6.854 Fall 2012
Advanced Algorithms

Matroid Secretary Problems

Gautam Kamath
Instructor: David Karger g@csail.mit.edu

1 Introduction

In 1963, Dynkin introduced the secretary problem [6]. In this problem, an algorithm is presented with n positive values, one by one. After each value, the algorithm must either accept or reject the value, where all decisions are final. The algorithm can only pick one value, and the goal is to pick the maximum value in the sequence. The name for this problem arises from a situation where n candidates are interviewed for a secretary position, and the interviewer wants to hire the best candidate. Other names for this problem include the best choice problem, the marriage problem, and the MIT dating problem.

At first glance, it may seem impossible to hope to pick the greatest value - since the values are arbitrary, couldn't we always have a bigger element after we accept one? It turns out that one can pick the best value with probability approaching $\frac{1}{e}$. The essential idea is to learn properties of the sequence based on the first several values, and then using what we learned in order to make a decision on the remainder of the sequence.

This concept is very general, and can be applied on extensions of the original problem. Indeed, a number of extensions have studied - the most natural extension is the multiple-choice secretary problem [7], where the algorithm may choose k values, rather than just one. Other extensions include the submodular secretary problem [3] and the knapsack secretary problem [1]. In this survey, we concern ourselves with the matroid secretary problem. This is another multiple-choice secretary problem, but the elements selected must be an independent set in some matroid. This problem was introduced by Babaioff et. al in 2007 [2], where they provide a $O(\log k)$ approximation, where k is the rank of the matroid. While there have been many results on particular matroid structures [5] or in slightly modified models [8], it has proven challenging to improve on this approximation. In 2012, a new result by Chakraborty and Lachish improves the approximation factor to $O(\sqrt{\log k})$, though it is conjectured that there exists a constant factor approximation for general matroid settings.

Surprisingly, matroid secretary problems also have applications in mechanism design. We can consider an auction in which agents arrive one by one. Agents have a set of outcomes that would satisfy them - they gain a value of v_i if they are satisfied, and 0 otherwise. Once an agent arrives, he announces his value. At that point, the mechanism must either commit to satisfying the agent or not. After all agents have arrived, the mechanism must output an outcome which fulfills all prior commitments. We'd like an auction to be truthful (an agent has no incentive to lie about his value v_i) and to maximize the social welfare (maximizing $\sum_{i \in S} v_i$, where S is the set of satisfied agents). It turns out that an approximate solution to a matroid secretary problem can be used to design a truthful mechanism that approximately maximizes the social welfare [1].

2 Preliminaries

2.1 Matroids

A matroid is a combinatorial structure, which can be thought of as a generalization of a forest on a graph. We provide the formal definition first, followed by several examples to illustrate the properties of a matroid.

Definition 1 *A* **matroid** *is a pair* $(\mathcal{U}, \mathcal{I})$*, where* $\mathcal{U}$ *(the* **ground set***) is a finite set, and* $\mathcal{I}$ *(the* **independent sets***) is a set of subsets of* $\mathcal{U}$*.* $\mathcal{I}$ *has the following properties:*

1. $\emptyset \in \mathcal{I}$ *(The empty set is independent.)*
2. $\forall X \subset Y \subset E, Y \in \mathcal{I} \rightarrow X \in \mathcal{I}$ *(Every subset of an independent set is independent. This is the* **hereditary property***.)*
3. $X, Y \in \mathcal{I} \wedge |X| < |Y| \rightarrow \exists y \in Y : \{y\} \cup X \in \mathcal{I}$ *(Given two independent sets of different sizes, there exists an element in the larger set which can be added to the smaller set without breaking independence. This is the* **augmentation property***.)*

We will find it convenient to refer to the rank and a basis of a matroid.

Definition 2 *The* **rank** *of a matroid is the maximum cardinality of any independent set.*

Definition 3 *A* **basis** *of matroid is an independent set with cardinality equal to the rank of the matroid.*

Matroids arise naturally in a number of combinatorial settings, as shown in the following examples. As an exercise to gain intuition about matroids, verify that they follow the properties stated in the definition.

