Scheduling to Minimize Power Consumption using Submodular Functions

by

Morteza Zadimoghaddam

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author ....
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by ....
Erik D. Demaine
Associate Professor
Thesis Supervisor

Accepted by ....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Scheduling to Minimize Power Consumption using Submodular Functions

by

Morteza Zadimoghaddam

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

We develop logarithmic approximation algorithms for extremely general formulations of multiprocessor multi-interval offline task scheduling to minimize power usage. Here each processor has an arbitrary specified power consumption to be turned on for each possible time interval, and each job has a specified list of time interval/processor pairs during which it could be scheduled. (A processor need not be in use for an entire interval it is turned on.) If there is a feasible schedule, our algorithm finds a feasible schedule with total power usage within an $O(\log n)$ factor of optimal, where n is the number of jobs. (Even in a simple setting with one processor, the problem is Set-Cover hard.) If not all jobs can be scheduled and each job has a specified value, then our algorithm finds a schedule of value at least $(1-\epsilon)Z$ and power usage within an $O(\log(1/\epsilon))$ factor of the optimal schedule of value at least Z, for any specified Z and $\epsilon > 0$. At the foundation of our work is a general framework for logarithmic approximation to maximizing any submodular function subject to budget constraints.

We also introduce the online version of this scheduling problem, and show its relation to the classical secretary problem. In order to obtain constant competitive algorithms for this online version, we study the secretary problem with submodular utility function. We present several constant competitive algorithms for the secretary problem with different kinds of utility functions.

Thesis Supervisor: Erik D. Demaine
Title: Associate Professor

Acknowledgments

I would like to thank my thesis supervisor, Professor Erik D. Demaine for his guidance and support during the years of my education. He was one of the main reasons I decided to come to MIT. I also learned many things from him about scientific methods of research. He is denitely the best teacher I have ever had.

I also want to thank Dr. MohammadTaghi Hajiaghayi for the insightful discussions we had. His strong discipline in research was very helpful for me to organize my work, and it really increased my productivity in research. Having friends like Mohammad Hossein Bateni and Seyed Amin Seyedi-Roshkhar, helped me tackle different problems during my studies. They always listened to my ideas and I am lucky to have them.

Without the support of my parents, I could not have accomplished my education. I cannot thank them enough. My success in Computer Olympiads and consequently entering the best engineering school in Iran is undoubtedly because of their efforts.

I especially want to thank my excellent wife for her encouragement. She was always supportive and understanding during my research and she was my main motivation. Without her help, I could not have resolved my problems and nished this thesis.

Contents

1 Introduction **6**

2 Scheduling with SubModular Maximization **11**

2.1 Submodular Maximization with Budget Constraints 11

2.2 Scheduling to Minimize Power in Parallel Machines 16

2.3 Prize-Collecting Scheduling Problem 21

3 Online setting and Secretary Problem **28**

3.1 Motivations and Preliminaries 28

3.2 The submodular secretary problem 34

3.2.1 Algorithms 34

3.2.2 Analysis 37

3.3 The submodular matroid secretary problem 43

3.4 Knapsack constraints 46

3.5 The subadditive secretary problem 49

3.5.1 Hardness result 50

3.5.2 Algorithm 52

3.6 Conclusions and further results 52

.1 Hardness Results 55

.2 Polynomial-Time Algorithm for Prize-Collecting One-interval Gap Minimization Problem 56

.3 Omitted proofs and theorems 58

Chapter 1

Introduction

Power management systems aim to reduce energy consumption while keeping the performance high. The motivations include battery conservation (as battery capacities continue to grow much slower than computational power) and reducing operating cost and environmental impact (both direct from energy consumption and indirect from cooling).

Processor energy usage A common approach in practice is to allow processors to enter a *sleep state*, which consumes less energy, when they are idle. All previous work assumes a simple model in which we pay zero energy during the sleep state (which makes approximation only harder), a unit energy rate during the awake state (by scaling), and a fixed restart cost α to exit the sleep state. Thus the total energy consumed is the sum over all awake intervals of α plus the length of the interval.

There are many settings where this simple model may not reflect reality, which we address in this paper:

1. When the processors are not identical: different processors do not necessarily consume energy at the same rate, so we cannot scale to have all processors use a unit rate.

2. When the energy consumption varies over the time: keeping a processor active for two intervals of the same length may not consume the same energy. One example is if we optimize energy *cost* instead of actual energy, which varies substantially in

energy markets over the course of a day. Another use for this generalization is if a processor is not available for some time slots, which we can represent by setting the cost of the processor to be infinity for these time slots.

3. When the energy consumption is an arbitrary function of its length: the growth in energy use might not be an affine function of the duration a processor is awake. For example, if a processor stays awake for a short time, it might not need to cool with a fan, saving energy, but the longer it stays awake, the faster the fan may need to run and the more energy consumed.

We allow the energy consumption of an awake interval to be an arbitrary function of the interval and the processor. We also allow the processor to be idle (but still consume energy) during such an interval. As a result, our algorithms automatically choose to combine multiple awake intervals (and the intervening sleep intervals) together into one awake interval if this change causes a net decrease in energy consumption.

Multi-interval task scheduling Most previous work assumes that each task has an arrival time, deadline, and processing time. The goal is then to find a schedule that executes all tasks by their deadlines and consumes the minimum energy (according to the notion above). This setup implicitly assumes identical processors.

We consider a generalization of this problem, called *multi-interval scheduling*, in which each task has a list of one or more time intervals during which it can execute, and the goal is to schedule each job into one of its time intervals. The list of time intervals can be different for each processor, for example, if the job needs specific resources held by different processors at different times.

Prize-collecting version All previous work assumes that all jobs can be scheduled using the current processors and available resources. This assumption is not necessarily satisfied in many practical situations, when jobs outweigh resources. In these cases, we must pick a subset of jobs to schedule.

We consider a general weighted *prize-collecting* version in which each job has a specified *value*. The bicriterion problem is then to find a schedule of value at least Z and minimum energy consumption subject to achieving this value.

Online Setting We introduce an interesting version of our problem which is closely related to the classical secretary problem. Assume that you have a set of tasks to do, and the processors arrive one by one. You want to pick a number of processors (according to your budget) to do the tasks, i.e. say you can pick k processors. We can see the processors as some secretaries, and we want to hire k secretaries to do the tasks. The secretaries arrive one at a time, and we have to decide immediately whether we want to hire the arrived secretary or not. At first we show how to characterize this problem using submodular functions in classical secretary problem. We later present constant competitive algorithms for this problem.

Our results We obtain in Section 2.2 an $O(\log n)$-approximation algorithm for scheduling n jobs to minimize power consumption. For the prize-collecting version, we obtain in Section 2.3 an $O(\log(1/\epsilon))$-approximation for scheduling jobs of total value at least $(1-\epsilon)Z$, comparing to an adversary required to schedule jobs of total value at least Z (assuming such a schedule exists), for any specified Z and $\epsilon > 0$. Both of our algorithms allow specifying an arbitrary processor energy usage for each possible interval on each processor, specifying an arbitrary set of candidate intervals on each processor for each job, and specifying an arbitrary value for each job.

These results are all best possible assuming P $\neq$ NP: we prove in Appendix .1 that even simple one-processor versions of these problems are Set-Cover hard.

Our approximation algorithms are based on a technique of independent interest. In Section 2.1, we introduce a general optimization problem, called *submodular maximization with budget constraints*. Many interesting optimization problems are special cases of this general problem, for example, Set Cover and Max Cover [33, 43] and the submodular maximization problems studied in [38, 39]. We obtain bicriteria $((1-\epsilon), O(\log 1/\epsilon))$-approximation factor for this general problem.

In Section 2.2, we show how our schedule-all-jobs problem can be formulated by a bipartite graph and its matchings. We define a matching function in bipartite graphs, and show that this function is submodular. Then the general technique of Section 2.1 solves the problem.

In Section 2.3, we show how the prize-collecting version of our scheduling problem can be formulated with a bipartite graph with weights on its nodes. Again we define a matching function in these weighted bipartite graphs, and with a more complicated proof, show that this function is also submodular. Again the general technique of Section 2.1 applies.

The general algorithm in Section 2.1 has many different and independent applications because submodular functions arise in a variety of applications. They can be seen as utility and cost functions of bidding auctions in game theory application [16]. These functions can be seen as covering functions which have many applications in different optimization problems: Set Cover functions, Edge Cut functions in graphs, etc.

Previous work The one-interval one-processor case of our problem with simple energy consumption function (α plus the interval length) remained an important and challenging open problem for several years: it was not even clear whether it was NP-hard.

The first main results for this problem considered the power-saving setting, which is easier with respect to approximation algorithms. Augustine, Irani, and Swamy [5] gave an online algorithm, which schedules jobs as they arrive without knowledge of future jobs, that achieves a competitive ratio of $3 + 2\sqrt{2}$. (The best lower bound for this problem is 2 [9, 31].)

For the offline version, Irani, Shukla, and Gupta [31] obtained a 3-approximation algorithm. Finally, Baptiste [9] solved the open problem: he developed a polynomial-time optimal algorithm based on an sophisticated dynamic programming approach. Demaine et al. [13] later generalized this result to also handle multiple processors.

The multi-interval case was considered only by Demaine et al. [13], after Baptiste mentioned the generalization during his talk at SODA 2006. They show that this problem is Set-Cover hard, so it does not have an $o(\log n)$-approximation. They also obtain a $1 + \frac{2}{3}\alpha$-approximation for the multi-interval multi-processor case, where α is the fixed restart cost. Note that α can be as large as n, so there is no general algorithm with approximation factor better than $\Theta(n)$ in the worst case (when α is around n).

However, both the Baptiste result [9] and Demaine et al. results [13] assume that processors enter the sleep state whenever they go idle, immediately incurring an α cost. For this reason, the problem can also be called *minimum-gap scheduling*. But this assumption seems unreasonable in practice: we can easily leave the processor awake during sufficiently short intervals in order to save energy. As mentioned above, the problem formulations considered in this paper fix this issue.

Chapter 2

Scheduling with SubModular Maximization

2.1 Submodular Maximization with Budget Constraints

Submodular functions arise in a variety of applications. They can represent different forms of functions in optimization problems. As a game theoretic example, both profit and budget functions in bid optimization problems are Set-Cover type functions (including the weighted version) which are special cases of submodular functions. As another application of these functions in online algorithms, we can mention the secretary problem in different models, the bipartite graph setting in [37], and the submodular functions setting in [1].

The authors of [39] studied the problem of submodular maximization under matroid and knapsack constraints (which can be seen as some kind of budget constraints), and they give the first constant factor approximation when the number of constraints is constant. We try to find solutions with more utility by relaxing the budget constraints. We give the first $(1-\epsilon)$-approximation for utility maximization with relaxing the budget constraint by $\log(1/\epsilon)$. In our model, we allow the cost of a subset of items be less than their sum. This way we can cover more general cases (nonlinear or submodular cost functions). All previous works on submodular functions assume that the cost function is linear. Therefore they can not cover many interesting optimization problems including the scheduling problems we are studying in this paper. Later we combine this result with other techniques to give optimal scheduling strategies for energy minimization problem with parallel machines.

Now we formulate the problem of submodular maximization with budget constraints.

Definition 1. *Let $U = \{a_1, a_2, \ldots, a_n\}$ be a set of n items. We are given a set $S = \{S_1, S_2, \ldots, S_m\} \subseteq 2^U$ specifying m allowable subsets of U that we can add to our solution. We are also given costs $C_1, C_2, \ldots, C_m$ for the subsets, where S_i costs C_i. Finally, we are given a* utility function *$F : 2^U \to \mathbb{R}$ defined on subsets of U. We require that F is* submodular *meaning that, for any two subsets A, B of U, we have*

$$F(A) + F(B) \geq F(A \cap B) + F(A \cup B).$$

We also require that F is monotone *(being a utility function) meaning that, for any subsets $A \subseteq B \subseteq U$, we have $F(A) \leq F(B)$.*

The problem is to choose a collection of the input subsets with reasonable cost and utility. The cost of a collection of subsets is the sum of their costs. The utility of these subsets is equal to the utility of their union. In particular, if we pick k subsets $S_1, S_2, \ldots, S_k$, their cost is $\Sigma_{i=1}^{k} C_i$ and their utility is equal to $F(\cup_{i=1}^{k} S_i)$. We are given a utility threshold x, and the problem is to find a collection with utility at least x having minimum possible cost.

