

Secretary Problems: Laminar Matroid and Interval Scheduling

Sungjin Im *

Yajun Wang †

Abstract

The classical secretary problem studies the problem of hiring the best secretary from the secretaries that arrive in random order by making immediate and irrevocable decisions. After its interesting connection to online mechanism design was found [19, 20], the random order input assumption has been studied for a variety of problems. Babaioff et al. [4] formalized a general version of the secretary problem, namely the matroid secretary problem. In the problem, a secretary corresponds to an element in the universe U . The goal is to select the maximum weight independent set. They conjectured that the matroid secretary problem, for any matroid, allows a constant competitive algorithm. The conjecture is still open. Some constant approximation algorithms are currently known for some special cases of matroids. Another interesting type of secretary problems were studied where elements have non-uniform sizes as in the knapsack secretary problem [3, 6].

In this paper, we consider two interesting secretary problems. One is when the matroid is a laminar matroid, which generalizes uniform / partition / truncated partition matroids. For the laminar matroid problem, via a novel replacement rule which we call “kick next,” we give the *first* constant-competitive algorithm. The other is the interval scheduling secretary problem, which generalizes the knapsack secretary problem. In the problem, each job J_i arrives with interval I_i , processing time p_i and weight w_i . If J_i is accepted it must be scheduled during I_i , not necessarily continuously. The goal is to accept the jobs of the maximum total weight which are schedulable. We give a simple $O(\log D)$ -competitive algorithm and a nearly matching lower bound on the competitive ratio of any randomized algorithm, where D is the maximum interval length of any job.

*Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801. im3@cs.illinois.edu
Partially supported by NSF grants CCF-0728782, CNS-0721899, and Samsung Fellowship. This work was done while the author was visiting Microsoft Research Asia.

†Microsoft Research Asia Beijing, China. yajunw@microsoft.com

1 Introduction

In the classical secretary problem [15, 12, 14], one interviews n secretaries one by one who arrive in random order. The interviewer must make an immediate and irrevocable decision of whether to hire each secretary or not upon his or her arrival. The total number of secretaries n is known prior to the algorithm. The goal is to hire the best secretary. The beauty of the problem is that the random order assumption makes the problem surprisingly tractable which is otherwise hopeless¹. Furthermore, an optimal online algorithm is very simple: the algorithm that observes the first n/e secretaries, and from the remaining secretaries hires the first secretary who is better than anyone of the first n/e secretaries, successfully hires the best secretary with probability $1/e$. For the fascinating history of the secretary problems, we refer the reader to [13].

Recently its interesting connection to online mechanism design was revealed [19, 20]. The classical secretary problem, for example, exactly captures the situation where agents arrive with different values for a single item (if agents are assumed to arrive in random order). Accordingly, the essence of the secretary problem that the random input order overcomes the restriction of decisions being irrevocable, has been studied for a variety of problems.

One very interesting line of work was initiated by Babaioff et al. [4]. They formulated the matroid secretary problem. In the problem we are given a matroid $\mathfrak{M}(U, \mathcal{I})$. The elements arrive in random order. When an element arrives, it reveals its weight. The algorithm must make an immediate and irrevocable decision of whether to accept it or not. The goal is to select an independent set $X \in \mathcal{I}$ of the maximum total weight of the elements in the set. They gave an $O(\log r)$ -approximation for a general matroid, where r is the rank of the given matroid; henceforth if there is no confusion in the context, for simplicity, we will say an algorithm is c -approximation if it is c -competitive. They conjectured that any matroid secretary problem allows a constant approximation. This conjecture remains open. So far constant approximations have been found for several special cases of matroids. They include uniform / partition matroids [20, 3], truncated partition matroids [4], graphic matroids [1, 21] and transversal matroids [11, 21]. For definition of these matroids, see [24].

Another interesting line of work for secretary-type problems was done when elements have non-uniform sizes. The knapsack secretary problem is one of them. In this problem, items of different sizes and weights arrive in random order. One has to select some items which can be packed into the given knapsack of a limited size. It adds more difficulty that one item can take lots of space in which many smaller items can fit. Constant approximations were given for this problem [3, 6].

Although many progresses were made toward understanding the realm of secretary-type problems, it would be fair to say that our understanding is still limited. In this paper, we consider two interesting secretary problems. One is the secretary problem constrained on a laminar matroid, which generalizes uniform matroids, partition matroids and truncated partition matroids. A laminar matroid is defined as follows. Let \mathcal{F} be a laminar family of sets defined over U , i.e. for any two sets $B_1, B_2 \in \mathcal{F}$, it must be the case that $B_1 \subseteq B_2$ or $B_2 \subseteq B_1$ or $B_1 \cap B_2 = \emptyset$. Each $B \in \mathcal{F}$ is associated with a *capacity* $\mu(B)$. A set $S \subseteq U$ is in \mathcal{I} if and only if $\forall B \in \mathcal{F}, |B \cap S| \leq \mu(B)$. It makes the problem non-trivial that one element may belong to *multiple* (possibly more than a constant number of) sets in the family. Thus when an element is considered to be selected, one has to make sure that the element does not violate any constraint.

Laminar matroids are often found in practice. Say we want to elect some representatives for towns, cities and states. To better represent people, we may want to put a limit on the number of representatives for each town, city and state. This can be captured by a laminar matroid. Laminar matroids were specifically addressed as an important case in the submodular minimization on matroid constraints [7, 8].

The other problem we are considering is the problem which we call the Interval Scheduling Secretary

¹In the worst case input model, it is known that any randomized algorithm cannot have the best secretary with a probability greater than $\frac{1}{n}$.

Problem (ISSP), which generalizes the knapsack secretary problem. In this problem, there is a unique resource which is available during a time interval $[0, T]$, where $T > 0$ is an integer. Each agent (or job) J_i arrives in random order asking for the resource exclusively for p_i amount of time during interval I_i . The quantity p_i is called J_i 's processing time or equivalently its size. The job J_i gives a weight (or profit) of w_i if it is accepted. Here the number of jobs n is known prior to the algorithm. The accepted jobs must be schedulable; each accepted job J_i must be scheduled in its interval I_i . We allow preemption. Here by preemption we mean that a job does not have to be scheduled continuously. Any decision is irrevocable, i.e. the acceptance or rejection of each job cannot be revoked. Our goal is to select jobs giving the maximum total profit.

Our results: We give the *first* constant approximation for the laminar matroid secretary problem (LMSP).

Theorem 1.1. *There exists a constant-competitive polynomial time algorithm for the Laminar Matroid Secretary Problem.*

For the ISSP problem, we give a simple $O(\log D)$ -competitive algorithm, and a nearly matching lower bound on the competitive ratio of any randomized online algorithm.