- *Uniform matroid*: A uniform matroid of rank k has a ground set $\mathcal{U}$, where a subset X of $\mathcal{U}$ is independent iff $|X| \leq k$.
- *Partition matroid*: Suppose we have some collection of k disjoint sets S_i. A partition matroid has a ground set $\mathcal{U} = \cup_i S_i$, where a subset X of $\mathcal{U}$ is independent iff $|X \cap S_i| \leq 1 \ \forall i$. The rank of this matroid is k.
- *Graphic matroid*: Suppose we have some graph $G = (V, E)$. A graphic matroid has a ground set $\mathcal{U} = E$, where a subset X of $\mathcal{U}$ is independent iff X does not contain a cycle in G. If the graph is connected, the rank of this matroid is $|V| - 1$. Otherwise, the rank is $|V| - c$, where c is the number of connected components. Note that a basis of this matroid corresponds to a spanning forest of G.

- *Gammoid*: Suppose we have a graph $G = (V, E)$, with two (not necessarily disjoint) subsets of V: S and T. A gammoid has a ground set $\mathcal{U} = T$, where a subset X of $\mathcal{U}$ is independent iff there exists $|X|$ vertex disjoint paths from S onto X. A gammoid is strict if $\mathcal{U} = T = V$.

- *Transversal matroid*: Suppose we have a bipartite graph $G = (U, V, E)$. A transversal matroid has a ground set $\mathcal{U} = U$, where a subset X of $\mathcal{U}$ is independent iff there exists a matching of size $|X|$ between the elements of X and V. Note that this is a special case of a gammoid.

- *Vector matroid*: Suppose we have a vector space V. A vector matroid has a ground set $\mathcal{U} = S$, where S is some finite subset of V. A subset X of $\mathcal{U}$ is independent iff X is linearly independent in V.

A natural extension to a matroid is a weighted matroid, in which we assign weights to the elements.

Definition 4 *A* **weighted matroid** *is a pair $(w, \mathcal{M})$, where $\mathcal{M} = (\mathcal{U}, \mathcal{I})$ is a matroid and $w : \mathcal{U} \to \mathbb{R}^+$ is a function which assigns a positive weight to each element.*

For ease of notation, we let $w(X) = \sum_{x \in X} w(x)$.

There exists a very simple greedy algorithm which computes a maximum weight basis of a matroid. This algorithm is a generalization of Kruskal's algorithm on graphic matroids.

Algorithm 1 The greedy algorithm on matroids

```
Sort the elements of U in decreasing order of weight
B = ∅
for x ∈ U do
    if B ∪ {x} ∈ I then
        B ← B ∪ {x}
    end if
end for
return B
```

2.2 The Secretary Problem

2.2.1 The Classical Secretary Problem

While there are many variants of the classical secretary problem, the most common variant is as follows: An algorithm is given n positive values sequentially, where the value of n is known beforehand. The specific values given are arbitrary, but they are provided to the algorithm in a uniformly random order. After being given each value, the algorithm can either choose to

accept or reject the value, where all decisions are irreversible. The algorithm may only accept one value, and the goal is to maximize the probability of choosing the greatest value.

It turns out that a very simple algorithm yields a constant probability of choosing the greatest value. Consider the following algorithm:

Algorithm 2 Constant probability secretary problem algorithm

```
threshold = 0
for i = 1, ..., n/2 do
    Reject A[i]
    threshold = max(threshold, A[i])
end for
for i = n/2, ..., n do
    if A[i] > threshold then
        Accept A[i]
    else
        Reject A[i]
    end if
end for
```

We reject the first half of the values, compute a threshold equal to the maximum value in the first half, and then accept the first value greater than the threshold. Observe that, if the second largest element is in the first half, and the largest element is in the second half, then this algorithm will output the maximum element. This happens with probability $\frac{n}{2n} \cdot \frac{n}{2n-1} > \frac{1}{4}$, giving us at least 25% chance of selecting the maximum element. This can be improved to a fraction $\frac{1}{e} \approx 36.8\%$ using a very similar algorithm with a neater analysis - instead of rejecting the first $\frac{n}{2}$, we reject only the first $\frac{n}{e}$ elements before setting the threshold.

