rank of the matroid. Again, the rank of the matroid refers to the maximum size of an independent set within the matroid, we will also call the optimum solution V^*. Now, if we find the range of values, such that we have V_{min} and V_{max}, then we can divide the set into log n buckets. Now, this is obviously contingent on the fact that we are able to obtain this information. We can do this by either being passed in these values or gaining this information in a pseudo random manner by choosing each element with .5 probability and calculating V_{min} and V_{max}. Then, you can break the elements into different size buckets, such that each bucket contains the elements that have weight in the range from $[V_{min} \cdot 2^i, V_{min} \cdot 2^{i+1}]$. Then, we can randomly pick from one of these buckets to greedily obtain a feasible solution. Here is the algorithm.

Bucket Selection Algorithm

1: Obtain $V_{\min}$ and $V_{\max}$ from the elements by looking at the first $n/2$ elements if not given
2: Place the first $[V_{\min} \cdot 2^i, V_{\min} \cdot 2^{i+2}]$ elements into a bucket
3: Increment i until $\log n$ buckets have been created
4: Choose a bucket with equal probability: $\frac{1}{\log n}$
5: Greedily add elements from bucket into solution set S such that $S \cup \{e\}$ is independent, and $e \notin S$ =0

Theorem 1 *Bucket Selection Algorithm obtains a competitive ratio equivalent to at least* log *(rank)*

As we generate S^*, within a factor of two, we obtain an optimum solution. We can use a proof from class. We have buckets $B_0, B_1, \ldots, B_{\log 2}$, and each bucket B_i contains at least $\frac{V^*}{2}$ because let B_i be the set of edges with

values exactly $\frac{v_{\max}}{2^i}$. This means that the elements in S^* contained within the buckets $B_0, B_1, \ldots, B_{\log_2 n}$ together have a total value of at least $V^*/2$.

Suppose this is not the case. Then, at least $V^*/2$ of the value in S^* would come from elements with values less than $\frac{v_{\max}}{2^{\log_2 n}}$. However S^* contains at most $n-1$ elements. Even if each of these elements contributes $\frac{v_{\max}}{n} < \frac{V^*}{4n}$, we cannot obtain a total value of $V^*/2$ from them, leading to a contradiction. Thus, we conclude that

$$\sum_{i=0}^{\log_2 n} \left(\frac{v_{\max}}{2^i}\right) \cdot |S^* \cap B_i| \geq \frac{V^*}{2}.$$

Now, for any of the B buckets, we select an item with a value $\frac{v_{max}}{2^i}$. Next, if we for any choice in bucket, we have an equal probability of $1/\log n$, so using our algorithm, we get a value $\Omega(V^*/\log n)$. Thus, we have a log (rank) competitive algorithm.

2 The Bucketing Algorithm

Before we get into the log log (rank) algorithm, it is critical that before we first introduce the algorithm for the log log rank approach, we show the bucketing algorithm. The bucketing algorithm essentially buckets the bucket and selects a solution set that is ale to improve the competitive ratio. The manner in which the bucketing algorithm works ensures that the solution is feasible and also efficient. Here are the steps of the algorithm from the Feldman paper [2].

The Bucketing Algorithm

Let S be a set containing every element with probability $1/2$.
Let $H = H_{odd}$ with probability $1/2$, and $H = H_{even}$ otherwise.
Let $T_i \leftarrow \emptyset$ for $i \in H$.
for every element $e \in N \setminus S$ revealed **do**
  Let i be the index of the bucket containing e (i.e., $e \in B_i$).
  if $i \in H$ and $e \in N_i$ and $e + T_i \in \mathcal{I}$ **then**
    Add e to T_i.
  end if
end for
return $T = \bigcup_{i \in H} T_i$. $=0$

We know that this algorithm ensures feasibility because it checks to make sure that in selecting an element that is not spanned by heavier elements

that are already in S and that the element is not spanned by elements in S of similar weight. This ensures that we are, in fact, choosing the best weights to add to our set S.

Corollary 1 *The Bucketing Algorithm returns an independent set.*

For every $j \in H$, let $F_j = \bigcup_{i \in H, i \geq j} I_i$. We are looking at element j in H, and we are concerned with the union of elements that are greater than i. We can show by induction that $F_j \in I$ for all $j \in H$. If we choose j to be the largest possible index, such that the conditions still hold true, then $F_j \in I$ when j is the largest index in H, then F_j would be equivalent to I_j and both sets are not empty.

Now, if j isn't the largest, then, since we are only concerned with either H_{even} or H_{odd}, we have F_{j+2} if we focus on the even, and we know that $F_j \in I$. Now, let's think about what I represents, I is a part of the ground set N, so $I_i \subseteq N_i$, and the bucket is of I_i, the span$(S \cap B_{i-1})$ is also a subset of both I_i and N_i and the fact that $I_j \neq \emptyset$, we can create a sequence of matroids, that has been used in previous matroid secretary problems to prove feasibility and see that the sets form a chain, and this chain would have each I_i be independent in the original matroid for H_{even}, which means that the elements chosen would be independent. More formally, we have $F_{j+2} \subseteq \text{span}(S \cap B_{\geq j+1})$, so $r(F_{j+2}) = |F_{j+2}|$ and this means that $F_j = F_{j+2} \cup I_j \in I$. We can also easily see that $M_i^* = (M/A_2, A_4, \ldots A_{h-1})$, such that $A_2 \supseteq A_4 \supseteq \ldots A_{h-1}$. Thus, we have a valid and feasible solution to the matroid secretary problem that is created through the bucketing algorithm if it is fed the appropriate buckets.