Note that all previous work assumes that the set S of allowable subsets consists only of single-item subsets, namely $\{a_1\}, \{a_2\}, \ldots, \{a_n\}$. Equivalently, they assume that the cost of picking a subset of items is equal to the sum of the costs of the picked items (a linear cost function). By contrast, we allow that there be other subsets that we can pick with different costs, but that all such subsets are explicitly given in the input. The cost of a subset might be different from the sum of the costs of the items in that subset; in practice, we expect the cost to be less than the sum of the item costs.

We need the following result in the proof of the main algorithm of this section. Similar lemmas like this are proved in the literature of submodular functions. But we need to prove this more general lemma.

Lemma 2.1.1. *Let T be the union of k subsets $S_1, S_2, \ldots, S_k$, and S' be another arbitrary* *subset. For a monotone submodular function F defined on these subsets, we have that*

$$\sum_{j=1}^{k}[F(S' \cup S_j) - F(S')] \geq F(T) - F(S').$$

Proof. Let T' be the union of T and S'. We prove that $\sum_{j=1}^{k}[F(S' \cup S_j) - F(S')] \geq F(T') - F(S')$ which also implies the claim. Define subset S'_i be $(\cup_{j=1}^{i} S_j) \cup S'$ for any $0 \leq i \leq k$. We prove that

$$F(S' \cup S_i) - F(S') \geq F(S'_i) - F(S'_{i-1}).$$

Because F is submodular, we know that $F(A) + F(B) \geq F(A \cup B) + F(A \cap B)$ for any pair of subsets A and B. Let A be the set $S' \cup S_i$, and B be the set S'_{i-1}. Their union is S'_i, and their intersection is a superset of S'. So we have that

$$\begin{aligned} F(S' \cup S_i) + F(S'_{i-1}) &\geq F(S'_i) + F([S' \cup S_i] \cap [S'_{i-1}]) \\ &\geq F(S'_i) + F(S'). \end{aligned}$$

This completes the proof of the inequality, $F(S' \cup S_i) - F(S') \geq F(S'_i) - F(S'_{i-1})$. If we sum this inequality over all values of $1 \leq i \leq k$, we can conclude the claim:

$$\begin{aligned} \sum_{i=1}^{k} F(S' \cup S_i) - F(S') &\geq \sum_{i=1}^{k} F(S'_i) - F(S'_{i-1}) \\ &= F(T') - F(S') \\ &\geq F(T) - F(S'). \end{aligned}$$

□

Now we show how to find a collection with utility $(1 - \epsilon)x$ and cost $O(\log(1/\epsilon))$ times the optimum cost. Later we show how to find a subset with utility x in our particular application, scheduling with minimum energy consumption. It is also interesting that the following algorithm generalizes the well-known greedy algorithm for Set Cover in the sense that the Set-Cover type functions are special cases of monotone submodular functions. In order to use the following algorithm to solve the Set Cover problem with a logarithmic approximation factor (which is the best possible result for Set Cover), one just needs to set ϵ to some value less than 1 over the number of items in the Set-Cover instance.

Lemma 2.1.2. *If there exists a collection of subsets (optimal solution) with cost at most B and utility at least x, there is a polynomial time algorithm that can find a collection of subsets of cost at most $O(B\log(1/\epsilon))$, and utility at least $(1-\epsilon)x$ for any $0 < \epsilon < 1$.*

Proof. The algorithm is as follows. Start with set $S = \emptyset$. Iteratively, find the set S_i with maximum ratio of $\min\{x, F(S \cup S_i)\} - F(S)/C_i$ for $1 \le i \le m$ where $\min\{a, b\}$ is the minimum of a and b. In fact we are choosing the subset that maximizes the ratio of the increase in the utility function over the increase in the cost function, and we just care about the increments in our utility up to value x. If a subset increases our utility to some value more than x, we just take into account the difference between previous value of our utility and x, not the new value of our utility. We do this iteratively till our utility is at least $(1-\epsilon)x$.

We prove that the cost of our solution is $O(B\log(1/\epsilon))$. Assume that we pick some subsets like $S'_1, S'_2, \ldots, S'_{k'}$ respectively. We define the subsets of our solution into $\log(1/\epsilon)$ phases. Phase $1 \le i \le \log(1/\epsilon)$, ends when the utility of our solution reaches $(1-1/2^i)x$, and starts when the previous phase ends. In each phase, we pick a sequence of the k' subsets $S'_1, S'_2, \ldots, S'_{k'}$. We prove that the cost of each phase is $O(B)$, and therefore the total cost is $O(B\log(1/\epsilon))$ because there are $\log(1/\epsilon)$ phases.

Let S'_{a_i} be the last subset we pick in phase i. So $F(\cup_{j=1}^{a_i} S'_j)$ is our utility at the end of phase i, and is at least $(1-1/2^i)x$, and $F(\cup_{j=1}^{a_i-1} S'_j)$ is less than $(1-1/2^i)x$. So we pick subsets $S'_{a_{i-1}+1}, S'_{a_{i-1}+2}, \ldots, S'_{a_i}$ in phase i. We prove that the ratio of utility per cost of all subsets inserted in phase i is at least $\frac{x/2^i}{B}$. Assume that we are in phase i, and we want to pick another set (phase i is not finished yet). Let S' be our current set (the union of all subsets we picked up to now). $F(S')$ is less than $(1-1/2^i)x$. We also know that there exists a solution (optimal solution) with cost B and utility x. Without loss of generality, we assume that this solution consists of k subsets $S_1, S_2, \ldots, S_k$. Let T be the union of these

k subsets. Using lemma 2.1.1, we have that

$$\sum_{j=1}^{k}[F(S' \cup S_j) - F(S')] \geq F(T) - F(S') > x/2^i.$$

If $F(S' \cup S_j)$ is at most x for any $1 \leq j \leq k$, we can say that

$$\sum_{j=1}^{k}[\min\{x, F(S' \cup S_j)\} - F(S')] =$$

$$\sum_{j=1}^{k}[F(S' \cup S_j) - F(S')] \geq F(T) - F(S') > x/2^i.$$

Otherwise there is some j for which $F(S' \cup S_j)$ is more than x. So $\min\{x, F(S' \cup S_j)\} - F(S')$ is at least $x/2^i$ because $F(S')$ is less than $(1 - 1/2^i)x$. So in both cases we can claim the above inequality. We also know that

$$\sum_{j=1}^{k} C_j \leq B,$$

where C_j is the cost of set S_j. In every iteration, we find the subset with the maximum ratio of utility per cost (the increase in utility per the cost of the subset). Note that we also consider these k subsets $S_1, S_2, \ldots, S_k$ as candidates. So the ratio of the subset we find in each iteration is not less than the ratio of each of these k subsets. The ratio of subset S_j is $[\min\{x, F(S' \cup S_j)\} - F(S')]/C_j$. The maximum ratio of these k subsets is at least the sum of the nominators of the k ratios of these sets over the sum of their denominators which is

$$\frac{\sum_{j=1}^{k}[\min\{x, F(S' \cup S_j)\} - F(S')]}{\sum_{j=1}^{k} C_j} > \frac{x}{2^i B}.$$

So in phase i, the utility per cost ratio of each subset we add is at least $\frac{x}{2^i B}$. Now we can bound the cost of this phase. We pick subsets $S'_{a_{i-1}+1}$, $S'_{a_{i-1}+2}$, $\ldots$, S'_{a_i} in phase i. Let u_0 be our utility at the beginning of phase i. In other words, u_0 is $F(\cup_{j=1}^{a_{i-1}} S'_j)$. Assume we pick l subsets in this phase, i.e., l is $a_i - a_{i-1}$. Let u_j be our utility after inserting jth subset in this phase where $1 \leq j \leq l$. Note that we stop the algorithm when our utility reaches $(1-\epsilon)x$. So our utility after adding the first $l-1$ subsets is less than x. Our utility at the end of this phase, u_l might be more than x. For any $1 \leq j \leq l-1$, the utility per cost ratio is $u_j - u_{j-1}$ divided by the cost of the jth subset. For the last subset, the ratio is $\min\{x, u_l\} - u_{l-1}$ divided by the cost of the last subset of this phase. According to the definition of the phases, our utility at the beginning of this phase, u_0 is at least $(1-1/2^{i-1})x$. So we have that

$$\min\{x, u_l\} - u_{l-1} + \sum_{j=1}^{l-1} u_j - u_{j-1} =$$

$$\min\{x, u_l\} - u_0 \leq x - (1 - 1/2^{i-1})x = x/2^{i-1}.$$

On the other hand, we know that the utility per cost ratio of all these subsets is at least $\frac{x}{2^i B}$. Therefore the total cost of this phase is at most

$$\frac{[\min\{x, u_l\} - u_{l-1} + \sum_{j=1}^{l-1} u_j - u_{j-1}]}{x/2^i B} \leq \frac{x/2^{i-1}}{x/2^i B},$$

which is at most $2B$. So the total cost in all phases is not more than $\log(1/\epsilon) \cdot 2B$. □

2.2 Scheduling to Minimize Power in Parallel Machines

We proved how to find almost optimal solutions with reasonable cost when the utility functions are submodular. Here we show how the scheduling problem can be formulated as an optimization problem with submodular utility functions.

First we explain the power minimization scheduling problem in more detail.

Definition 2. *There are p processors $P_1, P_2, \ldots, P_p$ and n jobs $j_1, j_2, \ldots, j_n$. Each processor has an energy cost $c(I)$ for every possible awake interval I. Each job j_i has a unit processing time (which is equivalent to allowing pre-emption), and set T_i of valid time slot/processor pairs. (Unlike previous work, T_i does not necessarily form a single interval, and it can have different valid time slots for different processors.) A* feasible schedule *consists of a set of awake time intervals for each processor, and an assignment of each job to an integer time and one of the processors, such that jobs are scheduled only during awake time slots (and during valid choices according to T_i) and no two jobs are scheduled at the* *same time on the same processor. The* cost *of such a schedule is the sum of the energy costs of the awake intervals of all processors.*

In the simple case which has been studied in [9, 13], it is assumed that the cost of an interval is a fixed amount of energy (restart cost α) plus the size of the interval. We assume a very general case in which the cost of keeping a machine active during an interval is a function of that machine, and the interval. For instance, it might take more energy to keep some machines active comparing to other machines, or some time intervals might have more cost. So there is a cost associated with every pair of a time interval and a machine. These costs might be explicitly given in the input, or can be accessed through a query oracle, i.e., when the number of possible intervals are not polynomial.

If we pick a collection of active intervals for each machine at first, we can then find and schedule the maximum number of possible jobs that can be all together scheduled in the active time slots without collision using the maximum bipartite matching algorithms. So the problem is to find a set of active intervals with low cost such that all jobs can be done during them.

Let U be the set of all time slots in different machines. In fact for every unit of time, we put p copies in U, because at each unit of time, we can schedule p jobs in different machines, so each of these p units is associated with one of the machines. We can define a function F over all subsets of U as follows. For every subset of time slot/processor pairs like $S \subset U$, $F(S)$ is the maximum number of jobs that can be scheduled in time slot/processor pairs of S. Our scheduling problem can be formulated as follows. We want to find a collection of time intervals $I_1, I_2, \ldots, I_k$ with minimum cost and $F(\cup_{i=1}^{k} I_i) = n$ (this means that all n jobs can be scheduled in these time intervals). Note that each I_i is a pair of a machine and a time interval, i.e., I_1 might be $(P_2, [3, 6])$ which represents the time interval $[3, 6]$ in machine P_2. The cost of each I_i can be accessed from the input or a query oracle. The cost of this collection of intervals is the sum of the costs of the intervals. We just need to prove that function F is monotone and submodular. The monotonicity comes from its definition. The submodularity proof is involved, and needs some graph theoretic Lemmas. Now we can present our main result for this broad class of scheduling problems.