2.2.2 Matroid Secretary Problems

It turns out the most natural generalization of the secretary problem is to modify the nature of the objective function. While before, we were trying to maximize the probability of picking the best value, we now attempt to maximize our expected value. The description of the matroid secretary problem is as follows: There is a weighted matroid $(w, \mathcal{M})$. Before the process begins, an algorithm is provided with the matroid $\mathcal{M}$, but not the weights. The algorithm maintains an independent set S, which is initially $\emptyset$. The weights of the elements of $\mathcal{U}$ are provided to the algorithm in a uniformly random order. Immediately after observing the weight of an element x, if $S \cup \{x\} \in \mathcal{I}$, then the algorithm may choose to accept it (thereby adding it to the set S) or reject it. The goal of the algorithm is to maximize the value of $w(S)$.

In the models studied in this survey, weights are assigned to elements arbitrarily (the zero information model). Since it appears to be quite difficult to obtain a constant factor approximation in this model, there are some results on relaxations of this model [8].

Finally, there exists a more difficult restriction of the problem, which we call the **strict-comparison** model. In this model, the algorithm only learns the ranking of an element relative to the previous elements, and not the precise weight of an element.

The performance of an algorithm is typically measured by taking the ratio of $w(OPT)$ and $E[w(S)]$, where OPT is a maximum weight basis of $\mathcal{M}$. We say an algorithm is ***c*-competitive** if $w(OPT) \leq cE[w(S)]$.

2.3 Connections to Mechanism Design

In the introduction, we described a connection between the matroid secretary problem and online auction mechanism design. Now that we have some matroid terminology at our disposal, we can be more concrete in our definition of the outcomes which satisfy an agent. We can define a one-to-one correspondence between agents and elements in the ground set of a matroid, and say an agent is satisfied when the element is in our chosen independent set of the matroid. Some auction settings are described below - for more details, see [2].

- *Selling k identical items*: There are n agents and k identical items for sale. An agent will be satisfied if he gets any item, and dissatisfied otherwise. This corresponds to the uniform matroid of rank k.
- *Unit-Demand Domain*: There are n agents and m non-identical items for sale. Each agent wants to get exactly one item from some subset of items which would satisfy him. This corresponds to a transversal matroid.
- *Gammoids*: In a gammoid, we are trying to connect sources to sinks in edge-disjoint paths. This can be seen as a sort of routing auction, with potential applications in the routing of internet traffic.

While there are specific algorithms giving constant factor approximations for several of these settings, we have yet to develop a constant factor approximation for the general matroid secretary problem. Having a unified framework for all matroid secretary problems would greatly enhance our understanding of the problem, and pave the way for more advanced work.

3 Algorithms

3.1 A $O(1)$ Approximation on Uniform Matroids [7]

This result predates work on matroid secretary problems, and was originally phrased as a multiple-choice secretary problem. First, we note that the result is phrased differently. The result says that we can obtain a value of at least $\left(1 - \frac{5}{\sqrt{k}}\right) v$, where v is the optimal solution. Converting this to our terminology, it says that the algorithm is $\frac{\sqrt{k}}{\sqrt{k}-5}$ competitive. This can be converted to a constant factor approximation by using the given algorithm for sufficiently

large k, and using the classical secretary algorithm for all k smaller than this value. For large k (say, larger than 36), we can see that the approximation factor is < 6. For small k, the classical algorithm can be analyzed in this case as follows: The value of the maximum element is at least $\frac{v}{k}$, and we pick it with probability $\frac{1}{e}$, so our expected payoff is at least $\frac{v}{ke}$. For k smaller than 36, this gives a $36e$ approximation, providing us with a $O(1)$ approximation for the problem.