Theorem 1.2. *For the ISSP problem, there exists an $O(\log D)$ -competitive algorithm, where $D = \max_{i \in [n]} |I_i|$.*

Theorem 1.3. *For the ISSP problem, any randomized online algorithm has a competitive ratio of $\Omega(\frac{\log D}{\log \log D})$, where $D = \max_{i \in [n]} |I_i|$. Further, this holds even when $|I_i| = p_i = w_i$ for all $i \in [n]$.*

The instance used to show Theorem 1.3 is fairly simple. The intervals form a laminar structure. That is, for any two intervals I_i and I_j , one contains the other or the two are disjoint. Further, each job has a weight equal to its length and processing time. This suggests that it is generally hard to obtain a constant approximation for the secretary problems constrained on some laminar structure.

Our techniques: In the Laminar Matroid Secretary Problem, the difficulty in obtaining a constant factor algorithm, as already mentioned, lies in that each element may belong to multiple (possibly more than a constant number of) sets in \mathcal{F} . In the secretary problems literature, a popular approach is to show that an element in the optimal solution is chosen by the algorithm with a constant probability. In this method, the element is shown to, with a constant probability, pass a certain constraint which varies depending on each secretary problem. Thus a naive extension of the standard approach fails since the failure probability accrues for multiple constraints. To overcome this hurdle, we consider a novel yet simple algorithm. Following the standard approach, we build a reference set for each $B \in \mathcal{F}$, which is the best solution constructed from the sample. The reference set is used to decide whether to accept a new arriving element i or not. We accept element i only when it can kick out one element of smaller weight from the reference set. If we accept i , we kick out an element of weight *next* to w_i from the reference set. Due to this replacement rule, elements are “locally” replaced; kicking out the smallest weight element does not have this local property. This helps the analysis in a setting such as a laminar matroid where multiple constraints intervene. We, however, remark that we do not know if some threshold-type algorithms might work. To our best knowledge, this “kicking the next element” has not been used in the secretary literature. Our algorithm is simple but the analysis is non-trivial. We do not show that each element in the optimal solution is accepted with a constant probability. Rather, we identify some “good” elements in the optimal solution whose total weight is greater than the sum of other elements in the optimal solution. Then we show that each good element is chosen by the algorithm with a constant probability. For the goal, we show that each good element can find plenty of victims that it can kick out overall for all sets in \mathcal{F} that the element belongs to. For a detailed overview of the analysis, see the first paragraph in Section 2.2.

Related works: The random input model was also considered in the online adwords problem [10, 16]. The setting where an accepted element can be canceled later with some penalty was considered in [9, 2].

Babaioff et al. [1] studied the secretary problem where the values of elements decrease over time. Gupta et al. [18] recently studied the matroid secretary problem with a submodular objective function. They gave an $O(\log r)$ -approximation for any matroid of rank r , and constant approximations for uniform matroids ([6] gives this result as well) and partition matroids. Bateni et al. [6] also considered multiple knapsack secretary problem and gave an $O(l)$ approximation where l is the number of knapsack constraints. We note that the result [21] is in fact on the maximum weight matching in bipartite graphs and hypergraphs, which generalizes the transversal matroid secretary problem. In the offline setting, the ISSP problem is now a classical problem. For the problem of selecting non-overlapping intervals of the maximum total weight, it is well known that there exists an optimal dynamic programming [17]. Lawler gave a pseudo polynomial time algorithm for ISSP [22]. For more pointers on ISSP, we refer the reader to [5].

Organization: In Section 2, we give the main algorithm for the Laminar Matroid Secretary Problem and prove its constant competitiveness. In Section 3, we study the Interval Scheduling Secretary Problem. Finally, we conclude with open problems in Section 4.

2 Monotone Laminar Secretary Problem

In this section, we consider the Monotone Laminar Secretary Problem (MLSP). We first set up some notation. Let $U = \{1, 2, \dots, n\}$ be the set of elements. Element i has weight w_i . A laminar family \mathcal{F} is given over U . WLOG, we can assume that $U \in \mathcal{F}$. Each set A in \mathcal{F} is associated with capacity $\mu(A)$. Again, WLOG, we can assume that $\mu(A) < \mu(B)$ for any $A, B \in \mathcal{F}$ such that $A \subset B$; otherwise the constraint for A is redundant. Given $X \subseteq U$, let $w(X)$ denote the total weight of all elements in X . We say that $X \subseteq U$ is feasible if for all $A \in \mathcal{F}$, $|A \cap X| \leq \mu(A)$. The goal is to find a feasible set of elements $X \subseteq U$ which gives the maximum total weight.

2.1 Algorithm

Let $M(i)$ denote the inclusion-wise minimal set B in \mathcal{F} such that $i \in B$. We say $B_1 \in \mathcal{F}$ is a child of $B_2 \in \mathcal{F}$ if $B_1 \subset B_2$ and there exists no intermediate set $B' \in \mathcal{F}$ such that $B_1 \subset B' \subset B_2$. Naturally, B_2 is said to be the parent of B_1 . We denote it by $B_2 = p(B_1)$. The k_{th} closest ancestor of B_1 is denoted by $p^{(k)}(B_1)$. Thus when B_2 is the parent of B_1 , then we can write it as $B_2 = p^{(1)}(B_1) = p(B_1)$.

For any $A, B \in \mathcal{F}$ s.t. $A \subseteq B$, we define $\text{Chain}[A, B]$ to be the sequence of sets in \mathcal{F} starting with A and ending with B where each set is a child of the following set. Notation-wise, by $\text{Chain}[A, B]$ we sometimes mean just the collection of sets on the chain. For simple notation, to denote all sets in \mathcal{F} that i is in, we may interchangeably use $\text{Chain}[M(i), U]$ or $\mathcal{F}(i)$. For any $V \subseteq U$ and $B \in \mathcal{F}$, let $\text{OPT}_V(B)$ denote the optimal feasible solution that can be obtained from $V \cap B$. We present the main algorithm as follows.

Algorithm KICKNEXT FOR THE GENERAL LAMINAR SECRETARY PROBLEM (LMSP):

Let $t \leftarrow \text{Binom}(n, 1/2)$ and $S = [t]$.

for each $B \in \mathcal{F}$

let $R(B) \leftarrow \text{OPT}_S(B)$

for each $i \in T = [n] \setminus [t]$ (in random order)

 with probability $1/10^3$, $\text{AddIt} \leftarrow 1$, otherwise $\text{AddIt} \leftarrow 0$

for each $B \leftarrow \text{Chain}[M(i), U]$

if $R(B) \neq \emptyset$ and w_i is bigger than the weight of some element in $R(B)$ **then**

if $\text{AddIt} = 1$ **then**

 add i to $\text{SOL}(B)$ and remove the element of the largest weight next to w_i from $R(B)$

else

 break (consider next i)

return $\text{SOL}(U)$.