Theorem 2.2.1. *If there is a schedule with cost B which schedules all jobs, there is a polynomial time algorithm which schedules all jobs with cost $O(B \log n)$.*

Proof. We are looking for a collection of intervals with utility at least n, and cost $O(B \log n)$. Lemma 2.2.2 below states that F (defined above) is submodular. Using the algorithm of Lemma 2.1.2, we can find a collection of time intervals with utility at least $(1 - \epsilon)n$ and cost at most $O(B \log (1/\epsilon))$ because there exists a collection of time intervals (schedule) with utility n (schedules all n jobs) and cost B. Let ϵ be $1/(n+1)$. The cost of the result of our algorithm is $O(B \log (n+1))$, and its utility is at least $(1 - 1/(n+1))n > n - 1$. Because the utility function F always take integer values, the utility of our result is also n. So we can find a collection of time intervals that all jobs can be scheduled in them. We just need to run the maximum bipartite matching algorithm to find the appropriate schedule. This means that our algorithm also schedules all jobs, and has cost $O(B \log (n+1))$. □

There is another definition of submodular functions that is equivalent to the one we presented in the previous section. We will use this new definition in the following lemma.

Definition 3. *A function F is submodular if for every pair of subsets $A \subset B$, and an element z, we have:*

$$F(A \cup \{z\}) - F(A) \geq F(B \cup \{z\}) - F(B)$$

Now we just need to show that F is submodular. We can look at this function as the maximum matching function of subgraphs of a bipartite graph. Construct graph G as follows. Consider time slots of U as the vertices of one part of G named X. Put n vertices representing the jobs in the other side of G named Y. Note that the time slots of U are actually pairs of a time unit and a processor. Put an edge between one vertex of X and a vertex of Y if the associated job can be scheduled in that time slot (which is a pair of a time unit and a processor), i.e., if the job can be done in that processor and in that time unit. Now every subset of $S \subset X$ is a subset of time slots, and $F(S)$ is the maximum number of jobs that can be executed in S. So $F(S)$ is in fact the maximum cardinality matching that saturates only vertices of S in part X (it can saturate any subset of vertices in Y). A vertex is saturated by a matching if one of its incident edges participates in the matching. Now we can present this submodularity Lemma in this graph model.

Lemma 2.2.2. *Given a bipartite graph G with parts X and Y. For every subset $S \subset X$, define $F(S)$ to be the maximum cardinality matching that saturates only vertices of S in part X. The function F is submodular.*

Proof. We just need to prove that, for two subsets $A \subset B \subset X$ and a vertex v in X, the following inequality holds:

$$F(A \cup \{v\}) - F(A) \geq F(B \cup \{v\}) - F(B).$$

Let M_1 and M_2 be two maximum matchings that saturate only vertices of A and B respectively. Note that there might be more than just one maximum matching in each case (for sets A and B). We first prove that there are two such maximum matchings that M_1 is a subset of M_2, i.e., all edges in matching M_1 also are in matching M_2. This can be proved using the fact that $A \subset B$ as follows.

Consider two maximum matchings M_1 and M_2 with the maximum number of edges in common. The edges of $M_1 \Delta M_2$ form a bipartite graph H where $A_1 \Delta A_2$ is $A_1 \cup A_2 - A_1 \cap A_2$ for every pair of sets A_1 and A_2. Because it is a disjoint union of two matchings, every vertex in H has degree 0, 1 or 2. So H is a union of some paths and cycles. We first prove that there is no cycle in H. We prove this by contradiction. Let C be a cycle in H. The edges of C are alternatively in M_1 and M_2. All vertices of this cycle are either in part Y of the graph or in $A \subset X$. Now consider matching $M_1' = M_1 \Delta C$ instead of M_1. It also saturates only some vertices of A in part X, and has the same size of M_1. Therefore M_1' is also a maximum matching with the desired property, and has more edges in common with M_2. This contradiction implies that there is no cycle in H.

Now we study the paths in H. At first we prove that there is no path in H with even number of edges. Again we prove this by contradiction. The edges of a path in H alternate between matchings M_1 and M_2. Let P be a path in H with even number of edges. This path has equal number of edges from M_1 and M_2. Now if we take $M_2' = M_2 \Delta P$ instead of M_2, we have a new matching with the same number of edges, and it has more edges in common with M_1. This contradiction shows that there is no even path in H.

Finally we prove that all other paths in H are just some single edges from M_2, and therefore there is no edge from M_1 in H. This completes the proof of the claim that M_1 is a subset of M_2. Again assume that there is a path P' with odd and more than one number of edges. Let $e_1, e_2, \ldots, e_{2l+1}$ are the edges of P'. The edges with even index are in M_1, the rest of the edges are in M_2 otherwise $M_2'' = M_2 \Delta P'$ would be a matching for set B which has more edges than M_2 (this is a contradiction). Because P' is an odd path, we can assume that it starts from part Y, and ends in part X without loss of generality. Now if we delete edges $e_2, e_4, \ldots, e_{2l}$ from M_1, and insert edges $e_1, e_3, \ldots, e_{2l-1}$ instead, we reach a new matching M_1'. This matching uses a new vertex from Y, but the set of saturated vertices of X in matching M_1' is the same as the ones in M_1. These two matchings also have the same size. But M_1' has more edges in common with M_2. This is also contradiction, and implies that there is no such a path in H. So M_1 is a subset of M_2.

We are ready to prove the main claim of this theorem. Note that we have to prove this inequality:

$$F(A \cup \{v\}) - F(A) \geq F(B \cup \{v\}) - F(B).$$

We should prove that if adding v to B increases its maximum matching, it also increases the maximum matching of A. Let M_3 be the maximum matching of $B \cup \{v\}$. Let H' be the subgraph of G that contains the edges of $M_2 \Delta M_3$. Because M_3 has more edges than M_2, there exists a path Q in H' that has more edges from M_3 than M_2 (cycles have the same number of edges from both matchings). The vertex v should be in path Q, otherwise we could have used the path Q to find a matching in B greater than M_2, i.e., matchings $M_2 \Delta Q$ could be a greater matching for set B in that case which is a contradiction.

The degree of v in H is 1, because it does not participate in matching M_2, does participate in M_3. So v can be seen as the starting vertex of path Q. Let $e_1, e_2, \ldots, e_{2l'+1}$ be the edges of Q. The edges $e_2, e_4, \ldots, e_{2l'}$ are in M_2, and some of them might be in M_1. Let $0 \leq i \leq l'$ be the maximum integer number for which all edges $e_2, e_4, \ldots, e_{2i}$ are in M_1. If e_2 is not in M_1, we set i to be 0. If we remove edges $e_2, e_4, \ldots, e_{2i}$ from M_1, and insert edges $e_1, e_3, \ldots, e_{2i+1}$ instead, we reach a matching for set $A \cup \{v\}$ with more edges than

M_1. So adding v to A increases the size of its maximum matching.

Now the only thing we should check is that edges $e_1, e_3, \ldots, e_{2i+1}$ does not intersect with other edges of M_1. Let $v = v_0, v_1, v_2, \ldots, v_{2l'+1}$ be the vertices of Q. Because we remove edges $e_2, e_4, \ldots, e_{2i}$ from M_1, we do not have to be worried about inserting the first i edges $e_1, e_3, \ldots, e_{2i-1}$. The last edge we add is $e_{2i+1} = (v_{2i}, v_{2i+1})$. If v_{2i+1} is not saturated in M_1, there will be no intersection. So we just need to prove that v_{2i+1} is not saturated in M_1.

If i is equal to l', the vertex $v_{2i+1} = v_{2l'+1}$ is not saturated in M_2. Because M_1 is a subset of M_2, the vertex v_{2i+1} is also not saturated in M_1.

If i is less than l', the vertex v_{2i+1} is saturated in M_2 by edge e_{2i+2}. Assume v_{2i+1} is saturated in M_1 by an edge e'. The edge e' should be also in M_2 because all edges of M_1 are in M_2. The edge e' intersects with e_{2i+2}, so e' has to be equal to e_{2i+2}. The definition of value i implies that e_{2i+2} should not be in M_1 (we pick the maximum i with the above property). This contradiction shows that the vertex v_{2i+1} is not saturated in M_1, and therefore we get a greater matching in $A \cup \{v\}$ using the changes in M_1. $\square$

2.3 Prize-Collecting Scheduling Problem

We introduce the prize-collecting version of the scheduling problems. All previous work assumes that we can schedule all jobs using the existing processors. There are many cases that we can not execute all jobs, and we have to find a subset of jobs to schedule using low energy. There might be priorities among the jobs, i.e., there might be more important jobs to do. We formalize this problem as follows.

As before, there are P processors and n jobs. Each job j_i has a set T_i of time slot/processor pairs during which it can execute. Each job j_i also has a value z_i. We want to schedule a subset of jobs S with value at least a given threshold Z, and with minimum possible cost. The *value* of set S is the sum of its members' values, and it should be at least Z. Following we prove that there is a polynomial-time algorithm which finds a schedule with value at least $(1-\epsilon)Z$ and cost at most $O(\log(1/\epsilon))$ times the optimum solution. Note that the optimum solution has value at least Z.

Later in this section, we show how to find a solution with utility at least Z, and logarithmic approximation on the energy consumption (cost).

Theorem 2.3.1. *If there is an schedule for the prize-collecting scheduling problem with value at least Z and cost B, there is an algorithm which finds a schedule with value at least $(1-\epsilon)Z$ and cost at most $O(B \log(1/\epsilon))$.*

Proof. Like the simple version of the scheduling problem, we construct a bipartite graph, and relate it to our algorithm in Lemma 2.1.2. The difference is that the bipartite graph here has some weights (job values) on the vertices of one of its parts. And it makes it more complicated to prove that the corresponding utility function is submodular. At first we explain the construction of the bipartite graph, and show how to reduce our problem to it. Then we use Lemma 2.3.2 to prove that the utility function is submodular.

We make graph G with parts X and Y. The vertices of part X represent the time slot/processor pairs. So for each pair of a time unit in a processor, we have a vertex in X. On the other part, Y, we have the n jobs. The edges connect jobs to their sets of time slot/processor pairs, i.e., job j_i has edges only to time slot/processors pairs in T_i, so a job might have edges to different time units in different processors. The only difference is that each edge has a weight in this graph. Each edge connects a job to a time slot/processor pair, the weight of an edge is the value of its job. Every schedule is actually a matching in this bipartite graph, and the value of a matching is the sum of the values of the jobs that are scheduled in it. This is why we set the weight of an edge to the value of its job.

The problem again is to find a collection of time intervals for each processor, and schedule a subset of jobs in those intervals such that the value of this subset is close to Z, and the cost of the schedule is low. If we have a subset of intervals, we can find the best subset of jobs to schedule in it. This can be done using the maximum weighted bipartite matching. The only thing we have to prove is that the utility function associated with this weighted bipartite graph is submodular. This is also proved in Lemma 2.3.2. □

Lemma 2.3.2. *Given a bipartite graph G with parts X and Y. Every vertex in Y has a value. For every subset $S \subset X$, define $F(S)$ be the maximum weighted matching that saturates only vertices of S in part X. The weight of a matching is the sum of the values of* *the vertices saturated by this matching in Y. The function F is submodular.*

Proof. Let A and B be two subsets of X such that $A \subseteq B$. Let v be a vertex in X. We have to prove that:

$$F(A \cup \{v\}) - F(A) \geq F(B \cup \{v\}) - F(B)$$

Let M_1 and M_2 be two maximum weighted matchings that saturate only vertices of A and B in X respectively. Among all options we have, we choose two matchings M_1 and M_2 that have the maximum number of edges in common. We prove that every saturated vertex in M_1 is also saturated in M_2 (note that we can not prove that every edge in M_1 is also in M_2). We prove this by contradiction.