The result in the paper is actually stronger than this statement - it shows that our approximation can get arbitrarily close to 1 for large enough k, and gives a rate at which our approximation improves. It turns out this is asymptotically optimal - any algorithm can only obtain a $1 - \Omega\left(\frac{1}{\sqrt{k}}\right)$ fraction of the optimal value.

At a high level, the algorithm can be approximated as the following: We break the sequence of n elements into several segments which roughly double in length each time, where the first segment is of length approximately $\frac{n}{k}$. There's a tradeoff here, similar to the classical problem, in which we choose the optimal point to stop learning and start trying to make a decision. We would like to defer the decisions as long as possible, so that we know the maximum amount about the distribution. However, if we wait too long, we will miss all the best elements. It turns out an appropriate way to choose the items is to pick one element from the first segment, and double the number of elements we choose in each subsequent segment. We use information from all the previous segments to make decisions - specifically, if we have seen a fraction p of all elements, we will take any element that would fall in the top pk fraction of the past elements, since under a random permutation, this sample would be a good representation of elements which fall in the top k elements. In particular, note that when we reach the end of the process, this threshold value would be exactly the top k elements in the sequence.

More precisely, the algorithm is as follows: If $k = 1$, then apply Algorithm 2. Otherwise, draw m from $Binomial\left(n, \frac{1}{2}\right)$. Recursively select $l = \lfloor \frac{k}{2} \rfloor$ elements from the first m values. Set a threshold t, equal to the lth largest value from the first m values. After the first m values, select every value which is greater than t, up to a maximum of k elements.

As this is a recursive algorithm, it is natural to analyze it by induction on k. First, we adjust the weights of the elements. If an element is in the optimal solution T (i.e., it is one of the largest k values), it maintains its original weight. Otherwise, it is assigned a weight of 0.

One key difference between this algorithm and the classical algorithm is the method in which we partition the sequence. Before, we deterministically split the sequence at the halfway point. Now, we randomly split the sequence at a position determined by a sample from $Binomial\left(n, \frac{1}{2}\right)$. Let S be the set of all elements, and Y and Z refer to the elements before and after the split, respectively. One can verify that Y will be a uniformly random subset of all 2^n subsets of S.

The analysis is broken into two halves - we argue about the value added to our solution by Y, and then by Z. If we have r items from T in Y, then in expectation, the total value of Y is $\frac{r}{k}v$. We further modify the weights for the elements in Y - if it is in the top l elements of Y, it keeps its value, otherwise, it gets the value of 0. This makes the total expected value of Y (given r) be $\frac{\min(r,l)}{k}v$. To make this unconditional on r, we take the weighted sum over the possible values of r, getting a value of at least $\left(1 - \frac{1}{2\sqrt{k}}\right)\frac{v}{2}$. This allows us to use our induction hypothesis, giving a total expected value obtained from Y of at least $\left(1 - \frac{5}{\sqrt{\frac{k}{2}}}\right)\left(1 - \frac{1}{2\sqrt{k}}\right)\frac{v}{2}$.

Algorithm 3 Multiple-choice secretary problem

```
if k = 1 then
    Run Algorithm 2, the classic secretary algorithm
else
    m ← Binom(n, 1/2)
    l ← ⌊k/2⌋
    Recursively run algorithm on first m items in sequence, pick up to l values
    B ← sort(A[1], ..., A[m])
    threshold ← B[l]
    for i = m + 1, ..., n do
        if we have selected fewer than k items total, and w(A[i]) ≥ threshold then
            Accept A[i]
        else
            Reject A[i]
        end if
    end for
end if
```

Next, we turn our attention to the value obtained from Z. Let $y_1 > y_2 > \ldots > y_l$ be the l largest elements from Y. We let q_i be the number of elements between y_{i-1} and y_i, and $q = \sum_{i=1}^{l} q_i$. Intuitively, q is the number of elements which we would pick based on our threshold, if our total number of choices could be greater than k. We would like this number to be close to l - if it is too large, we will pick many worthless elements, and if it is too small, our threshold will be too high to realize that some of the valuable items are worth taking. This quantity q is bounded in distance from l by relating the q_i to geometrically distributed random variables. We get that the expected value of the elements selected from Z is at least $\left(1 - \frac{1}{2\sqrt{k}}\right)\frac{v}{2}$. Adding this to the value obtained from Y, we can confirm the induction hypothesis, giving a total expected value of at least $\left(1 - \frac{5}{\sqrt{k}}\right)v$.