Remark 2.1. *The probability used in the algorithm can be substantially improved by addressing some cases separately.*

In the algorithm, we obtain a sample $S = [t]$ by observing the first t elements, where t is a random number obtained from the binomial distribution $Binom(n, 1/2)$, i.e. the number of heads when a fair coin is tossed n times. It is easy to see that the sample S can be equivalently obtained by sampling each element with a half probability from $[n]$. For each $B \in \mathcal{F}$, we set $R(B)$ to be the optimal solution for the elements that are restricted to $B \in \mathcal{F}$ and appear in the sample S . We call $R(B)$ the *reference set* for B . To decide whether or not to add a new arriving element to the solution $SOL(U)$, we need to check if our solution, when i is added, does not violate any capacity constraint for any set in $\mathcal{F}(i)$. Thus we consider each B on $Chain[M(i), U]$ in the order that the sets are ordered on the chain.

The element i can be added to the solution only if it can find an element of smaller weight in $R(B)$ for each $B \in Chain[M(i), U]$. However, it is added to the solution with a small probability even though such a condition is satisfied for each $B \in \mathcal{F}(i)$. Note that `ADDIT`, the value used in the decision of whether or not to add i to SOL , is obtained only *once* for each element. It is important to note that the element of weight *next* to w_i is removed from $R(B)$ (not the element of the smallest weight). To our best knowledge, this replacement rule does not seem to have been used before in the secretary problems literature. After the algorithm ends, the set $SOL(B)$ is the solution of our algorithm restricted for B . Thus the algorithm returns $SOL(U)$ as the final solution. We note that maintaining $SOL(B)$ except when $B = U$ is solely for the purpose of analysis.

It is easy to see that the following simple algorithm `BottomToTop (BTT)` gives the optimal solution for $B \cap S$ for each $B \in \mathcal{F}$; the sub-procedure is implemented via dynamic programming.

Algorithm BTT(B):
If B has no child **then**
 return the $\mu(B)$ elements in $B \cap S$ of the largest weights.
else
 let $\mathcal{C}(B) \subseteq \mathcal{F}$ be the collection of the children sets of B .
 return the $\mu(B)$ elements of the largest weights
 from $[B \cap S \setminus \bigcup_{B' \in \mathcal{C}(B)} B'] \cup \bigcup_{B' \in \mathcal{C}(B)} \text{BTT}(B')$.

Note that if $OPT_S(B)(= \text{BTT}(B))$ selects some elements from B' , which is a child of B , then they must come from $OPT_S(B')$.

2.2 Analysis

We first give an overview of the analysis. The algorithm accepts an element by kicking out an element of smaller weight in $R(A)$ for all $A \in \mathcal{F}(i)$. This can be seen as replacing an element in $R(A)$ with an element of larger weight in $SOL(A)$. Thus the capacity constraint for each set in \mathcal{F} is easily satisfied. Note that an element i may belong to a large number of sets in \mathcal{F} , and thus to be accepted it needs a victim element to kick out in $R(A)$ for each $A \in \mathcal{F}(i)$ when it arrives. Intuitively, it is more likely to be accepted if it can find many potential victims (of smaller weights). Thus we define a backward rank for each element i and each $A \in \mathcal{F}$ which is the number of potential victims that i can kick out in $R(A)$. We show that the probability that there is no element remaining in $R(A)$ for i to kick out when it arrives, is exponentially small depending on the backward rank of i in $R(A)$. Thus if i 's backward rank grows overall for the sets on $Chain[M(i), U]$, we call such an element good, then we are able to show that i is selected by the algorithm with a constant probability. Then we show that there are plenty of good elements which account for the majority of the total weight of the optimal solution, and this completes our analysis.

We start with showing that the algorithm gives a feasible solution.

Lemma 2.2. *The algorithm KickNext returns a feasible solution.*

Proof. We show that for any $B \in \mathcal{F}$, $|\text{SOL}(U) \cap B| \leq \mu(B)$. We start with making an observation. Note that $\text{SOL}(U) \cap B \subseteq \text{SOL}(B)$. This is because the algorithm adds the element $i \in B$ to $\text{SOL}(U)$ only if it adds it to $\text{SOL}(B)$. Thus to prove the lemma it is sufficient to show that $|\text{SOL}(B)| \leq \mu(B)$. Recall that $R(B)$ is initially $\text{OPT}_S(B)$. Since each element in $\text{SOL}(B)$ was added when it kicks out an element from $\text{OPT}_S(B)$, it follows that $|\text{SOL}(B)| \leq \mu(B)$, completing the proof. \square

We now turn to showing the quality of the solution. We will show that our algorithm captures *some* “good” elements in OPT with a constant probability. We need more notation for further analysis including definition of “bad” elements in OPT . Consider any element $i \in \text{OPT}$. Define the *backward* rank of i for B , denoted by $\text{brank}(i, B)$, to be the number of elements in $\text{OPT}(B)$ of weight smaller than w_i ; for example, if i is the element of the smallest weight in $\text{OPT}(B)$ then $\text{brank}(i, B) = 1$. Similarly let $\text{brank}_S(i, B)$ denote the analogous quantity for the elements in $\text{OPT}_S(B)$. Note that $\text{brank}(i, B)$ does not depend on the sample S but $\text{brank}_S(i, B)$ does.

Intuitively, an element $i \in \text{OPT}$, when $\text{brank}_S(i, B)$ is large, is more likely to be picked by the algorithm, by kicking out an element of smaller weight from $R(B)$. However, the value $\text{brank}_S(i, B)$ varies depending on the sample S . So for the sake of analysis, we will use its lowerbound $\text{brank}(i, B)$ which does not depend on S . The following proposition is not difficult to show by observing how the algorithm BTT works.

Proposition 2.3. *For any $i \in \text{OPT} \cap B$ and any sample $S \subseteq U$, we have that $\text{brank}_S(i, B) \geq \text{brank}(i, B)$. In particular, if $i \notin S$, then we have $\text{brank}_S(i, B) > \text{brank}(i, B)$.*

Proof. For a set $Z \subseteq U$, let $Z^{\geq x}$ denote the elements in Z whose weights are no smaller than x . We show a stronger claim that for any x and any $B \in \mathcal{F}$, $|\text{OPT}^{\geq x}(B)| \geq |\text{OPT}_S^{\geq x}(B)|$. We show it by induction. Let $\mathcal{C}(B)$ denote the children of B in \mathcal{F} . Suppose that for any $A \in \mathcal{C}(B)$, $|\text{OPT}^{\geq x}(A)| \geq |\text{OPT}_S^{\geq x}(A)|$. If $|\text{OPT}^{\geq x}(B)| = \mu(B)$, we are done. So suppose $|\text{OPT}^{\geq x}(B)| < \mu(B)$. Let $B' := B \setminus \bigcup_{A \in \mathcal{C}(B)} A$. By observing how the algorithm BTT works, it is easy to see that $|\text{OPT}^{\geq x}(B)| = |B'^{\geq x}| + \sum_{A \in \mathcal{C}(B)} |\text{OPT}^{\geq x}(A)|$. Then, by the induction hypothesis and $|B'^{\geq x}| \geq |(B' \cap S)^{\geq x}|$, the claim easily follows. Thus we have that $\text{brank}(i, B) = \mu(B) - |\text{OPT}^{\geq w_i}(B)| \leq \mu(B) - |(\text{OPT}_S^{\geq w_i}(B))| = \text{brank}_S(i, B)$. The second part of the lemma is easily obtained by setting $x = w_i + \epsilon$ where $\epsilon > 0$ is a sufficiently small constant. More concretely, if $i \notin S$, then we have $\text{brank}(i, B) = \mu(B) - |\text{OPT}^{\geq w_i}(B)| = \mu(B) - (|\text{OPT}^{\geq w_i + \epsilon}(B)| + 1) \leq \mu(B) - |\text{OPT}_S^{\geq w_i + \epsilon}(B)| - 1 = \text{brank}_S(i, B) - 1$. \square