The saturated vertices in M_1 are either in set A or in set Y. At first, let v' be a vertex in A that is saturated in M_1, and not saturated in M_2. Let u' be its match in part Y (v' is a time slot/processor pair, and u' is a job). The vertex u' is saturated in M_2 otherwise we could add edge (v', u') to matching M_2, and get a matching with greater value instead of M_2. So u' is matched with a vertex of B like v'' in matching M_2. If we delete the edge (v'', u') from matching M_2, and use edge (v', u') instead, the value of our matching remains unchanged, but we get a maximum matching instead of M_2 that has more edges in common with M_1 which is contradiction. So any vertex in X that is saturated in M_1 is also saturated in M_2.

The other case is when there is vertex in Y like u' that is saturated in M_1, and not saturated in M_2. The vertex u' is matched with vertex $w \in A$ in matching M_1. Again if w is not saturated in M_2, we can insert edge (w, u') to M_2, and get a matching with greater value. So w should be saturated in M_2. Let u'' be the vertex matched with w in M_2. For now assume that u'' is not saturated in M_1. Note that u' and u'' are some jobs with some values, and w is a time slot/processor pair. If the values of jobs u' and u'' are different, we can switch the edges in one of the matchings M_1 or M_2, and get a better matching. For example, if the value of u' is greater than u'', we can use edge (w, u') instead of (w, u'') in matching M_2, and increase the value of M_2. If the value of u'' is greater than u', we can use edge (w, u'') instead of (w, u') in matching M_1, and increase the value of M_1. So the value of u' and u'' are the same, we again can use (w, u'') instead of (w, u') in matching

M_1, and get a matching with the same value but more edges in common with M_2. This is a contradiction. So u'' should be saturated in M_1 as well, but if we continue this process we find a path P starting with vertex u'. The edges of this path alternate between M_1 and M_2. Path P starts with an edge in M_1, so it can not end with another edge in M_1 otherwise we can take $M_2 \Delta P$ instead of M_2 to increase the size of our matching for set B which is a contradiction. So path P starts with vertex u' and an edge in M_1, and ends with an edge in M_2. We have the same situation as above, and we can reach the contradiction similarly (just take the last vertex of the path as u''). So we can say that all saturated vertices in M_1 are also saturated in M_2.

Despite the unweighted graphs, $F(A \cup \{v\}) - F(A)$ and $F(B \cup \{v\}) - F(B)$ might take values other than zero or one.

If M_2 is also a maximum matching for set $B \cup \{v\}$, we do not need to prove anything. Because $F(B \cup \{v\})$ would be equal to $F(B)$ in that case, and we know that $F(A \cup \{v\})$ is always at least $F(A)$. So assume that M_2' is a maximum matching for set $B \cup \{v\}$ that has the maximum number of edges in common with M_2, and its value is more than the value of M_2. Consider the graph H that consists of edges $M_2' \Delta M_2$. We know that H is union of some paths and cycles. We can prove that H is only a path that starts with vertex v. In fact, if there exists a connected component like C in H that does not include vertex v, we can take matching $M_2' \Delta C$ which is a matching for set $B \cup \{v\}$ with more edges in common with M_2. Note that the value of matching $M_2' \Delta C$ can not be less than the value of M_2' otherwise we can use the matching $M_2 \Delta C$ for set B instead of matching M_2, and get a greater value which is a contradiction (M_2 is a maximum value matching for set B).

So graph H has only one connected component that includes vertex v. Because vertex v does not participate in matching M_2, its degree in graph H should be at most 1. We also know that v is saturated in M_2', so its degree is one in H. Therefore, graph H is only a path P. This path starts with vertex v, and one of the edges in M_2'. The edges of P are alternatively in M_2' and M_2. If P ends with an edge in M_2, the set of jobs that these two matchings, M_2 and M_2', schedule are the same. So their values would be also the same, and $F(B \cup \{v\})$ would be equal to $F(B)$ which is a contradiction. So path P has odd number of edges. Let $e_1, e_2, \ldots, e_{2l+1}$ be the edges of P, and $v = v_0, v_1, v_2, \ldots, v_{2l+1}$ be its vertices.

Note that $v_0, v_2, \ldots, v_{2l}$ are some time slot/processor pairs, and the other vertices are some jobs with some values. Edges $e_2, e_4, \ldots, e_{2l}$ are in M_2, and the rest are in M_2'.

The only job that is scheduled in M_2', and not scheduled in M_2 is the job associated with vertex v_{2l+1}. Let x_i be the value of the vertex v_{2i+1} for any $0 \le i \le l$. So $F(B \cup \{v\}) - F(B)$ is equal to x_l. We prove that x_l is not greater than any x_i for $0 \le i < l$ by contradiction. Assume x_i is less than x_l for some $i < l$. We could change the matching M_2 in the following way, and get a matching with greater value for set B. We could delete edges $e_{2i+2}, e_{2i+4}, \ldots, e_{2l}$, and insert edges $e_{2i+3}, e_{2i+5}, \ldots, 2_{2l+1}$ instead. This way we schedule job v_{2l+1} instead of job v_{2i+1}, and increase our value by $x_l - x_i$. Because M_2 is a maximum matching for set B, this is a contradiction so x_l should be the minimum of all x_is.

If all edges $e_2, e_4, \ldots, e_{2l}$ are also in matching M_1, we can use path P to find a matching for set $A \cup \{v\}$ with value x_l more than the value of M_1. We can take matching $M_1 \Delta P$ for set $A \cup \{v\}$. Because vertex v_{2l+1} is not saturated in M_2, it is also not saturated in M_1. So $M_1 \Delta P$ is a matching for set $A \cup \{v\}$. We conclude that $F(A \cup \{v\}) - F(A)$ is at least x_l which is equal to $F(B \cup \{v\}) - F(B)$. This completes the proof for this case.

In the other case, there are some edges among $e_2, e_4, \ldots, e_{2l}$ that are not in M_1. Let e_{2j} be the first edge among these edges that is not in M_1. So all edges $e_2, e_4, \ldots, e_{2j-2}$ are in both M_1 and M_2. Note that e_{2j} matches job v_{2j-1} with the time slot/processor pair v_{2j} in matching M_2. If job v_{2j-1} is not used (saturated) in matching M_1, we can find a matching as follows for set $A \cup \{v\}$. We can delete edges $e_2, e_4, \ldots, e_{2j-2}$ from M_1, and insert edges e_1, e_3, $\ldots$, e_{2j-1} instead. This way we schedule job x_{2j-1} in addition to all other jobs that are scheduled in M_1. So the value of $F(A \cup \{v\})$ is at least x_{j-1} (the value of job x_{2j-1}) more than $F(A)$. We conclude that $F(A \cup \{v\}) - F(A) = x_{j-1}$ is at least $F(B \cup \{v\}) - F(B) = x_l$.

Finally we consider the case that v_{2j-1} is also saturated in M_1 using some edge e other than e_{2j}. Edges e and e_{2j} are in M_1 and M_2 respectively, and vertex v_{2j-1} is their common endpoint. So these two edges should come in the same connected component in the graph $M_1 \Delta M_2$. We proved that all connected components of $M_1 \Delta M_2$ are paths with odd number of edges that start and end with edges in M_2. Let Q be the path that contains edges e and e_{2j}. This path contains edges e_1', e_2', $\ldots$, $e_i' = e_{2j}$, $e_{i+1}' = e$, e_{i+2}', $\ldots$, $e_{2l'+1}'$. The last edge of this path, $e'_{2l'+1}$ matches a job v' with a time slot/processor pair. Let x' be the value of v'. Vertex v' is not scheduled in matching M_1. At first we prove that x' is at least x_l (the value of job v_{2l+1}). Then we show how to find a matching for set $A \cup \{v\}$ with value at least x' more than the value of M_1.

If x' is less than x_l, we can find a matching with greater value for set B instead of M_2. Delete edges $e'_i = e_{2j}$, e'_{i+2}, e'_{i+4}, $\ldots$, $e'_{2l'+1}$, and also edges e_{2j+2}, e_{2j+4}, $\ldots$, e_{2l} from M_2, and insert edges $e'_{i+1} = e$, e'_{i+3}, $\ldots$, $e'_{2l'}$, and edges e_{2j+1}, e_{2j+3}, $\ldots$, e_{2l+1} to M_2 instead of the deleted edges. In the new matching, job v' with value x' is not saturated any more, but the vertex v_{2l+1} with value x_l is saturated. So the value of the new matching is $x_l - x' > 0$ more than the value of M_2 which is a contradiction. So x' is at least x_l.

Now we prove that there is a matching for set $A \cup \{v\}$ with value x' more than the value of M_1. We can find this matching as follows. Delete edges $e'_{i+1} = e$, e'_{i+3}, $\ldots$, $e'_{2l'}$, and edges e_2, e_4, $\ldots$, e_{2j-2}, and insert edges e'_{i+2}, e'_{i+4}, $\ldots$, $e'_{2l'+1}$, and edges e_1, e_3, $\ldots$, e_{2j-1}. This way we schedule job v' with value x' in addition to all other jobs that are scheduled in M_1. So we find a matching for set $A \cup \{v\}$ with value x' more than the value of M_1.

So $F(A \cup \{v\}) - F(A)$ is at least x'. We also know that $F(B \cup \{v\}) - F(B)$ is equal to x_l. Because x' is at least x_l, the proof is complete. $\square$

Now we are ready to represent our algorithm which finds an optimal solution (with respect to values).

Theorem 2.3.3. *If there is an schedule for the prize-collecting scheduling problem with value at least Z and cost B, there is an algorithm which finds a schedule with value at least Z and cost at most $O([\log n + \log \Delta]B)$ where δ is the ratio of the maximum value over the minimum value of all n jobs.*

Proof. Let $v_{\max}$ and $v_{\min}$ be the maximum and minimum value among all n jobs respectively. We know that Z can not be more than $n \cdot v_{max}$. Define ϵ to be $\frac{v_{\min}}{n \cdot v_{max}} = \frac{1}{n\Delta}$. Using Theorem 2.3.1, we can find a solution with value at least $(1-\epsilon)Z$ and cost at most $O(B\log(n\Delta)) = O([\log n + \log \Delta]B)$. Let S' be this solution. If the value of S' is at least Z, we exit and return this set as our solution. Otherwise we do the following. Note that we just need ϵZ more value to reach the threshold Z, and ϵZ is at most $v_{\min}$. So we just need to insert another interval which increases our value by at least $v_{\min}$. In the proof of Lemma 2.3.2, we proved that the value of $F(B \cup \{v\}) - F(B)$ is either zero or equal to the value of some jobs (in the proof it was x_l the value of vertex v_{2l+1}). So if we add an interval the value of set is either unchanged or increased by at least $v_{\min}$. So among all intervals with cost at most B, we choose one of them that increase our value by at least $v_{\min}$. At first note that this insertion reaches our value to Z, and our cost would be still $O([\log n + \log \Delta]B)$.

We now prove that there exists such an interval. Note that the optimum solution consists of some intervals $S_1, S_2, \ldots, S_k$. The union of these intervals, T has value $F(T)$ which is at least Z. So $F(T)$ is greater than the value of our solution $F(S')$. Using Lemma 2.1.1, $F(S' \cup S_i) - F(S')$ should be positive for some $1 \leq i \leq k$. We also know that the cost of this set is not more than B because the cost of the optimum solution is not more than B. So there exists a time interval (a set like S_i) that solves our problem with additional cost at most B. We also can find it by a simple search among all time intervals. $\square$

Note that in the simple case studied in the literature, the values are all identical, and Δ is equal to 1.

Chapter 3

Online setting and Secretary Problem

3.1 Motivations and Preliminaries

Online auction is an essence of many modern markets, particularly networked markets, in which information about goods, agents, and outcomes is revealed over a period of time, and the agents must make irrevocable decisions without knowing future information. Optimal stopping theory is a powerful tool for analyzing such scenarios which generally require optimizing an objective function over the space of stopping rules for an allocation process under uncertainty. Combining optimal stopping theory with game theory allows us to model the actions of rational agents applying competing stopping rules in an online market. This first has been considered by Hajiaghayi et al. [27] which initiated several follow-up papers (see e.g. [6–8, 26, 30, 36]).