3.2 A $\log k$ Approximation on General Matroids [2]

This algorithm is almost identical to the classical secretary problem. We observe the first half of the input, set a threshold based on this sample, and choose every element in the second half which is greater than this threshold (with the added requirement that we can not pick an element which would break independence). While before, we simply chose the maximum value in the first half as a threshold, we now scale this maximum value down, dividing by some number between 1 and the rank of the matroid, k. It is slightly difficult to motivate this threshold. We can examine a few cases - if the values in the optimal solution are roughly uniform, then scaling by a factor of 1 or 2 is most appropriate, since it is likely that we'll find several more elements with weights around this value. However, if the optimal solution has a few elements with very large weight, then scaling by a factor close to k is appropriate - we have likely already missed many of the larger values, so we must lower our threshold to hope to

make up for missing the large elements by getting many smaller elements. Randomizing over the choice of scaling serves to balance between all these cases, and rather than having a perfect algorithm in one of these cases, we have an approximate algorithm in all cases.

More formally, the algorithm is as follows: Let S be the first $\frac{n}{2}$ elements. Reject all elements from S. Let l be the element with the maximum weight in S, and $w(l)$ be its weight. We select a random integer j from 0 to $\lceil \log k \rceil$, and let our threshold be $t = \frac{w(l)}{2^j}$ After observing S, we initialize our set B to $\emptyset$. Then, for all remaining items x, we accept it if $B \cup \{x\} \in \mathcal{I}$ and $w(x) \geq t$, and reject otherwise. Note that we need very little information to run this algorithm, as this is oblivious to the structure of the matroid. It turns out that we don't even need the rank of the matroid, as we can estimate the rank based on the rank of S at the cost of a constant factor to our approximation (this method is detailed explicitly in the algorithm in the following section). The only knowledge we actually need is an oracle for checking if a set is independent or not.

Algorithm 4 $O(\log k)$ approximation for the matroid secretary problem

```
for i = 1, ..., n/2 do
    Reject A[i]
end for
j ← Uniform(0, log k)
threshold ← max_{i∈{1,...,n/2}} w(A[i]) / 2^j
B ← ∅
for i = n/2 + 1, ..., n do
    if w(A[i]) ≥ threshold & B ∪ {A[i]} ∈ I then
        B ← B ∪ {A[i]}
    end if
end for
return B
```

We need a few definitions for the analysis. Let $v_1 \geq v_2 \geq \ldots \geq v_k$ be the values of elements in the optimal solution B^*. We consider only the values that are at least $\frac{v_1}{k}$ - these values will be $v_1, \ldots, v_q$. Note that $\sum_{i=1}^{q} v_i \geq \frac{1}{2} \sum_{i=1}^{k} v_i$, since the values we excluded are sufficiently small. For a set A, we define $n_i(A)$ to be the number of elements which are at least v_i, and $m_i(A)$ to be the number of elements which are at least $\frac{v_i}{2}$. This factor of two becomes relevant due to the fact that our threshold $\frac{l}{2^j}$ is a value divided by a power of 2.

We show a $32\lceil \log k \rceil$ approximation factor. This is equivalent to showing a factor of $16\lceil \log k \rceil$ approximation to the sum of the largest q values from the optimal solution. The largest q values have a sum $v_q n_q(B^*) + \sum_{i=1}^{q-1}(v_{i+1} - v_i) n_i(B^*)$. If B is our algorithm's solution, the value is at least $\frac{v_q}{2} m_q(B) + \sum_{i=1}^{q-1}(\frac{v_{i+1}}{2} - \frac{v_i}{2}) m_i(B)$. Therefore, it suffices to show that $8(\log k) m_i(B) \geq n_i(B^*)$ in expectation.