We now define bad elements $\text{OPT}^{\text{bad}} \subseteq \text{OPT}$. For any integer $k \geq 0$, define $\text{OPT}_k^{\text{bad}}$: $i \in \text{OPT}_k^{\text{bad}}$ if and only if k is the smallest integer such that $|\{B \in \mathcal{F}(i) : \text{brank}(i, B) \leq k\}| \geq 8(k+1)^4$. Let $\text{OPT}^{\text{bad}} := \bigcup_{k \geq 0} \text{OPT}_k^{\text{bad}}$. In words, an element i in OPT^{bad} has a small value of brank for many sets in $\mathcal{F}(i)$. To guarantee the quality of our solution $\text{SOL}(U)$, it is not enough to bound the number of bad elements. We show that there are good elements in OPT whose total weight is greater than that of bad elements in OPT . More concretely, we will show that there exists a function f of mapping each bad element in OPT^{bad} to a distinct element in OPT^{good} of larger weight.

Lemma 2.4. *There exists an injective function $f : \text{OPT}^{\text{bad}} \rightarrow \text{OPT}^{\text{good}}$ s.t. $w_i < w_{f(i)}$ for $\forall i \in \text{OPT}^{\text{bad}}$.*

To prove Lemma 2.4, we will show, in Lemma 2.6, that in any number of elements of the largest weights in OPT , there are only a fraction of bad elements. After that, in Lemma 2.8, we will bound the probability that $i \in \text{OPT}^{\text{good}}$ is cannot be added to $\text{SOL}(B)$ for any $B \in \mathcal{F}(i)$, is exponentially small depending on $\text{brank}(i, B)$. Since any good element i , roughly speaking, does not have the same value of brank for many sets, we will be able to bound the sum of the bad probabilities for all $B \in \mathcal{F}(i)$.

Let $\text{OPT}_\ell^{\text{large}}$ denote the ℓ elements in OPT of the largest weights. We say $B \in \mathcal{F}$ is OPT_ℓ -different if $\text{OPT}_\ell^{\text{large}} \cap B \supset \text{OPT}_\ell^{\text{large}} \cap B'$ for any child B' of B ; note that “ \supseteq ” trivially holds (When B has no child, then B is said to be OPT_ℓ -different if $\text{OPT}_\ell^{\text{large}} \cap B \neq \emptyset$). In words, B has at least one more element from $\text{OPT}_\ell^{\text{large}}$ than any child of B . Let $\mathcal{F}_\ell^{\text{diff}}$ denote all OPT_ℓ -different sets in \mathcal{F} . From the definition of OPT_ℓ -different sets, we can easily bound the total number of OPT_ℓ -different sets.

Proposition 2.5. *For any $\ell \geq 0$, $|\mathcal{F}_\ell^{\text{diff}}| \leq 2\ell$.*

We are now ready to bound the number of bad elements in $\text{OPT}_\ell^{\text{large}}$.

Lemma 2.6. *For any $\ell \geq 0$, $|\text{OPT}^{\text{bad}} \cap \text{OPT}_\ell^{\text{large}}| < \frac{1}{2}\ell$.*

Proof. Consider any fixed integer $k \geq 0$. For $i \in \text{OPT}_\ell^{\text{large}}$, let d_i denote the number of sets in $\mathcal{F}(i)$ for which i 's brank is at most k . Note that $i \in \text{OPT}^{\text{bad}}$ if and only if $d_i \geq 8(k+1)^4$. Let $d = \sum_{i \in \text{OPT}_\ell^{\text{large}}} d_i$. It is easy to see that $|\text{OPT}_k^{\text{bad}} \cap \text{OPT}_\ell^{\text{large}}| \leq \frac{d}{8(k+1)^4}$. Thus we will focus on bounding d . For $B \in \mathcal{F}(i)$, we say that i increases brank for $p(B)$ when $\text{brank}(i, p(B)) > \text{brank}(i, B)$; we will say that i 's brank increases for B if B has no child. We claim that all elements in $\text{OPT}_\ell^{\text{large}} \cap B$ increase their branks for any $B \notin \mathcal{F}_\ell^{\text{diff}}$. Notice that B , by definition of $\mathcal{F}_\ell^{\text{diff}}$, has a child C such that $B \cap \text{OPT}_\ell^{\text{large}} = C \cap \text{OPT}_\ell^{\text{large}}$. Note that the elements in $B \cap \text{OPT}_\ell^{\text{large}} = C \cap \text{OPT}_\ell^{\text{large}}$ have the largest branks. Thus the claim follows from the fact that $|\mu(B)| > |\mu(C)|$.

We now count how much each $B \in \mathcal{F}$ contributes to d . Suppose that B does not have any OPT_ℓ -different set which is within B 's k_{th} descendant, i.e. $B \neq D^{p(k')}$ for any $D \in \mathcal{F}_\ell^{\text{diff}}$ and any $0 \leq k' \leq k$. Then by the above claim, all elements in $\text{OPT} \cap B$ have brank of at least $k+1$ for B . Thus B can contribute to d only when B has any OPT_ℓ -different set that is within B 's k_{th} descendant. It is not difficult to see that there are at most $2(k+1)\ell$ those sets via Proposition 2.5. Further, each set in \mathcal{F} can contribute to d at most $(k+1)$. Thus we conclude that $d \leq 2(k+1)^2\ell$, and we have $|\text{OPT}_k^{\text{bad}} \cap \text{OPT}_\ell^{\text{large}}| \leq \frac{2(k+1)^2\ell}{8(k+1)^4} \leq \frac{\ell}{4(k+1)^2}$. Thus we obtain $|\text{OPT}^{\text{bad}} \cap \text{OPT}_\ell^{\text{large}}| \leq \sum_{k \geq 0} |\text{OPT}_k^{\text{bad}} \cap \text{OPT}_\ell^{\text{large}}| \leq \sum_{k \geq 0} \frac{\ell}{4(k+1)^2} = \frac{\pi^2}{24}\ell \leq 0.42\ell$. \square

Using Lemma 2.6, it is easy to prove Lemma 2.4.