Perhaps the most classic problem of stopping theory is the well-known *secretary problem*. Imagine that you manage a company, and you want to hire a secretary from a pool of n applicants. You are very keen on hiring only the best and brightest. Unfortunately, you cannot tell how good a secretary is until you interview him, and you must make an irrevocable decision whether or not to make an offer at the time of the interview. The problem is to design a strategy which maximizes the probability of hiring the most qualified secretary. It is well-known since 1963 [14] that the optimal policy is to interview the first $t-1$ applicants, then hire the next one whose quality exceeds that of the first $t-1$ applicants, where

t is defined by $\sum_{j=t+1}^{n} \frac{1}{j-1} \leq 1 < \sum_{j=t}^{n} \frac{1}{j-1}$; as $n \to \infty$, the probability of hiring the best applicant approaches $1/e$, as does the ratio t/n. Note that a solution to the secretary problem immediately yields an algorithm for a slightly different objective function optimizing the expected value of the chosen element. Subsequent papers have extended the problem by varying the objective function, varying the information available to the decision-maker, and so on, see e.g., [3,24,46,48].

An important generalization of the secretary problem with several applications (see e.g., a survey by Babaioff et al. [7]) is called the *multiple-choice secretary problem* in which the interviewer is allowed to hire up to $k \geq 1$ applicants in order to maximize performance of the secretarial group based on their overlapping skills (or the joint utility of selected items in a more general setting). More formally, assuming applicants of a set $S = \{a_1, a_2, \cdots, a_n\}$ (applicant pool) arriving in a uniformly random order, the goal is to select a set of at most k applicants in order to maximize a profit function $f : 2^S \to R$. We assume f is non-negative throughout this paper. For example, when $f(T)$ is the maximum individual value [22,23], or when $f(T)$ is the sum of the individual values in T [36], the problem has been considered thoroughly in the literature. Indeed, both of these cases are special monotone non-negative submodular functions that we consider in this paper. A function $f : 2^S \to R$ is called *submodular* if and only if $\forall A, B \subseteq S : f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$. An equivalent characterization is that the marginal profit of each item should be non-increasing, i.e., $f(A \cup \{a\}) - f(A) \leq f(B \cup \{a\}) - f(B)$ if $B \subseteq A \subseteq S$ and $a \in S \setminus B$. A function $f : 2^S \to R$ is *monotone* if and only if $f(A) \leq f(B)$ for $A \subseteq B \subseteq S$; it is *non-monotone* if is not necessarily the case. Since the number of sets is exponential, we assume a value oracle access to the submodular function; i.e., for a given set T, an algorithm can query an oracle to find its value $f(T)$. As we discuss below, maximizing a (monotone or non-monotone) submodular function which demonstrates economy of scale is a central and very general problem in combinatorial optimization and has been subject of a thorough study in the literature.

The closest in terms of generalization to our submodular multiple-choice secretary problem is the *matroid secretary problem* considered by Babaioff et al. [8]. In this problem, we are given a matroid by a ground set $\mathcal{U}$ of elements and a collection of independent (feasible) subsets $\mathcal{I} \subseteq 2^{\mathcal{U}}$ describing the sets of elements which can be simultaneously accepted. We recall that a matroid has three properties: 1) the empty set is independent; 2) every subset of an independent set is independent (closed under containment)[1]; and finally 3) if A and B are two independent sets and A has more elements than B, then there exists an element in A which is not in B and when added to B still gives an independent set[2]. The goal is to design online algorithms in which the structure of $\mathcal{U}$ and $\mathcal{I}$ is known at the outset (assume we have an oracle to answer whether a subset of $\mathcal{U}$ belongs to $\mathcal{I}$ or not), while the elements and their values are revealed one at a time in random order. As each element is presented, the algorithm must make an irrevocable decision to select or reject it such that the set of selected elements belongs to $\mathcal{I}$ at all times. Babaioff et al. present an $O(\log r)$-competitive algorithm for general matroids, where r is the rank of the matroid (the size of the maximal independent set), and constant-competitive algorithms for several special cases arising in practical scenarios including graphic matroids, truncated partition matroids, and bounded degree transversal matroids. However, they leave as a main open question the existence of constant-competitive algorithms for general matroids. Our constant-competitive algorithms for the submodular secretary problem in this paper can be considered in parallel with this open question. To generalize both results of Babaioff et al. and ours, we also consider the *submodular matroid secretary problem* in which we want to maximize a submodular function over all independent (feasible) subsets $\mathcal{I}$ of the given matroid. Moreover, we extend our approach to the case in which l matroids are given and the goal is to find the set of maximum value which is independent with respect to all the given matroids. We present an $O(l \log^2 r)$-competitive algorithm for the submodular matroid secretary problem generalizing previous results.

Prior to our work, there was no polynomial-time algorithm with a nontrivial guarantee for the case of l matroids—even in the offline setting—when l is not a fixed constant. Lee et al. [?] give a local-search procedure for the offline setting that runs in time $O(n^l)$ and achieves approximation ratio $l + \epsilon$. Even the simpler case of having a linear function cannot be approximated to within a factor better than $\Omega(l/\log l)$ [?]. Our results imply an

[1]This is sometimes called the *hereditary property*.

[2]This is sometimes called the *augmentation property* or the *independent set exchange property*.

algorithm with guarantees $O(l \log r)$ and $O(l \log^2 r)$ for the offline and (online) secretary settings, respectively. Both these algorithms run in time polynomial in l. In case of the knapsack constraints, the only previous relevant work that we are aware of is that of Lee et al. [?] which gives a $(5+\epsilon)$-approximation in the offline setting if the number of constraints is a constant. In contrast, our results work for arbitrary number of knapsack constraints.

Our competitive ratio for the submodular secretary problem is $\frac{7}{1-1/e}$. Though our algorithm is relatively simple, it has several phases and its analysis is relatively involved. As we point out below, we cannot obtain any approximation factor better than $1 - 1/e$ even for offline special cases of our setting unless $\mathbf{P} = \mathbf{NP}$. A natural generalization of a submodular function while still preserving economy of scale is a subadditive function $f : 2^S \to R$ in which $\forall A, B \subseteq S : f(A) + f(B) \geq f(A \cup B)$. In this paper, we show that if we consider the subadditive secretary problem instead of the submodular secretary problem, there is no algorithm with competitive ratio $\tilde{o}(\sqrt{n})$. We complement this result by giving an $O(\sqrt{n})$-competitive algorithm for the subadditive secretary problem.

Background on submodular maximization Submodularity, a discrete analog of convexity, has played a central role in combinatorial optimization [40]. It appears in many important settings including cuts in graphs [25, 32, 42], plant location problems [11, 12], rank function of matroids [15], and set covering problems [18].

The problem of maximizing a submodular function is of essential importance, with special cases including Max Cut [25], Max Directed Cut [28], hypergraph cut problems, maximum facility location [2, 11, 12], and certain restricted satisfiability problems [17, 29]. While the Min Cut problem in graphs is a classical polynomial-time solvable problem, and more generally it has been shown that any submodular function can be minimized in polynomial time [32, 44], maximization turns out to be more difficult and indeed all the aforementioned special cases are NP-hard.

Max-k-Cover, where the goal is to choose k sets whose union is as large as possible, is another related problem. It is shown that a greedy algorithm provides a $(1 - 1/e)$-approximation for Max-k-Cover [35] and this is optimal unless $\mathbf{P} = \mathbf{NP}$ [18]. More generally, we can view this problem as maximization of a monotone submodular function under a cardinality constraint, that is, we seek a set S of size k maximizing $f(S)$. The greedy algorithm again provides a $(1 - 1/e)$-approximation for this problem [41]. A $1/2$-approximation has been developed for maximizing monotone submodular functions under a matroid constraint [21]. A $(1 - 1/e)$-approximation has been also obtained for a knapsack constraint [45], and for a special class of submodular functions under a matroid constraint [10].

Recently constant factor $(\frac{3}{4}+\epsilon)$-approximation algorithms for maximizing non-negative non-monotone submodular functions has also been obtained [20]. Typical examples of such a problem are max cut and max directed cut. Here, the best approximation factors are 0.878 for max cut [25] and 0.859 for max directed cut [17]. The approximation factor for max cut has been proved optimal, assuming the Unique Games Conjecture [34]. Generalizing these results, Vondrak very recently obtains a constant factor approximation algorithm for maximizing non-monotone submodular functions under a matroid constraint [47]. Subadditive maximization has been also considered recently (e.g. in the context of maximizing welfare [19]).

Submodular maximization also plays a role in maximizing the difference of a monotone submodular function and a modular function. A typical example of this type is the maximum facility location problem in which we want to open a subset of facilities and maximize the total profit from clients minus the opening cost of facilities. Approximation algorithms have been developed for a variant of this problem which is a special case of maximizing nonnegative submodular functions [2, 11, 12]. The current best approximation factor known for this problem is 0.828 [2]. Asadpour et al. [4] study the problem of maximizing a submodular function in a stochastic setting, and obtain constant-factor approximation algorithms.

Our results and techniques The main theorem in this paper is as follows.

Theorem 3.1.1. *There exists a* $\frac{7}{1-1/e}$*-competitive algorithm for the monotone submodular secretary problem. More generally there exists a* $8e^2$*-competitive algorithm for the non-monotone submodular secretary problem.*

We prove Theorem 3.1.1 in Section 3.2. We first present our simple algorithms for the problem. Since our algorithm for the general non-monotone case uses that of monotone case, we first present the analysis for the latter case and then extend it for the former case. We divide the input stream into equal-sized segments, and show that restricting the algorithm to pick only one item from each segment decreases the value of the optimum by at most a constant factor. Then in each segment, we use a standard secretary algorithm to pick the best item conditioned on our previous choices. We next prove that these local optimization steps lead to a global near-optimal solution.

The argument breaks for the non-monotone case since the algorithm actually approximates a set which is larger than the optimal solution. The trick is to invoke a new structural property of (non-monotone) submodular functions which allows us to divide the input into two equal portions, and randomly solve the problem on one.

Indeed Theorem 3.1.1 can be extended for the submodular matroid secretary problem as follows.

Theorem 3.1.2. *There exists an $O(l \log^2 r)$ competitive algorithm for the (non-monotone) matroid submodular secretary problem, where r is the maximum rank of the given l matroids.*

We prove theorem 3.1.2 in Section 3.3. We note that in the submodular matroid secretary problem, selecting (bad) elements early in the process might prevent us from selecting (good) elements later since there are matroid independence (feasibility) constraints. To overcome this issue, we only work with the first half of the input. This guarantees that at each point in expectation there is a large portion of the optimal solution that can be added to our current solution without violating the matroid constraint. However, this set may not have a high value. As a remedy we prove there is a near-optimal solution all of whose large subsets have a high value. This novel argument may be of its own interest.

We shortly mention in Section 3.4 our results for maximizing a submodular secretary problem with respect to l knapsack constraints. In this setting, there are l knapsack capacities $C_i : 1 \leq i \leq l$, and each item j has different weights w_{ij} associated with each knapsack. A set T of items is feasible if and only if for *each* knapsack i, we have $\sum_{j \in T} w_{ij} \leq C_i$.

Theorem 3.1.3. *There exists an $O(l)$-competitive algorithm for the (non-monotone) multiple knapsack submodular secretary problem, where l denotes the number of given knapsack constraints.*

Lee et al. [?] gives a better $(5 + \epsilon)$-approximation in the offline setting if l is a fixed constant.

We next show that indeed submodular secretary problems are the most general cases that we can hope for constant competitiveness.