First, we consider the special case of $i = 1$. The analysis of this case is similar to the classical secretary algorithm. We know $n_i(B^*) = 1$, by definition. We know that S contains the second largest element, but not the largest element with probability at least $\frac{1}{4}$. If we choose the

threshold with $j = 0$ (an event that occurs with probability $\frac{1}{\log k}$), we will select the largest element. Therefore, the expected value of $m_1(B)$ is at least $\frac{1}{4 \log k}$.

Next, we consider $i > 1$. This time, we condition that the largest element is in the sample S, which occurs with probability $\frac{1}{2}$. We also condition that our choice of j is the one satisfying $\frac{v_i}{2} \leq \frac{w(l)}{2^j} \leq v_i$. Such a j exists based on the fact that $v_i \geq \frac{v_1}{k} = \frac{w(l)}{2^{\log k}}$, and we pick it with probability $\frac{1}{\log k}$. There are at least i items which exceed the threshold, and in expectation, at least $\frac{i-1}{2} \geq \frac{i}{4}$ of these items are in the second half of the input. Therefore, we'll pick at least $\frac{i}{4}$ items using this algorithm. Given the above conditionings, $m_i(B) \geq \frac{i}{4}$ in expectation. We remove the conditioning by multiplying by their probability, giving $E[m_i(B)] \geq \frac{i}{8 \log k}$. Since $n_i(B^*) = i$, this gives the desired result.

3.3 A $\log k$ Approximation on General Matroids in the Strict-Comparison Model [8]

Recall that in the strict-comparison model, we do not obtain the specific weight of an item, we only learn which elements it is greater than and less than. In the auction setting, this might not be a necessary restriction - it seems impractical to design a mechanism in which we ask agents their relative value compared to other agents. One can imagine settings closer to the literal secretary problem - where we are trying to hire a good secretary, and we can only compare applicants relative to each other. It does not seem obvious where this sort of situation might apply in a matroid setting. However, this problem is most interesting due to the challenge - this is a non-trivial restriction to the matroid secretary problem, and from a theoretical standpoint, it is interesting that we can still approximate it to within the same factor as the best known algorithm (at the time).

This algorithm is closer in flavor to the uniform matroid approximation than the other general matroid algorihm. In the general matroid case, we throw out a lot of information - we are only concerned with the maximum element of the sample, and we scale it by some factor. On the other hand, in the uniform matroid case, we used the relative ranking of prior elements extensively - in particular, we assumed that the greatest l values in some fraction p of the sequence behaves like the greatest $\frac{l}{p}$ values overall. This is the case here - we compute the optimal basis of the sample, and assume that this is representative of the optimal basis for the entire sequence. Specifically, we set the threshold equal to one of the elements in the optimal basis of the sample. However, as in the previous general matroid algorithm, we run into the same problem - we don't know how the values are distributed. Therefore, to avoid adversarial distributions, we randomize over the choice of our threshold - we can pick one of the larger or smaller values of the optimal basis, based on the outcome of a random drawing.

We now describe the algorithm in more detail. The algorithm has a number of peculiar characteristics. First, with probability $\frac{1}{2}$, we simply run the classical secretary algorithm. As before, this will be used to give a constant factor approximation when the rank of the matroid is low. Otherwise, we choose a stopping point $m \sim Binom(n, \frac{1}{2})$. We reject the first m samples, which we denote by S. We then run the greedy algorithm to compute a maximum weight independent set on S, which we denote $A = \{a_1, \ldots, a_r\}$, where $a_1 \geq \ldots \geq a_r$. If we happen to know the

rank k, we choose $t = \lfloor \log_3 k \rfloor$. Otherwise, we choose t to be either $\lfloor \log_3 r \rfloor$ or $\lfloor \log_3 r \rfloor + 1$ with equal probability. The idea is that, in expectation, the rank r of the first half is a good estimate of the overall rank. Next, j is an integer, chosen uniformly randomly between 0 and t. We set a threshold equal to $w(a_{3^j})$ - for the remainder of the input, we add an item x to our solution B if $B \cup \{x\} \in \mathcal{I}$ and $w(x) \geq w(a_{3^j})$, and reject it otherwise.