Proof of [Lemma 2.4] For each $i \in \text{OPT}^{\text{bad}}$, let N_i denote the elements in OPT^{good} whose weights are greater than w_i . By Hall's theorem, it is enough to show that for any $C \subseteq \text{OPT}^{\text{bad}}$, $|\bigcup_{i \in C} N_i| \geq |C|$. Let s denote the smallest weight element in C . Then it is easy to see that $\bigcup_{i \in C} N_i = N_s$. Say s has a rank of ℓ in OPT . Then by Lemma 2.6, $|C| < \frac{\ell}{2}$, and therefore $|\bigcup_{i \in C} N_i| > \ell - \frac{\ell}{2} = \frac{\ell}{2}$. This completes the proof. \square

Our remaining task is to show that each element $i \in \text{OPT}^{\text{good}}$ is chosen to be in $\text{SOL}(U)$ with a constant probability. For this goal, we will bound the probability of some bad events occurring. Since i may belong to multiple sets in \mathcal{F} , namely $\mathcal{F}(i)$, we need to check if our solution remains feasible with i being added for each $B \in \mathcal{F}(i)$. Let $\text{ALLKICKED}(B, i)$ denote the bad event that all elements of weight smaller than w_i in $\text{OPT}_S(B)$ are kicked out when the algorithm completes. In Lemma 2.8, we will show $\Pr[\text{ALLKICKED}(i, B) | i \notin S] \leq 3 \cdot 0.02^{\text{brank}(B, i)+1}$. Since the probability exponentially decreases depending on i 's brank which increases “overall” on $\text{Chain}[M(i), U]$, we will be able to bound the probability for all bad events for the element i .

To obtain an exponentially decreasing probability, we need the following lemma. We say that an element i qualifies for B if for each $B' \in \text{Chain}[M(i), B]$, there exists an element in $\text{OPT}_S(B')$ of smaller weight than w_i . Note that an element i will be considered to be included in $\text{SOL}(B)$ by the algorithm only if i qualifies for B . In the following lemma, we will bound the number of qualifying elements which appear in $T = U \setminus S$ between two adjacent elements in $\text{OPT}_S(B)$. A similar idea was used in the uniform matroid secretary problem [20].

Lemma 2.7. Consider any $B \in \mathcal{F}$. Let $\text{OPT}_S(B) = \{a_1, a_2, \dots, a_m\}$, where elements are sorted in decreasing order of weights. Let $w(a_\ell)$ denote the weight of a_ℓ ; for simple notation, let $w(a_0) = \infty$. Let $N_\ell(B) \subseteq T$, $1 \leq \ell \leq m$ denote the elements which qualify for B and whose weights are bigger than $w(a_\ell)$ and smaller than $w(a_{\ell-1})$. Let $I \subseteq [m]$. Let $n_\ell \geq 0$, $\ell \in I$ be any integer. Then $\Pr[|N_\ell(B)| = n_\ell, \forall \ell \in I] \leq \prod_{\ell \in I} \frac{1}{2^{n_\ell+1}}$. Further, for any $i \in U$, we have that $\Pr[|N_\ell(B)| = n_\ell, \forall \ell \in I \mid i \notin S] \leq \prod_{\ell \in I} \frac{1}{2^{n_\ell+1}}$.

Proof. To prove the lemma, we will show that $\Pr[|N_\ell(B)| = n_\ell \mid |N_{\ell'}(B)| = n_{\ell'}, \forall \ell' : \ell' \in I, \ell' < \ell] \leq \frac{1}{2^{n_\ell+1}}$ for any $n_{\ell'} \geq 0, \ell' \in I$ s.t. $\ell' < \ell$. Note that a sample S is an outcome. Conditional on the event that $(|N_{\ell'}(B)| = n_{\ell'}, \forall \ell' : \ell' \in I, \ell' < \ell)$, consider each case where $a_{\ell'} = x$ for an integer $x \geq 0$, where ℓ' is the largest index in I next to ℓ . Let us call this event $\varepsilon(x)$. Let $L(x) \subseteq U$ denote the set of elements whose weights are no smaller than x . It is not difficult to see that $U \setminus L(x)$ does not affect the event $\varepsilon(x)$. Conditional on $\varepsilon(x)$, consider each case where each element in $L(x)$ is fixed either in S or in T . Let $L'(x) \subseteq L(x)$ denote the elements that are fixed to be in S . Let $Y(\ell, B) \subseteq B \setminus L(x)$ denote the set of elements such that $j \in Y(\ell, B)$ if and only if $j \in \text{OPT}_{L'(x) \cup \{j\}}(B)$. If $|Y(\ell, B)| \leq n_\ell$, the event $|N_\ell(B)| = n_\ell$ cannot occur, thus the probability is 0; this is the reason why we obtain only an upper bound on the probability. So suppose that $|Y(\ell, B)| > n_\ell$. Note that all elements together from $Y(\ell, B)$ may not be included, since they together may violate some capacity constraints. It is easy to see that $|N_\ell(B)| = n_\ell$ occurs only if the n_ℓ element of the largest weights in $Y(\ell, B)$ are in T and the $(n_\ell + 1)$ th largest weight element is in S . This gives the desired probability. The second part of the lemma can be similarly obtained. \square

We now bound the probability that a bad event occurs. In the proof, Lemma 2.7 and Proposition 2.3 will be used with the Chernoff inequality. We remind the reader that $\text{brank}(B, i)$ does not depend on the sample S .

Lemma 2.8. Consider any $i \in \text{OPT}$ and any set $B \in \mathcal{F}(i)$. Then $\Pr[\text{ALLKICKED}(i, B) \mid i \notin S] \leq 3 \cdot 0.02^{\text{brank}(i, B)+1}$.

Proof. Let $\text{OPT}_S(B) := \{e_m, e_{m-1}, \dots, e_1\}$ where the elements are ordered in decreasing order of weights. Let $w(e_i)$ denote the weight of e_i . Let $d = \text{brank}(B, i)$. Suppose that $i \notin S$. Note that $w_i > w(e_{d+1})$ by Proposition 2.3. Let A_ℓ denote the number of elements in $\text{SOL}(B)$ of weight smaller than $w(e_{\ell+1})$; for simple notation, let $w(e_{m+1}) = \infty$. Let Q_ℓ denote the number of qualifying elements of weight smaller than $w(e_{\ell+1})$. It is not difficult to see that $\text{ALLKICKED}(i, B)$ occurs only if $A_\ell \geq \ell$ for some $d+1 \leq \ell \leq m$. Thus we focus on bounding $\Pr[\bigvee_{d+1 \leq \ell \leq m} (A_\ell \geq \ell)]$. To this end, we use

$$\Pr[A_\ell \geq \ell] \leq \Pr[Q_\ell \geq 20\ell] + \Pr[(Q_\ell < 20\ell) \wedge (A_\ell \geq \ell)].$$

We first bound $\Pr[(Q_\ell \geq 20\ell)]$. By Lemma 2.7, we know that Q_ℓ is stochastically dominated by the random variable for the number of heads observed until ℓ number of tails show up when a fair coin is repeatedly tossed. Thus the probability is bounded by the probability that at least 20ℓ heads are observed when a fair coin is tossed 21ℓ times, which is at most $\exp(-8.5\ell)$ by the Chernoff inequality (Theorem A.1).