3 The Log(Log(rank)) Approach

The $O(\log \log(\text{rank}))$-competitive algorithm for the Matroid Secretary Problem introduces an innovative mechanism of bucketing that plays a crucial role in its efficacy and efficiency. The essence of this approach lies in how the set of elements (or "buckets") is selected and processed. Initially, the algorithm divides the entire set of elements into groups, termed as buckets, where each bucket contains an equal number of weight classes. These classes are determined by a parameter τ, which is chosen randomly and uniformly from the range $[0, \lceil \log_2(h+1) \rceil]$, where h is the maximum number of weight classes conceivable based on the problem's constraints.

The chosen value of τ effectively determines the granularity of the bucketing: a smaller τ results in fewer weight classes per bucket, leading to more

buckets, while a larger τ yields fewer, larger buckets. This randomization in the selection and size of the buckets introduces a level of unpredictability that is crucial for dealing with the adversarial nature of online algorithms. Each bucket thus encompasses a subset of elements whose collective weights fall within a specific range.

Upon establishing this structured partitioning, the bucketing-based algorithm is executed. The algorithm's strategy involves observing and selectively processing these buckets to make real-time decisions on which elements to select, aiming to maximize the total weight of the chosen subset. This process leverages the inherent properties of matroids—particularly their independent set structures—to ensure that the selections are optimal or near-optimal with respect to the algorithm's constraints and capabilities.

The competitive ratio of $O(\log\log(\text{rank}))$ emerges from the logarithmic reduction in complexity enabled by the bucketing mechanism. Instead of dealing with each element individually, the algorithm handles manageable clusters of elements, allowing for a rapid convergence towards an optimal solution by dramatically narrowing down the potential choices at each step. This methodology not only simplifies the computational overhead but also sharpens the algorithm's ability to approximate the best possible outcome under uncertainty, harnessing the power of structured randomness and matroid theory in a sophisticated, yet elegantly simple framework. Here is the algorithm from the paper [2].

Feldman Matroid Secretary Algorithm

Let S be a set containing every element with probability $1/2$.

Let $H = \text{Hodd}$ with probability $1/2$, and $H = \text{Heven}$ otherwise.

Let $T_i \leftarrow \emptyset$ for $i \in H$.

For every element $e \in N \setminus S$ revealed do

Let i be the index of the bucket containing e (i.e., $e \in B_i$).

If $i \in H$ and $e \in N_i$ and $e + T_i \in I_i$ then

Add e to T_i.

Return $T = \bigcup_{i \in H} T_i$.

The algorithm organizes elements into weight classes C_i and applies a random bucketing strategy parameterized by τ, affecting selection efficiency.

Lemma 4.1 ($\tau = 0$)

Statement: Each class C_i is a bucket; the expected intersection size satisfies:

$$E[|T \cap C_i|] \geq \frac{1}{4} \sum_{e \in C_i \cap \mathrm{OPT}} p_{e,i}$$

Proof: With $\tau = 0$, classes are independent. Each element's fixed probability $p_{e,i}$ of being chosen reflects directly in the expected number of selections from C_i.

Lemma 4.2 ($\tau \geq 1$)

Statement: For elements e in C_i under $\tau \geq 1$, the selection probability is:

$$\Pr[e \in T \mid \tau \geq 1] \geq \frac{1 - p_{e,i}}{8 \cdot \lceil \log_2(h+1) \rceil}$$

Proof: Complex bucketing leads to increased competition within buckets. The logarithmic denominator accounts for the bucketing complexity.

Corollary 4.3

Combining results from both lemmas, the expected weight from each class is:

$$E[|T \cap C_i|] \geq \frac{|C_i \cap \mathrm{OPT}|}{8(\lceil \log_2(h+1) \rceil + 1)}$$

Theorem 4.4

Statement: The competitive ratio is $16(\lceil \log_2(h+1) \rceil + 1)$, indicating $O(\log \log \rho)$-competitiveness.

Proof: Summing the probabilities across all classes, adjusted for the highest weight division level h, confirms near-optimal performance against adversarial actions and random arrivals.

4 Our Approach 1: Bucketing Buckets

In order to better improve the algorithm, we need to first think about the ways in which we can improve parts of the log log (rank) approach. This requires careful analysis of its most useful parts. Clearly, the main difference between the original approach and the more competitive approach is that we bucket the buckets and find a feasible solution that is found from the buckets. From this grain, we thought that we could apply this one more time. If we could simply bucket the buckets again, then maybe we could get a a ratio that is equivalent to triple log of the rank. Essentially, from the Feldman Matroid Secretary Algorithm above, we would change the steps related to initializing S and the members holding H. This set consists of the elements that will be passed into the bucketing algorithm, and our idea is that if we can somehow optimize the elements that are placed into this set, we may be able to improve the competitive ratio using the same bucketing algorithm that is used in the original approach.