Theorem 3.1.4. *For the subadditive secretary problem, there is no algorithm with competitive ratio in $\tilde{o}(\sqrt{n})$. However there is an algorithm with almost tight $O(\sqrt{n})$ competitive ratio in this case.*

We prove Theorem 3.1.4 in Section 3.5. The algorithm for the matching upper bound is very simple, however the lower bound uses clever ideas and indeed works in a more general setting. We construct a subadditive function, which interestingly is almost submodular, and has a "hidden good set". Roughly speaking, the value of any query to the oracle is proportional to the intersection of the query and the hidden good set. However, the oracle's response does not change unless the query has considerable intersection with the good set which is hidden. Hence, the oracle does not give much information about the hidden good set.

Finally in our concluding remarks in Section 3.6, we briefly discuss two other aggregate functions max and min, where the latter is not even submodular and models a bottle-neck situation in the secretary problem.

All omitted proofs can be found in the appendix.

3.2 The submodular secretary problem

3.2.1 Algorithms

In this sections, we present the algorithms used to prove Theorem 3.1.1. In the classic secretary problem, the efficiency value of each secretary is known only after she arrives. In order to marry this with the value oracle model, we say that the oracle answers the query regarding the efficiency of a set $S' \subseteq S$ only if all the secretaries in S' have already arrived and been interviewed.

Algorithm 1 Monotone Submodular Secretary Algorithm

Input: A monotone submodular function $f : 2^S \rightarrow R$, and a randomly permuted stream of secretaries, denoted by $(a_1, a_2, \ldots, a_n)$, where n is an integer multiple of k.
Output: A subset of at most k secretaries.

$T_0 := \emptyset$
$l := n/k$
For $i := 1$ to k
$u_i := (i-1)l + l/e$
$\alpha_i := \max_{(i-1)l \le j < u_i} f(T_{i-1} \cup \{a_j\})$
If $\alpha_i < f(T_{i-1})$ then $\alpha_i := f(T_{i-1})$
Pick an index $p_i : u_i \le p_i < il$ such that $f(T_{i-1} \cup \{a_{p_i}\}) \ge \alpha_i$
If such an index p_i exists then $T_i := T_{i-1} \cup \{a_{p_i}\}$
Else
$T_i := T_{i-1}$
Output T_k as the solution

Our algorithm for the monotone submodular case is relatively simple though its analysis is relatively involved. First we assume that n is a multiple of k, since otherwise we could virtually insert $n - k\lfloor \frac{n}{k} \rfloor$ dummy secretaries in the input: for any subset A of dummy secretaries and a set $B \subseteq S$, we have that $f(A \cup B) = f(B)$. In other words, there is no profit in employing the dummy secretaries. To be more precise, we simulate the augmented input in such a way that these secretaries are arriving uniformly at random similarly to the real ones. Thus, we say that n is a multiple of k without loss of generality.

We partition the input stream into k equally-sized segments, and, roughly speaking, try to employ the *best* secretary in each segment. Let $l := \frac{n}{k}$ denote the length of each segment. Let $a_1, a_2, \cdots, a_n$ be the actual ordering in which the secretaries are interviewed. Break the input into k segments such that $S_j = \{a_{(j-1)l+1}, a_{(j-1)l+2}, \ldots, a_{jl}\}$ for $1 \le j < k$, and $S_k = \{a_{(k-1)l+1}, a_{(k-1)l+2}, \ldots, a_n\}$. We employ at most one secretary from each segment S_i. Note that this way of having several phases of (almost) equal length for the secretary problem seems novel to this paper, since in previous works there are usually only two phases (see e.g. [27]). The phase i of our algorithm corresponds to the time interval when the secretaries in S_i arrive. Let T_i be the set of secretaries that we have employed from $\bigcup_{j=1}^{i} S_j$. Define $T_0 := \emptyset$ for convenience. In phase i, we try to employ a secretary e from S_i that maximizes $f(T_{i-1} \cup \{e\}) - f(T_{i-1})$. For each $e \in S_i$, we define $g_i(e) = f(T_{i-1} \cup \{e\}) - f(T_{i-1})$. Then, we are trying to employ a secretary $x \in S_i$ that has the maximum value for $g_i(e)$. Using a classic algorithm for the *secretary problem* (see [14] for instance) for employing the single secretary, we can solve this problem with constant probability $1/e$. Hence, with constant probability, we pick the secretary that maximizes our local profit in each phase. It leaves us to prove that this local optimization leads to a reasonable global guarantee.

The previous algorithm fails in the non-monotone case. Observe that the first **if** statement is never true for a monotone function, however, for a non-monotone function this guarantees the values of sets T_i are non-decreasing. Algorithm 2 first divides the input stream into two equal-sized parts: U_1 and U_2. Then, with probability $1/2$, it calls Algorithm 1 on U_1, whereas with the same probability, it skips over the first half of the input, and runs Algorithm 1 on U_2.

Algorithm 2 Submodular Secretary Algorithm

Input: A (possibly non-monotone) submodular function $f : 2^S \to R$, and a randomly permuted stream of secretaries, denoted by $(a_1, a_2, \ldots, a_n)$, where n is an integer multiple of $2k$.
Output: A subset of at most k secretaries.

$U_1 := \{a_1, a_2, \ldots, a_{n/2}\}$
$U_2 := \{a_{n/2} + 1, \ldots, a_{n-1}, a_n\}$
$0 \leq X \leq 1$ be a uniformly random value.
If $X \leq 1/2$
Run Algorithm 1 on U_1 to get S_1
Output S_1 as the solution
Else
Run Algorithm 1 on U_2 to get S_2
Output S_2 as the solution

3.2.2 Analysis

In this section, we prove Theorem 3.1.1. Since the algorithm for the non-monotone submodular secretary problem uses that for the monotone submodular secretary problem, first we start with the monotone case.

Monotone submodular

We prove in this section that for Algorithm 1, the expected value of $f(T_k)$ is within a constant factor of the optimal solution. Let $R = \{a_{i_1}, a_{i_2}, \cdots, a_{i_k}\}$ be the optimal solution. Note that the set $\{i_1, i_2, \cdots, i_k\}$ is a uniformly random subset of $\{1, 2, \cdots, n\}$ with size k. It is also important to note that the permutation of the elements of the optimal solution on these k places is also uniformly random, and is independent from the set $\{i_1, i_2, \cdots, i_k\}$. For example, any of the k elements of the optimum can appear as a_{i_1}. These are two key facts used in the analysis.

Before starting the analysis, we present a simple property of submodular functions which will prove useful in the analysis. The proof of the lemma is standard, and is included in the appendix for the sake of completeness.

Lemma 3.2.1. *If $f : 2^S \to R$ is a submodular function, we have $f(B) - f(A) \leq \sum_{a \in B \setminus A} [f(A \cup \{a\}) - f(A)]$ for any $A \subseteq B \subseteq S$.*

Define $X := \{S_i : |S_i \cap R| \neq \emptyset\}$. For each $S_i \in X$, we pick one element, say s_i, of $S_i \cap R$ randomly. These selected items form a set called $R' = \{s_1, s_2, \cdots, s_{|X|}\} \subseteq R$ of size $|X|$. Since our algorithm approximates such a set, we study the value of such random samples of R in the following lemmas. We first show that restricting ourselves to picking at most one element from each segment does not prevent us from picking many elements from the optimal solution (i.e., R).

Lemma 3.2.2. *The expected value of the number of items in R' is at least $k(1 - 1/e)$.*

Proof. We know that $|R'| = |X|$, and $|X|$ is equal to k minus the number of sets S_i whose intersection with R is empty. So, we compute the expected number of these sets, and subtract this quantity from k to obtain the expected value of $|X|$ and thus $|R'|$.

Consider a set S_q, $1 \leq q \leq k$, and the elements of $R = \{a_{i_1}, a_{i_2}, \ldots, a_{i_k}\}$. Define E_j as the event that a_{i_j} is not in S_q. We have $\Pr(E_1) = \frac{(k-1)l}{n} = 1 - \frac{1}{k}$, and for any $i : 1 < i \leq k$, we get

$$\Pr\left(E_i \middle| \bigcap_{j=1}^{i-1} E_j\right) = \frac{(k-1)l - (i-1)}{n - (i-1)} \leq \frac{(k-1)l}{n} = 1 - \frac{1}{k},$$

where the last inequality follows from a simple mathematical fact: $\frac{x-c}{y-c} \leq \frac{x}{y}$ if $c \geq 0$ and $x \leq y$. Now we conclude that the probability of the event $S_q \cap R = \emptyset$ is

$$\Pr(\cap_{i=1}^{k} E_i) = \Pr(E_1) \cdot \Pr(E_2|E_1) \cdots \Pr(E_k| \cap_{j=1}^{k-1} E_j) \leq \left(1 - \frac{1}{k}\right)^k \leq \frac{1}{e}.$$

Thus each of the sets $S_1, S_2, \ldots, S_k$ does not intersect with R with probability at most $1/e$. Hence, the expected number of such sets is at most k/e. Therefore, the expected value of $|X| = |R'|$ is at least $k(1 - 1/e)$. □

The next lemma materializes the proof of an intuitive statement: if you randomly sample elements of the set R, you expect to obtain a profit proportional to the size of your sample. An analog this is proved in [19] for the case when $|R|/|A|$ is an integer.

Lemma 3.2.3. *For a random subset A of R, the expected value of $f(A)$ is at least $\frac{|A|}{k} \cdot f(R)$.*

Proof. Let $(x_1, x_2, \ldots, x_k)$ be a random ordering of the elements of R. For $r = 1, 2, \ldots, k$, let F_r be the expectation of $f(\{x_1, \ldots, x_r\})$, and define $D_r := F_r - F_{r-1}$, where F_0 is interpreted to be equal to zero. Letting $a := |A|$, note that $f(R) = F_k = D_1 + \cdots + D_k$, and that the expectation of $f(A)$ is equal to $F_a = D_1 + \cdots + D_a$. We claim that $D_1 \geq D_2 \geq \cdots \geq D_k$, from which the lemma follows easily. Let $(y_1, y_2, \ldots, y_k)$ be a cyclic permutation of $(x_1, x_2, \ldots, x_k)$, where $y_1 = x_k, y_2 = x_1, y_3 = x_2, \ldots, y_k = x_{k-1}$. Notice that for $i < k$, F_i is equal to the expectation of $f(\{y_2, \ldots, y_{i+1}\})$ since $\{y_2, \ldots, y_{i+1}\}$ is equal to $\{x_1, \ldots, x_i\}$.

F_i is also equal to the expectation of $f(\{y_1, \ldots, y_i\})$, since the sequence $(y_1, \ldots, y_i)$ has the same distribution as that of $(x_1, \cdots, x_i)$. Thus, D_{i+1} is the expectation of $f(\{y_1, \ldots, y_{i+1}\}) - f(\{y_2, \ldots, y_{i+1}\})$, whereas D_i is the expectation of $f(\{y_1, \ldots, y_i\}) - f(\{y_2, \ldots, y_i\})$. The submodularity of f implies that $f(\{y_1, \ldots, y_{i+1}\})$ - $f(\{y_2, \ldots, y_{i+1}\})$ is less than or equal to $f(\{y_1, \ldots, y_i\})$ - $f(\{y_2, \ldots, y_i\})$, hence $D_{i+1} \leq D_i$. □

Here comes the crux of our analysis where we prove that the local optimization steps (i.e., trying to make the best move in each segment) indeed lead to a globally approximate solution.