We now bound $\Pr[(Q_\ell < 20\ell) \wedge A_\ell \geq \ell]$. We can upper bound it assuming that $Q_\ell = 20\ell$. Recall that each element i is added to $\text{SOL}(B)$ with a probability of $1/10^3$. We apply the Chernoff inequality (Theorem A.2) by setting $\mu = (20/10^3)\ell$ and $(1 + \delta) = 10^3/20$. Then, via a simple algebra, we have $\Pr[(Q_\ell < 20\ell) \wedge A_\ell \geq \ell] \leq (\frac{20}{10^3})^\ell$. Using the union bound of the probabilities, we obtain

$$\Pr \left[\bigvee_{\ell \geq d+1} (A_\ell \geq \ell) \right] \leq 3 \cdot 0.02^{d+1}$$

\square

We are now ready to prove the main lemma.

Lemma 2.9. *Each element $i \in \text{OPT}^{\text{good}}$ is chosen by the algorithm to be included in $\text{SOL}(U)$ with an $O(1)$ probability.*

Consider any element $i \in \text{OPT}^{\text{good}}$. We first bound the sum of all bad probabilities for i as follows.

$$\begin{aligned} & \Pr \left[\bigvee_{B \in \mathcal{F}(i)} \text{ALLKICKED}(i, B) \mid i \notin S \right] \leq \sum_{B \in \mathcal{F}(i)} \Pr \left[\text{ALLKICKED}(i, B) \mid i \notin S \right] \\ & \leq \sum_{d \geq 0} \sum_{B \in \mathcal{F}(i), \text{brank}(i, B) = d} \Pr \left[\text{ALLKICKED}(i, B) \mid i \notin S \right] \\ & \leq \sum_{d \geq 0} 8(d+1)^4 \cdot 3 \cdot 0.02^{d+1} < \frac{1}{4} \end{aligned}$$

The second last inequality is due to definition of OPT^{good} and Lemma 2.8. Since $\Pr[i \notin S] = \frac{1}{2}$, we have that $\Pr \left[\bigwedge_{B \in \mathcal{F}(i)} \neg \text{ALLKICKED}(i, B) \wedge i \notin S \right] > \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8}$. Thus when the element i arrives, with a probability of at least $\frac{3}{8}$, there exists at least one element remaining in $R(B)$ that i can kick out for each $B \in \mathcal{F}(i)$. And i is added to $\text{OPT}(U)$ with probability $\frac{1}{10^3}$. Thus we have that $i \in \text{OPT}(U)$ with a probability of at least $\frac{3}{8} \cdot \frac{1}{10^3}$. Via Lemma 2.9 and 2.4, the algorithm has a competitive ratio of at least $\frac{3}{16 \cdot 10^3}$, proving Theorem 1.1.

3 Interval Scheduling Secretary Problem

This section studies the Interval Scheduling Secretary Problem (ISSP). For the definition of the problem, see Section 1. Let D denote the maximum length of any interval I_i . In Section 3.1, we give a simple $O(\log D)$ -approximation and then in Section 3.2, show a nearly matching lower bound of $\Omega(\log D / \log \log D)$ on the competitive ratio that any online randomized algorithm can achieve.

3.1 $O(\log D)$ -approximation

In this section, we give a simple randomized algorithm with approximation factor $O(\log D)$. We pick an integer h uniformly randomly from $[0, \lceil \log_2 D \rceil]$. Let α be an integer which is sampled uniformly from $[0, 2^{h+2} - 1]$. Let $V_{h,\ell} = [2^{h+2}\ell - \alpha, 2^{h+2}(\ell+1) - \alpha]$, $\ell \geq 0$. A job i is considered by the algorithm only if $2^h \leq |I_i| < 2^{h+1}$ and I_i is fully contained in $V_{h,\ell}$ for some $\ell \geq 0$. For each $\ell \geq 0$, we run a different copy of the algorithm. Consider any fixed ℓ . We toss a fair coin. If it gives a head, run Dynkin's classical secretary algorithm [12]; here only weights are considered. Otherwise, run the knapsack secretary algorithm [3] with the knapsack having size 2^h .

Proof of [Theorem 1.2] We group the jobs in the optimal solution according to their associated interval sizes. Formally, J_i belongs to $\mathcal{C}_{h'}$ if $2^{h'} \leq |I_i| < 2^{h'+1}$, where $0 \leq h' \leq \lceil \log_2 D \rceil$. Let $\text{OPT}(h')$ denote the jobs that are in the optimal solution and also in $\mathcal{C}_{h'}$. Suppose $h = h'$, which happens with a probability of $\frac{1}{\lceil \log_2 D \rceil + 1}$. Consider any fixed $\ell \geq 0$. Let $V'_{h,\ell} = V_{h,\ell} \cap [0, T]$. Note that it may be the case that $|V'_{h,\ell}| \neq 2^{h+2}$. Let $\text{OPT}(h, \ell)$ denote the jobs in $\text{OPT}(h)$ whose intervals are fully contained in $V_{h,\ell}$. We let $w(\text{OPT}(h, \ell))$ denote the total weight (profit) of the jobs in $\text{OPT}(h, \ell)$. Let $A(h, \ell)$ and $w(A(h, \ell))$ denote the analogous set and quantity for the algorithm, respectively. Let $w(\text{OPT})$ denote the total weight of the jobs in the optimal solution. It is not difficult to see that the interval of each job in $\text{OPT}(h)$ is fully contained in $V_{h,\ell}$ for some $\ell \geq 0$ with at least a half probability. Thus we have

$$\sum_{h,\ell} \mathbb{E}[w(\text{OPT}(h, \ell))] \geq \frac{1}{2} w(\text{OPT}). \quad (1)$$

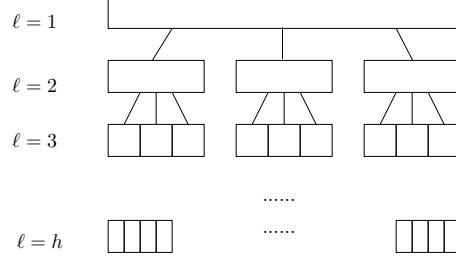


Figure 1: There are in total h levels of nodes in the h^2 -ary tree. In level ℓ , there are $h^{2(\ell-1)}$ nodes, each with size and weight $h^{-2(\ell-1)}$.