Algorithm 5 $O(\log k)$ approximation for matroid secretary problem in strict-comparison setting

```
p ← Bernoulli(1/2)
if p = 0 then
    Run Algorithm 2, the classic secretary algorithm
else
    m ← Binomial(n, 1/2)
    for i = 1, ..., m do
        Reject A[i]
    end for
    B ← Greedy(A[1], ..., A[m]), where Greedy is Algorithm 1
    S ← sort(B)
    t ← ⌊log_3 k⌋
    j ← Uniform(0, t)
    threshold ← w(S[3^j])
    for i = m + 1, ..., n do
        if w(A[i]) ≥ threshold & B ∪ {A[i]} ∈ I then
            B ← B ∪ {A[i]}
        end if
    end for
    return B
end if
```

A theme through the analysis of this algorithm (and indeed, many secretary algorithms) is that a random sample should be "pretty close" to the complement of this sample. This is made rigorous through the use of Chernoff bounds.

We start by assuming we know the rank. After, we will show that if we don't know the rank, our guess is correct with constant probability.

Here, we see the same useful property of sampling m from the binomial distribution as we did in the uniform matroid case. Particularly, if we choose S to be the first $m \sim Binomial(n, \frac{1}{2})$ elements, then each element is in S with probability $\frac{1}{2}$ independently, where the randomness is over both the choice of m and the random permutation. Therefore, in expecation, we have half the optimal solution in the first half, and $E[w(A)] \geq \frac{1}{2} w(OPT)$.

To compute the expected weight of the set returned by the algorithm, we do a weighted sum over each of the possible values of j. We know that each element returned by the algorithm is at least $w(a_{3^j})$ for a fixed j, so we sum over the expectation of $|ALG| \cdot w(a_{3^j})$. The number

of elements selected by the algorithm is the rank of the unsampled elements larger than our threshold, $w(a_{3^j})$. This is at least the number of unsampled elements larger than the 3^jth largest element overall. We'd expect this to be roughly $\frac{3^j}{2}$ - a more concrete statement is made through the use of Chernoff bounds. After lower bounding each term in the overall sum by the weight of a range of values, and some arithmetic, we get that $O(\log r)E[w(ALG)] \geq E[w(A)]$. Using the prior relationship between $E[w(A)]$ and $w(OPT)$, we conclude the desired result, $O(\log r)E[w(ALG)] \geq w(OPT)$.

In order to alleviate the problem of not knowing the rank, we consider the sampled rank r - the rank of the optimal solution on the first half, A. This can only be less than the actual rank of the matroid, which is k. Again using Chernoff bounds, one can show that, with constant probability, the sampled rank is at least $\frac{k}{3}$. If so, then value of $\lfloor \log_3 r \rfloor$ is either $\lfloor \log_3 k \rfloor$ or $\lfloor \log_3 k \rfloor - 1$, which are the two possible choices. Therefore, we will guess the rank with constant probability, multiplying our competitive ratio by only a constant factor, and still allowing us to be $O(\log k)$-competitive with the optimal solution.

3.4 A $\sqrt{\log k}$ Approximation on General Matroids [4]

The most recent result on the general matroid secretary problem is a $\sqrt{\log k}$ approximation algorithm. While this algorithm is very complex, and full description and analysis is beyond the scope of this survey, it uses several techniques from other algorithms, as well as some brand new techniques. Therefore, we will provide a rough description of the algorithm and some intuition about why it works.

First, the algorithm requires each weight to be a power of 2. If this is not the case, then we can treat each value as if it were rounded to the next power of 2, and we only double the competitive ratio (at worst). We group the values into buckets - a bucket will contain all elements with a given value.

Similar to a previous algorithm, we run the classical secretary algorithm on the problem with probability $\frac{1}{2}$. This allows us to handle the case where the largest element is at least the expected payoff of the algorithm. The rest of the analysis depends on the largest element being sufficiently small, and the classical algorithm provides a simple method for obtaining the desired approximation if this is not the case.