One way we could change the elements of the set that are sent to the bucketing algorithm is to create a bucket based on the buckets that we have already created. So far in the algorithm, we have weight classes and buckets that hold those weight classes. In our approach, we propose to perform a uniform random shift such that that bucket B_i is shifted by picking uniformly random variable $K \in [0, 1]$ with equal probability. This allows us to bucket the buckets again, since we can perform the same algorithm shown above, but we now have $\log_2 \log_2 \log_2(\text{rank})$ weight holders since we have the weight classes, the buckets, and now the buckets for the buckets, we hope to allow this change to give us an even more better competitive ratio.

However, as we look into the algorithm we have created, it is integral to point out that we still rely on the same bucketing algorithm to feasibly select a solution. If we look back on our proof for Corollary 1, we must maintain that the same conditions apply. Clearly, we can see that the proof is reliant on H_{even} and H_{odd} being independent sets, but contain similarly weighted elements. We can see that this is also the case in our approach. Next, we need a set or collection that the bucketing algorithm is able feasibly return a solution of, and quickly, we see that the bucket algorithm essentially combines the collection and chooses a feasible greedy solution based on the buckets. The key point in this is that there are similarly weighted weight classes that are combined

together. In our proposed approach, though we would actually hinder the performance of the algorithm because the bucketing algorithm since the algorithm checks to make sure the span of the elements is not covered by other elements and if we combine buckets, then it is more likely that the elements in an original bucket would be spanned by items in another bucket that we have now just combined. For this reason, our bucketing the buckets algorithm fails and we understand that we have to move on to a different kind of approach. This failure shows the fact that the Feldman Matroid Secretary Algorithm relies heavily on the independent sets that contain similar weight classes for feasibility and optimality.

Instead, we could perhaps change the way in which we choose the elements in the buckets.

5 Our Approach 2: Biasing Tau

Initial Estimation of τ and Continuous Data Monitoring

The initial estimation of τ sets the preliminary scale for the bucketing process, affecting how elements are grouped and processed. This initial value may be based on historical data, expert judgment, or a predefined scale according to the problem's characteristics. Once operational, continuous monitoring of data is crucial to capture evolving trends and distributions of element values. This ongoing monitoring acts as a feedback source for dynamically adjusting τ, ensuring the algorithm's effectiveness under changing conditions.

Dynamic Adjustment Rules

The adjustment of τ is dynamically governed by specific rules that respond to observed data during the algorithm's execution:

- **Increase τ:** Triggered if the maximum value observed in a batch significantly exceeds the current setting of τ, or if high-value elements are consistently missed because their values surpass τ. Raising τ allows the algorithm to accommodate higher value elements in subsequent batches, potentially increasing overall performance.
- **Decrease τ:** If data indicates that few elements exceed τ, suggesting it may be set too high, reducing τ makes the threshold

more sensitive to smaller yet significant values, enhancing the algorithm's granularity and potential for advantageous selections.

Feedback Mechanism Implementation

Implementing a robust feedback mechanism is vital for the dynamic adjustment of τ. This mechanism should analyze the performance impact of current settings and autonomously decide on adjustments. It often involves statistical tests or performance metrics that objectively evaluate how well the algorithm performs with the current settings of τ.

Adjustment Frequency and Learning Rate

The frequency of adjustments and the rate at which changes to τ are applied must be carefully managed to balance responsiveness with stability. Frequent adjustments can allow the algorithm to quickly adapt to new data but may also lead to performance volatility. Conversely, a slower adjustment rate can result in more stable but less responsive performance. The learning rate determines how significantly τ is adjusted each time, with a higher rate leading to more substantial changes.

Challenges

Several challenges arise with the dynamic biasing of τ, including:

- **Risk of Overfitting:** Continuously adjusting τ based on immediate past observations can lead the algorithm to fit too closely to random fluctuations or outliers in the data, potentially degrading overall performance.
- **Implementation Complexity:** Incorporating a dynamic adjustment mechanism for τ increases the algorithm's complexity, which can pose challenges in terms of computational efficiency and the robustness of the implementation.
- **Stability and Convergence Issues:** Ensuring that the adjustments to τ lead to stable and convergent behavior over time is critical. Without proper controls and limits, frequent changes to τ could cause erratic behavior and divergence from optimal performance.

Overall, dynamically biasing τ offers a promising approach to enhance the adaptability and efficiency of the Matroid Secretary Problem algorithm but requires careful implementation and monitoring to overcome associated challenges.

References

[1] S. Chakraborty and O. Lachish, *Improved Competitive Ratio for the Matroid Secretary Problem*, pp. 1702–1712. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611973099.135

[2] M. Feldman, O. Svensson, and R. Zenklusen, "A simple $o(\log \log(\text{rank}))$-competitive algorithm for the matroid secretary problem," 2014.