We now show that

$$\mathbb{E}[w(A(h, \ell))] \geq \frac{1}{\lceil \log_2 D \rceil + 1} \cdot \frac{1}{320e} \mathbb{E}[w(\text{OPT}(h, \ell))]. \quad (2)$$

We can assume that $|V'_{h,\ell}| \geq 2^h$; otherwise there is no job in $\text{OPT}(h, \ell)$. Let us say a job of processing time at least $2^h/2$ is “long”, and otherwise “short.” If the long jobs in $\text{OPT}(h, \ell)$ have a total weight greater than the short jobs in $\text{OPT}(h, \ell)$, then there exists a long job of weight at least $\frac{1}{16}w(\text{OPT}(h, \ell))$. This, with a $1/e$ probability, is captured by Dynkin’s algorithm [3] which runs with a half probability. Thus in this case, the expected profit is at least $\frac{1}{32e}w(\text{OPT}(h, \ell))$. Otherwise, we claim that the knapsack secretary algorithm with approximation factor $\frac{1}{10e}$, which runs with a half probability, achieves an expected profit of at least $\frac{1}{20e}w(\text{OPT}(h, \ell))$. Recall that we run the knapsack algorithm with the knapsack having size 2^h . Since $V'_{h,\ell}$ has length at least 2^h and each short job has length at most $2^h/2$, selecting the short jobs of the largest ratios of weight to processing time up to the total processing time 2^h gives a profit of at least $\frac{1}{16}w(\text{OPT}(h, \ell))$. Thus the knapsack secretary algorithm will give an expected profit of $\frac{1}{320e}w(\text{OPT}(h, \ell))$. It is not difficult to see that the jobs returned by the algorithm are schedulable. Since the above argument assumes that $h = h'$, which occurs with a probability of $\frac{1}{\lceil \log_2 D \rceil + 1}$, we obtain (2).

Hence from (1) and (2), the expected profit achieved by the algorithm is lower bounded as follows.

$$\sum_h \sum_\ell \mathbb{E}[w(A(h, \ell))] \geq \frac{1}{\lceil \log_2 D \rceil + 1} \cdot \frac{1}{640e} w(\text{OPT}) \quad \square$$

3.2 $\Omega\left(\frac{\log D}{\log \log D}\right)$ lower bound

We start with describing the “hard” instance. See Figure 1 for reference. In the instance each job J_i has size of the length of its associated interval, i.e. $p_i = |I_i|$. Thus J_i must be scheduled exactly on I_i . The jobs are randomly sampled from a h^2 -ary tree structure. By scaling, we assume the given entire interval is $[0, 1]$ (the top node), and the smallest job has size $h^{-2(h-1)}$ (in the h level); note that $D = h^{2(h-1)}$. In the ℓ -th level of the tree for $\ell \in [h]$, the intervals of the set of nodes are $\mathcal{L}_\ell := \{[\frac{k-1}{h^{2(\ell-1)}}, \frac{k}{h^{2(\ell-1)}}] \mid k \in [h^{2(\ell-1)}]\}$. In other words, \mathcal{L}_ℓ are the set of intervals that are obtained by partitioning the interval $[0, 1]$ seamlessly into subintervals of size $h^{-2(\ell-1)}$. Let $\mathcal{L} := \bigcup_{\ell \in [h]} \mathcal{L}_\ell$. Note that each interval in \mathcal{L}_ℓ when $\ell \geq 2$ is contained in exactly one interval in $\mathcal{L}_{\ell'}$ for any $1 \leq \ell' < \ell$. There are in total $H = \sum_{\ell \in [h]} h^{2(\ell-1)}$ nodes.

There will arrive H/h jobs. At each time $t \in [H/h]$, exactly one job J_t will arrive. Its interval I_t is sampled uniformly randomly from \mathcal{L} . If $I_t \in \mathcal{L}_\ell$, we will have $p_t = w_t = |I_t| = h^{-2(\ell-1)}$. Since at each time the interval is randomly sampled, our random instance is clearly oblivious to the random permutation which is assumed in the secretary problem.

Before giving the formal analysis, we give some intuition why any (randomized) online algorithm cannot do well. Since there are only H/h random jobs, all jobs only from one level \mathcal{L}_ℓ can give an expected profit of at most $\frac{1}{h}$. On the other hand, we will show that the expected profit of the optimal solution is at least

$1 - 1/e$, which means the optimal algorithm can carefully pack the nodes in different levels to achieve a good profit. However, selecting jobs only from one level, as in the $O(\log D)$ approximation algorithm, cannot give a good profit. Thus the algorithm should collect profits from many levels. A job of high profit (a node in higher level in the tree) arrives with a smaller probability in our samples. The algorithm, in order to capture a job of high profit, may have to wait while discarding jobs of smaller profit, sacrificing profits from lower levels. Further, the samplings are identical at all times, thus the standard technique used for the secretary problems, learning by sampling or waiting does not help here.

Lemma 3.1. *The optimal solution has an expected profit of at least $1 - 1/e$.*

Proof. Consider one node v in the lowest level, with size and length $h^{-2(h-1)}$. Let $C(v)$ be the set of nodes in the path from v to the root in the tree. If the algorithm accepts a node $u \in C(v)$, we say v is covered, i.e., there is a job scheduled during the time interval of v .

Clearly, the optimal algorithm will accept nodes as higher as possible in the tree to maximize the profit. Therefore, if v is not covered by the optimal algorithm, there is no node in $C(v)$ appeared in our H/h random samples. This happens with probability at most $(1 - h/H)^{H/h} \leq 1/e$. Hence the optimal algorithm will cover v with probability at least $1 - 1/e$. By linearity of expectation, the lemma follows by taking the expectation over all nodes in the lowest level. \square

We now show that no online algorithm can achieve an expected profit greater than $\frac{2}{h}$. We first define two notations: *empty node* and *maximal empty node*. A node v is an *empty node* at time t , iff no descendants or ancestors are selected by the algorithm. v is a *maximal empty node* if it is empty and there is no other empty nodes in the path from v to the root. In other words, v is the highest empty node in the path from any node to the root that contains v .

At any given time, the total length of the *maximal empty nodes* is at most 1, since they are mutually disjoint. For any algorithm, if the job J_t arriving at time t is a node in the set of *maximal empty nodes*, we should definitely accept the job. Of course, the algorithm can also accept jobs which are not *maximal empty*. The follow lemma, however, claims that such jobs are very rare.