Lemma 3.2.4. *The expected value of $f(T_k)$ is at least $\frac{|R'|}{7k} \cdot f(R)$.*

Lemma 3.2.4. Define $m := |R'|$ for the ease of reference. Recall that R' is a set of secretaries $\{s_1, s_2, \ldots, s_m\}$ such that $s_i \in S_{h_i} \cap R$ for $i : 1 \leq i \leq m$ and $h_i : 1 \leq h_i \leq k$. Also assume without loss of generality that $h_{i'} \leq h_i$ for $1 \leq i' < i \leq m$, for instance, s_1 is the first element of R' to appear. Define Δ_j for each $j : 1 \leq j \leq k$ as the gain of our algorithm while working on the segment S_j. It is formally defined as $\Delta_j := f(T_j) - f(T_{j-1})$. Note that due to the first **if** statement in the algorithm, $\Delta_j \geq 0$ and thus $Ex[\Delta_j] \geq 0$. With probability $1/e$, we choose the element in S_j which maximizes the value of $f(T_j)$ (given that the set T_{j-1} is fixed). Notice that by definition of R' only one s_i appears in S_{h_i}. Since $s_i \in S_{h_i}$ is one of the options,

$$Ex[\Delta_{h_i}] \geq \frac{Ex[f(T_{h_i-1} \cup \{s_i\}) - f(T_{h_i-1})]}{e}. \tag{3.1}$$

To prove by contradiction, suppose $Ex[f(T_k)] < \frac{m}{7k} \cdot f(R)$. Since f is monotone, $Ex[f(T_j)] < \frac{m}{7k} \cdot f(R)$ for any $0 \leq j \leq k$. Define $B := \{s_i, s_{i+1}, \cdots, s_m\}$. By Lemma 3.2.1 and monotonicity of f,

$$f(B) \leq f(B \cup T_{h_i-1}) \leq f(T_{h_i-1}) + \sum_{j=i}^{m} [f(T_{h_i-1} \cup \{s_j\}) - f(T_{h_i-1})],$$

which implies

$$Ex[f(B)] \leq Ex[f(T_{h_i-1})] + \sum_{j=i}^{m} Ex[f(T_{h_i-1} \cup \{s_j\}) - f(T_{h_i-1})].$$

Since the items in B are distributed uniformly at random, and there is no difference between s_{i_1} and s_{i_2} for $i \leq i_1, i_2 \leq m$, we can say

$$Ex[f(B)] \leq Ex[f(T_{h_i-1})] + (m - i + 1) \cdot Ex[f(T_{h_i-1} \cup \{s_i\}) - f(T_{h_i-1})]. \tag{3.2}$$

We conclude from (3.1) and (3.2)

$$Ex[\Delta_{h_i}] \geq \frac{Ex[f(T_{h_i-1} \cup \{s_i\}) - f(T_{h_i-1})]}{e} \geq \frac{Ex[f(B)] - Ex[f(T_{h_i-1})]}{e(m-i+1)}.$$

Since B is a random sample of R, we can apply Lemma 3.2.3 to get $Ex[f(B)] \geq \frac{|B|}{k} f(R) = f(R)(m-i+1)/k$. Since $Ex[f(T_{h_i-1})] \leq \frac{m}{7k} \cdot f(R)$, we reach

$$Ex[\Delta_{h_i}] \geq \frac{Ex[f(B)] - Ex[f(T_{h_i-1})]}{e(m-i+1)} \geq \frac{f(R)}{ek} - \frac{m}{7k} f(R) \cdot \frac{1}{e(m-i+1)}. \tag{3.3}$$

Adding up (3.3) for $i : 1 \leq i \leq \lceil m/2 \rceil$, we obtain

$$\sum_{i=1}^{\lceil m/2 \rceil} Ex[\Delta_{h_i}] \geq \left\lceil \frac{m}{2} \right\rceil \cdot \frac{f(R)}{ek} - \frac{m}{7ek} \cdot f(R) \cdot \sum_{i=1}^{\lceil m/2 \rceil} \frac{1}{m-i+1}.$$

Since $\sum_{j=a}^{b} \frac{1}{j} \leq \ln \frac{b}{a+1}$ for any integer values of $a, b : 1 < a \leq b$, we conclude

$$\sum_{i=1}^{\lceil m/2 \rceil} Ex[\Delta_{h_i}] \geq \left\lceil \frac{m}{2} \right\rceil \cdot \frac{f(R)}{ek} - \frac{m}{7ek} \cdot f(R) \cdot \ln \frac{m}{\lfloor \frac{m}{2} \rfloor}.$$

A similar argument for the range $1 \leq i \leq \lfloor m/2 \rfloor$ gives

$$\sum_{i=1}^{\lfloor \frac{m}{2} \rfloor} Ex[\Delta_{h_i}] \geq \left\lfloor \frac{m}{2} \right\rfloor \cdot \frac{f(R)}{ek} - \frac{m}{7ek} \cdot f(R) \cdot \ln \frac{m}{\lceil \frac{m}{2} \rceil}.$$

We also know that both $\sum_{i=1}^{\lfloor m/2 \rfloor} Ex[\Delta_{h_i}]$ and $\sum_{i=1}^{\lceil m/2 \rceil} Ex[\Delta_{h_i}]$ are at most $Ex[f(T_k)]$ because $f(T_k) \geq \sum_{i=1}^{m} \Delta_{h_i}$. We conclude with

$$2Ex[f(T_k)] \geq \left\lceil \frac{m}{2} \right\rceil \frac{f(R)}{ek} - \frac{mf(R)}{7ek} \cdot \ln \frac{m}{\lfloor \frac{m}{2} \rfloor}$$

$$+ \left\lfloor \frac{m}{2} \right\rfloor \frac{f(R)}{ek} - \frac{mf(R)}{7ek} \cdot \ln \frac{m}{\lceil \frac{m}{2} \rceil}$$

$$\geq \frac{mf(R)}{ek} - \frac{mf(R)}{7ek} \cdot \ln \frac{m^2}{\lfloor \frac{m}{2} \rfloor \lceil \frac{m}{2} \rceil}$$

and since $\frac{m^2}{\lfloor m/2 \rfloor \lceil m/2 \rceil} < 4.5$

$$\geq \frac{mf(R)}{ek} - \frac{mf(R)}{7ek}.ln(4.5)$$

$$= \frac{mf(R)}{k}.\left(\frac{1}{e} - \frac{\ln 4.5}{7e}\right) \geq \frac{mf(R)}{k}.\frac{2}{7}$$

which contradicts $Ex[f(T_k)]$ ¡ $\frac{mf(R)}{7k}$, hence proving the supposition false. □

The following theorem wraps up the analysis of the algorithm.

Theorem 3.2.5. *The expected value of the output of our algorithm is at least* $\frac{1-1/e}{7}f(R)$.

Proof. The expected value of $|R'| = m \geq (1 - 1/e)k$ from Lemma 3.2.2. In other words, we have $\sum_{m=1}^{k} \Pr[|R'| = m] \cdot m \geq \left(1 - \frac{1}{e}\right)k$. We know from Lemma 3.2.4 that if the size of R' is m, the expected value of $f(T_k)$ is at least $\frac{m}{7k}f(R)$, implying that $\sum_{v \in V} \Pr\left[f(T_k) = v \middle| |R'| = m\right] \cdot v \geq \frac{m}{7k}f(R)$, where V denotes the set of different values that $f(T_k)$ can get. We also know that

$$Ex[f(T_k)] = \sum_{m=1}^{k} Ex[f(T_k)||R'| = m] \Pr[|R'| = m] \geq \sum_{m=1}^{k} \frac{m}{7k}f(R) \Pr[|R'| = m] = \frac{f(R)}{7k}Ex[|R'|] \geq \frac{1 -}{}$$

□

Non-monotone submodular

Before starting the analysis of Algorithm 2 for non-monotone functions, we show an interesting property of Algorithm 1. Consistently with the notation of Section 3.2.2, we use R to refer to some optimal solution. Recall that we partition the input stream into (almost) equal-sized segments $S_i : 1 \leq i \leq k$, and pick one item from each. Then T_i denotes the set of items we have picked at the completion of segment i. We show that $f(T_k) \geq \frac{1}{2e}f(R \cup T_i)$ for some integer i, even when f is not monotone. Roughly speaking, the proof mainly follows from the submodularity property and Lemma 3.2.1.

Lemma 3.2.6. *If we run the monotone algorithm on a (possibly non-monotone) submodular function f, we obtain $f(T_k) \geq \frac{1}{2e^2}f(R \cup T_i)$ for some i.*

Proof. Consider the stage $i+1$ in which we want to pick an item from S_{i+1}. Lemma 3.2.1 implies

$$f(R \cup T_i) \leq f(T_i) + \sum_{a \in R \setminus T_i} f(T_i \cup \{a\}) - f(T_i).$$

At least one of the two right-hand side terms has to be larger than $f(R \cup T_i)/2$. If this happens to be the first term for any i, we are done: $f(T_k) \geq f(T_i) \geq \frac{1}{2} f(R \cup T_i)$ since $f(T_k) \geq f(T_i)$ by the definition of the algorithm: the first **if** statement makes sure $f(T_i)$ values are non-decreasing. Otherwise assume that the lower bound occurs for the second terms for all values of i.

Consider the events that among the elements in $R \setminus T_i$ exactly one, say a, falls in S_{i+1}. Call this event E_a. Conditioned on E_a, $\Delta_{i+1} := f(T_{i+1}) - f(T_i)$ is at least $f(T_i \cup \{a\}) - f(T_i)$ with probability $1/e$: i.e., if the algorithm picks the best secretary in this interval. Each event E_a occurs with probability at least $\frac{1}{k} \cdot \frac{1}{e}$. Since these events are disjoint, we have

$$Ex[\Delta_{i+1}] \geq \sum_{a \in R \setminus T_i} \Pr[E_a] . \frac{f(T_{i+1}) - f(T_i)}{e}$$

$$\geq \frac{1}{e^2 k} \sum_{a \in R \setminus T_i} f(T_i \cup \{a\}) - f(T_i)$$

$$\geq \frac{1}{2e^2 k} f(R \cup T_i)$$

and by summing over all values of i, we obtain:

$$Ex[f(T_k)] = \sum_i Ex[\Delta_i] \geq \sum_i \frac{1}{2e^2 k} f(R \cup T_i) \geq \frac{1}{2e^2} \min_i f(R \cup T_i). \quad \square$$

$\square$

Unlike the case of monotone functions, we cannot say that $f(R \cup T_i) \geq f(R)$, and conclude that our algorithm is constant-competitive. Instead, we need to use other techniques to cover the cases that $f(R \cup T_i) < f(R)$. The following lemma presents an upper bound on the value of the optimum.

Lemma 3.2.7. *For any pair of disjoint sets Z and Z', and a submodular function f, we have $f(R) \leq f(R \cup Z) + f(R \cup Z')$.*

Proof. The statement follows from the submodularity property, observing that $(R \cup Z) \cap (R \cup Z') = R$, and $f([R \cup Z] \cup [R \cup Z']) \geq 0$. $\square$

We are now at a position to prove the performance guarantee of our main algorithm.

Theorem 3.2.8. *Algorithm 2 has competitive ratio $8e^2$.*

Proof. Let the outputs of the two algorithms be sets Z and Z', respectively. The expected value of the solution is thus $[f(Z) + f(Z')]/2$.

We know that $Ex[f(Z)] \geq c' f(R \cup X_1)$ for some constant c', and $X_1 \subset U_1$. The only difference in the proof is that each element of $R \setminus Z$ appears in the set S_i with probability $1/2k$ instead of $1/k$. But we can still prove the above lemma for $c' := 1/4e^2$. Same holds for Z': $Ex[f(Z')] \geq \frac{1}{4e} f(R \cup X_2)$ for some $X_2 \subseteq U_2$.

Since U_1 and U_2 are disjoint, so are X_1 and X_2. Hence, the expected value of our solution is at least $\frac{1}{4e^2}[f(R \cup X_1) + f(R \cup X_2)]/2$, which via Lemma 3.2.7 is at least $\frac{1}{8e^2} f(R)$. $\square$

3.3 The submodular matroid secretary problem

In this section, we prove Theorem 3.1.2. We first design an $O(\log^2 r)$-competitive algorithm for maximizing a monotone submodular function, when there are matroid constraints for the set of selected items. Here we are allowed to choose a subset of items only if it is an independent set in the given matroid.