As in many other algorithms, we sample a set S, which is the first $m \sim Binomial(n, \frac{1}{2})$ elements. After S, we estimate the value of the optimal solution and the rank of the matroid. These estimates are based off the corresponding values computed on S - we can guarantee these estimates to be close to the actual values by Chernoff bounds. After this point, the algorithm diverges into two separate algorithms, based on certain properties of the elements of S. The authors of this paper name these algorithms the simple algorithm and the protection algorithm, respectively.

The simple algorithm could also be named the hindsight algorithm. We run this algorithm if one of two properties holds. If we knew the given property was going to hold before the sequence started, we would have exploited it. Since we can't go back, we simply exploit the property from that point onwards. Using symmetry and Chernoff bounds, it is likely that the remaining elements have some approximation of the property as well.

The first property that triggers the simple algorithm is if there exists a single bucket containing elements with a sufficiently high rank and weight. The second property that triggers the simple algorithm is if there exists a set of buckets X containing a subset of "high independence" of sufficient weight. By this, we mean that, for any element in this set, it can be added to any other independent set made from elements of X without breaking independence. For either of these properties, we continue by selecting any element which falls into the relevant bucket(s). It is likely that this sample is representative, so following this rule will give a comparable result.

The protection algorithm is significantly more complicated. We would like to avoid the following bad scenario: we have to turn down a high weight item since we accepted conflicting low weight items earlier. Ideally, when we were faced with the decision of choosing the low weight items, we would like to be able to look into the future and see if we'd be blocking valuable items later. Of course, we can't do this - instead, we look at the elements we already sampled. For some buckets, we pretend we selected some of the heavier items before, and only add new items if they are independent from the union of our current solution and these heavy items. Of course, there is some balancing of parameters present - if we take too many heavy items, we will not take any new items.

Further analysis shows that, no matter which of these subroutines is run, we get an approximation of $\sqrt{\log k}$, which is currently the best known approximation to the matroid secretary problem.

4 Conclusion

In this paper, we surveyed a number of algorithms for the matroid secretary problem. While the settings vary, we saw a number of themes common to several of the approaches, as well as some interesting design differences. For example, all the algorithms here reject some prefix of the input. However, this prefix length can vary - sometimes we choose a deterministic length, other times we choose a length based on a draw from a binomial distribution, depending on which is more convenient for analysis. We often assume some properties of the prefix to be representative of the sequence of the whole. Sometimes we compute a threshold based on a complicated function of an element's weight, sometimes we set this function to be the identity. These techniques provide a useful set of tools for analysis of matroid secretary problems, though it seems likely that new techniques will have to be employed to reach the elusive $O(1)$-competitive algorithm.

References

[1] M. Babaioff, N. Immorlica, D. Kempe, and R. Kleinberg. A knapsack secretary problem with applications. In *Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques*, APPROX '07/RANDOM '07, pages 16–28, Berlin, Heidelberg, 2007. Springer-Verlag.

[2] M. Babaioff, N. Immorlica, and R. Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 434–443, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[3] M. Bateni, M. Hajiaghayi, and M. Zadimoghaddam. Submodular secretary problem and extensions. In *Proceedings of the 13th international conference on Approximation, and 14 the International conference on Randomization, and combinatorial optimization: algorithms and techniques*, APPROX/RANDOM'10, pages 39–52, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] S. Chakraborty and O. Lachish. Improved competitive ratio for the matroid secretary problem. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1702–1712. SIAM, 2012.

[5] N. B. Dimitrov and C. G. Plaxton. Competitive weighted matching in transversal matroids. In *Proceedings of the 35th international colloquium on Automata, Languages and Programming, Part I*, ICALP '08, pages 397–408, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] E. B. Dynkin. The optimum choice of the instant for stopping a Markov process. *Soviet Math. Dokl*, 4, 1963.

[7] R. Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '05, pages 630–631, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.

[8] J. A. Soto. Matroid secretary problem in the random assignment model. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, pages 1275–1284. SIAM, 2011.