Lemma 3.2. *The total length of the non maximal empty nodes accepted by any algorithm is at most $1/h$.*

Proof. For any non *maximal empty node* v that the algorithm accepts at time t , its parent u as well as all u 's children must be empty before time t . Now after v is accepted, the other $h^2 - 1$ children of u become *maximal empty nodes* immediately. None of them can be accepted as non *maximal empty node* any more. Therefore, the total length of the non *maximal empty nodes* accepted is at most h^{-2} of the total length of the nodes in the entire tree, which is $h^{-2} \cdot h = 1/h$. \square

Theorem 3.3. *For a sequence of random jobs $\{J_t\}$ for $t \in [H/h]$ sampled from the tree, no online algorithm can achieve an expected profit larger than $2/h$.*

Proof. It is sufficient to bound the total profit taken by the algorithm on *maximal empty nodes* during the execution of the entire algorithm. Similar to the analysis of the optimal algorithm, consider any node v in the lowest level in the tree. In the chain $C(v)$, the path from v to the root, there is at most one *maximal empty node* by definition before any time t . Therefore, the probability that v is covered by a *maximal empty node* in time t is at most $1/H$. By simple union bound, the probability that v is covered by any *maximal empty node* during the algorithm is at most $1/H \cdot H/h = 1/h$. This implies the total expected profit any algorithm takes from *maximal empty nodes* is at most $1/h$, simply counting the lowest level nodes that are covered by them. The theorem follows by considering the additional profit from the non *maximal empty nodes* that were accepted. \square

Thus any online algorithm cannot have an expected profit of more than $2/h$, while the optimal solution has an expected profit of at least $1 - 1/e$. Thus any online algorithm has a competitive ratio of $\Omega(h) = \Omega(\frac{\log D}{\log \log D})$, and this completes the proof of Theorem 1.3.

4 Conclusions and Discussions

In this paper, we gave the first constant approximation for the laminar matroid secretary problem (LMSP). Babaioff et al. [4] showed that given a c -competitive algorithm for the matroid secretary problem for matroid \mathfrak{M} , one can obtain an $O(c)$ -competitive algorithm for the truncated matroid secretary problem. Can we extend this result to show that there exists a $O(c)$ -competitive algorithm for the intersection of \mathfrak{M} and any laminar matroid? It is not difficult to get an $O(1)$ -competitive algorithm for the ISSP problem when all jobs have unit size using the algorithm in [21]. We believe that there may be other interesting special cases for the ISSP problem that allow constant-competitive algorithms.

Acknowledgments: We thank Nitish Korula for his helpful discussion on the Laminar Matroid Secretary Problem and for his clear explanation on his past work [21].

References

- [1] Moshe Babaioff, Michael Dinitz, Anupam Gupta, Nicole Immorlica, and Kunal Talwar. Secretary problems: weights and discounts. In *SODA*, pages 1245–1254, 2009.
- [2] Moshe Babaioff, Jason D. Hartline, and Robert D. Kleinberg. Selling ad campaigns: online algorithms with cancellations. In *EC '09: Proceedings of the tenth ACM conference on Electronic commerce*, pages 61–70, New York, NY, USA, 2009. ACM.
- [3] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *APPROX '07/RANDOM '07: Proceedings of the 10th International Workshop on Approximation and the 11th International Workshop on Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 16–28, Berlin, Heidelberg, 2007. Springer-Verlag.
- [4] Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *SODA*, pages 434–443, 2007.
- [5] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph (Seffi) Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001.
- [6] Mohammad Hossein Bateni, MohammadTaghi Hajiaghayi, and Morteza Zadimoghaddam. The submodular secretary problem and its extensions. In *To appear in APPROX '10: 13th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems*, 2010.
- [7] Grigori Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *IPCO*, pages 182–196, 2007.
- [8] Chandra Chekuri and Jan Vondrák. Randomized pipage rounding for matroid polytopes and applications. *CoRR*, abs/0909.4348, 2009.
- [9] Florin Constantin, Jon Feldman, S. Muthukrishnan, and Martin Pál. An online mechanism for ad slot reservations with cancellations. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1265–1274, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.
- [10] Nikhil R. Devenur and Thomas P. Hayes. The adwords problem: online keyword matching with budgeted bidders under random permutations. In *EC '09: Proceedings of the tenth ACM conference on Electronic commerce*, pages 71–78, New York, NY, USA, 2009. ACM.

- [11] Nediialko B. Dimitrov and C. Greg Plaxton. Competitive weighted matching in transversal matroids. In *ICALP (1)*, pages 397–408, 2008.
- [12] E. B. Dynkin. Optimal choice of the stopping moment of a markov process. *Dokl.Akad.Nauk SSSR*, 150:238–240, 1963.
- [13] T.S. Ferguson. Who solved the secretary problem? *J. Statist. Sci.*, 4:282–289, 1989.
- [14] P. R. Freeman. The secretary problem and its extensions: a review. *Internat. Statist. Rev.*, 51(2):189–206, 1983.
- [15] M. Gardner. Mathematical games column. *Scientific American Feb., Mar.*, 35, 1960.
- [16] Gagan Goel and Aranyak Mehta. Online budgeted matching in random input models with applications to adwords. In *SODA '08: Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 982–991, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.
- [17] M. Golumbic. *Algorithmic graph theory and perfect graphs*. Academic Press, 1980.
- [18] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. *CoRR*, abs/1003.1517, 2010.
- [19] Mohammad Taghi Hajiaghayi, Robert D. Kleinberg, and David C. Parkes. Adaptive limited-supply online auctions. In *ACM Conference on Electronic Commerce*, pages 71–80, 2004.
- [20] Robert Kleinberg. A multiple-choice secretary algorithm with applications to online auctions. In *SODA '05: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 630–631, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics.
- [21] Nitish Korula and Martin Pál. Algorithms for secretary problems on graphs and hypergraphs. In *ICALP (2)*, pages 508–520, 2009.
- [22] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Ann. Oper. Res.*, 26(1-4):125–133, 1990.
- [23] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
- [24] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24. Springer-Verlag, 2003.

A Analysis Tools

Theorem A.1 ([23]). *Let X_1, X_2, \dots, X_n be n independent random variables such that $\Pr[X_i = 0] = \Pr[X_i = 1] = \frac{1}{2}$. Let $Y = \sum_{i=1}^n X_i$. Then, for any $\Delta > 0$, we have*

$$\Pr \left[\left| Y - \frac{n}{2} \right| \geq \Delta \right] \leq 2 \exp(-2\Delta^2/n).$$

Theorem A.2 ([23]). Let X_1, X_2, \dots, X_n be n independent random variables such that $\Pr[X_i = 0] = 1 - p_i$ and $\Pr[X_i = 1] = p_i$. Let $Y = \sum_{i=1}^n X_i$. Then we have that

$$\Pr \left[Y \geq (1 + \delta)\mu \right] \leq \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}} \right)^\mu.$$

In particular,

- for any $\delta \leq 2e - 1$, $\Pr \left[Y \geq (1 + \delta)\mu \right] \leq \exp(-\mu\delta^2/4)$.
- for any $\delta \geq 2e - 1$, $\Pr \left[Y \geq (1 + \delta)\mu \right] \leq 2^{-\mu(1+\delta)}$.