The matroid (U, I) is given by an oracle access to I. Let n denote the number of items, i.e., $n := |U|$, and r denotes the rank of the matroid. Let $S \in I$ denote an optimal solution that maximizes the function f. We focus our analysis on a refined set $S^* \subseteq S$ that has certain nice properties: 1) $f(S^*) \geq (1 - 1/e) f(S)$, and 2) $f(T) \geq f(S^*)/\log r$ for any

$T \subseteq S^*$ such that $|T| = \lfloor |S^*|/2 \rfloor$. We cannot necessarily find S^*, but we prove that such a set exists.

Start by letting $S^* = S$. As long as there is a set T violating the second property above, remove T from S^*, and continue. The second property clearly holds at the termination of the procedure. In order to prove the first property, consider one iteration. By submodularity (subadditivity to be more precise) we have $f(S^* \backslash T) \geq f(S^*) - f(T) \geq (1 - 1/\log r) f(S^*)$. Since each iteration halves the set S^*, there are at most $\log r$ iterations. Therefore, $f(S^*) \geq (1 - 1/\log r)^{\log r} \cdot f(S) \geq (1 - 1/e) f(S)$.

We analyze the algorithm assuming the parameter $|S^*|$ is given, and achieve a competitive ratio $O(\log r)$. If $|S^*|$ is unknown, though, we can guess its value (from a pool of $\log r$ different choices) and continue with Lemma 3.3.1. This gives an $O(\log^2 r)$-competitive ratio.

Algorithm 3 Monotone Submodular Secretary Algorithm with Matroid constraint

---

Input: A monotone submodular function $f : 2^U \to R$, a matroid (U, I), and a randomly permuted stream of secretaries, denoted by $(a_1, a_2, \ldots, a_n)$.
Output: A subset of secretaries that are independent according to I.

$U_1 := \{a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor}\}$
Pick the parameter $k := |S^*|$ uniformly at random
from the pool $\{2^0, 2^1, 2^{\log r}\}$
If $k = O(\log r)$
Select the best item of the U_1 and output the singleton
Else run Algorithm 1 on U_1 and respect the matroid
Run Algorithm 1 on U_1 to search for k items
and respect the matroid independence oracle I
$T_0 := \emptyset$
$l := \lfloor n/k \rfloor$
For $i := 1$ to k
$u_i := (i-1)l + l/e$
$\alpha_i := \max_{\substack{(i-1)l \leq j < u_i \\ T_{i-1} \cup \{a_j\} \in I}} f(T_{i-1} \cup \{a_j\})$
If $\alpha_i < f(T_{i-1})$ then $\alpha_i := f(T_{i-1})$
Pick an index $p_i : u_i \leq p_i < il$ such that $f(T_{i-1} \cup \{a_{p_i}\}) \geq \alpha_i$ and $T_{i-1} \cup \{a_{p_i}\} \in I$
If such an index p_i exists then $T_i := T_{i-1} \cup \{a_{p_i}\}$
Else $T_i := T_{i-1}$
Output T_k as the solution

Lemma 3.3.1. *Given $|S^*|$, Algorithm 3 picks an independent subset of items with size $|S^*|/2$ whose expected value is at least $f(S^*)/4e\log r$.*

Proof. Let $k := |S^*|$. We divide the input stream of n items into k segments of (almost) equal size. We only pick $k/2$ items, one from each of the first $k/2$ segments.

Similarly to Algorithm 1 for the submodular secretary problem, when we work on each segment, we try to pick an item that maximizes the marginal value of the function given the previous selection is fixed (see the **for** loop in Algorithm 1). We show that the expected gain in each of the first $k/2$ segments is at least a constant fraction of $f(S^*)/k\log r$.

Suppose we are working on segment $i \leq k/2$, and let Z be the set of items already picked; so $|Z| \leq i-1$. Furthermore, assume $f(Z) \leq f(S^*)/2\log r$ since otherwise, the lemma is already proved. By matroid properties we know there is a set $T \subseteq S^* \setminus Z$ of size $\lfloor k/2 \lfloor$ such that $T \cup Z \in I$. The second property of S^* gives $f(T) \geq f(S^*)/\log r$.

From Lemma 3.2.1 and monotonicity of f, we obtain

$$\sum_{s\in T}[f(Z\cup\{s\}) - f(Z)] \geq f(T\cup Z) - f(Z) \geq f(T) - f(Z) \geq f(S^*)/2\log r.$$

Note that each item in T appears in this segment with probability $2/k$ because we divided the input stream into $k/2$ equal segments. Since in each segment we pick the item giving the maximum marginal value with probability $1/e$, the expected gain in this segment is at least

$$\sum_{s\in T}\frac{1}{e}\cdot\frac{2}{k}\cdot[f(Z\cup\{s\}) - f(Z)] \geq f(S^*)/ek\log r.$$

We have this for each of the first $k/2$ segments, so the expected value of our solution is at least $f(S^*)/2e\log r$. □

Finally, it is straightforward (and hence the details are omitted) to combine the algorithm in this section with Algorithm 2 for the nonmonotone submodular secretary problem, to obtain an $O(\log^2 r)$-competitive algorithm for the non-monotone submodular secretary problem subject to a matroid constraint.

Here we show the same algorithm works when there are $l \geq 1$ matroid constraints and achieves a competitive ratio of $O(l \log^2 r)$. We just need to respect all matroid constraints in Algorithm 3. This finishes the proof of Theorem 3.1.2.

Lemma 3.3.2. *Given $|S^*|$, Algorithm 3 picks an independent subset of items (i.e., independent with respect to all matroids) with expected value at least $f(S^*)/4el \log r$.*

Proof. The proof is similar to the proof of Lemma 3.3.1. We show that the expected gain in each of the first $k/2l$ segments is at least a constant fraction of $f(S^*)/k \log r$.

Suppose we are working on segment $i \leq k/2l$, and let Z be the set of items already picked; so $|Z| \leq i - 1$. Furthermore, assume $f(Z) \leq f(S^*)/2 \log r$ since otherwise, the lemma is already proved. We claim that there is a set $T \subseteq S^* \backslash Z$ of size $k - l \times \lfloor k/2l \rfloor \geq k/2$ such that $T \cup Z$ is an independent set in all matroids. The proof is as follows. We know that there exists a set $T_1 \subseteq S^*$ whose union with Z is an independent set of the first matroid, and the size of T_1 is at least $|S^*| - |Z|$. This can be proved by the exchange property of matroids, i.e., adding Z to the independent set S^* does not remove more than $|Z|$ items from S^*. Since T_1 is independent with respect to the second matroid (as it is a subset of S^*), we can prove that there exists a set $T_2 \subseteq T_1$ of size at least $|T_1| - |Z|$ such that $Z \cup T_2$ is an independent set in the second matroid. If we continue this process for all matroid constraints, we can prove that there is a set T_l which is an independent set in all matroids, and has size at least $|S^*| - l|Z| \geq k - l \times \lfloor k/2l \rfloor \geq k/2$ such that $Z \cup T_l$ is independent with respect to all the given matroids. The rest of the proof is similar to the proof of Lemma 3.3.1—we just need to use the set T_l instead of the set T in the proof.

Since we are gaining a constant times $f(S^*)/k \log r$ in each of the first $k/2l$ segments, the expected value of the final solution is at least a constant times $f(S^*)/l \log r$. $\square$

3.4 Knapsack constraints

In this section, we prove Theorem 3.1.3. We first outline how to reduce an instance with multiple knapsacks to an instance with only one knapsack, and then we show how to solve the single knapsack instance.

Without loss of generality, we can assume that all knapsack capacities are equal to one. Let I be the given instance with the value function f, and item weights w_{ij} for $1 \le i \le l$ and $1 \le j \le n$. Define a new instance I' with one knapsack of capacity one in which the weight of the item j is $w'_j := \max_i w_{ij}$. We first prove that this reduction loses no more than a factor $4l$ in the total value. Take note that both the scaling and the weight transformation can be carried in an online manner as the items arrive. Hence, the results of this section hold for the online as well as the offline setting.

Lemma 3.4.1. *With instance I' defined above, we have* $\frac{1}{4l}\,\mathrm{OPT}(I) \le \mathrm{OPT}(I') \le \mathrm{OPT}(I)$.

Proof. The latter inequality is very simple: Take the optimal solultion to I'. This is also feasible in I since all the item weights in I are bounded by the weight in I'.

We next prove the other inequality. Let T be the optimal solution of I. An item j is called *fat* if $w'_j \ge 1/2$. Notice that there can be at most $2l$ fat items in T since $\sum_{j\in T} w'_j \le \sum_{j\in T}\sum_i w_{ij} \le l$. If there is any fat item with value at least $\mathrm{OPT}(I)/4l$, the statement of the lemma follows immediately, so we assume this is not the case. The total value of the fat items, say F, is at most $\mathrm{OPT}(I)/2$. Submodularity and non-negativity of f gives $f(T\setminus F) \ge f(T) - f(F) \ge \mathrm{OPT}(I)/2$. Sort the non-fat items in decreasing order of their value density (i.e., ratio of value to weight), and let T' be a maximal prefix of this ordering that is feasible with respect to I'. If $T' = T\setminus F$, we are done; otherwise, T' has weight at least $1/2$. Let x be the total weight of items in T' and let y indicate the total weight of item $T \setminus (F \cup T')$. Let α_x and α_y denote the densities of the two corresponding subsets of the items, respectively. Clearly $x+y \le l$ and $\alpha_x \ge \alpha_y$. Thus, $f(T\setminus F) = \alpha_x\cdot x + \alpha_y \cdot y \le \alpha_x(x+y) \le \alpha_x\cdot l$. Now $f(T') \ge \alpha_x\cdot\frac{1}{2} \ge \frac{1}{2l} f(T\setminus F) \ge \frac{1}{4l} f(T)$ finishes the proof. $\square$

Here we show how to achieve a constant competitive ratio when there is only one knapsack constraint. Let w_j denote the weight of item $j : 1 \le j \le n$, and assume without loss of generality that the capacity of the knapsack is 1. Moreover, let f be the value function which is a non-monotone submodular function. Let T be the optimal solution, and define $\mathrm{OPT} := f(T)$. The value of the parameter $\lambda \ge 1$ will be fixed below. Define T_1 and T_2 as the subsets of T that appears in the first and second half of the input stream, respectively. We first show the this solution is broken into two *blanaced* portions.

Lemma 3.4.2. *If the value of each item is at most* OPT/λ, *for sufficiently large* λ, *the random variable* $|f(T_1) - f(T_2)|$ *is bounded by* $\mathrm{OPT}/2$ *with a constant probability.*

Proof. Each item of T goes to either T_1 or T_2 with probability $1/2$. Let the random variable X_j^1 denote the increase of the value of $f(T_1)$ due to the possible addition of item j. Similarly X_j^2 is defined for the same effect on $f(T_2)$. The two variables X_j^1 and X_j^2 have the same probability distribution, and because of submodularity and the fact that the value of item j is at most OPT/λ, the contribution of item j in $f(T_1) - f(T_2)$ can be seen as a random variable that always take values in range $[-\mathrm{OPT}/\lambda, \mathrm{OPT}/\lambda]$ with mean zero. (In fact, we also use the fact that in an optimal solution, the marginal value of any item is non-negative. Submodularity guarantees that this holds with respect to any of the subsets of T as well.) Azuma's inequality ensures that with constant probability the value of $|f(T_1) - f(T_2)|$ is not more than $\max\{f(T_1), f(T_2)\}/2$ for sufficiently large λ. Since both $f(T_1)$ and $f(T_2)$ are at most OPT, we can say that they are both at least $\mathrm{OPT}/4$, with constant probability. □

The algorithm is as follows. Without loss of generality assume that all items are feasible, i.e., any one item fits into the knapsack. We flip a coin, and if it turns up "heads," we simply try to pick the one item with the maximum value. We do the following if the coin turns up "tails." We do not pick any items from the first half of the stream. Instead, we compute the maximum value set in the first half with respect to the knapsack constraint; Lee et al. give a constant fator approximation for this task. From the above argument, we know that $f(T_1)$ is at least $OPT/4$ since all the items have limited value in this case (i.e., at most OPT/λ). Therefore, we obtain a constant factor estimation of OPT by looking at the first half of the stream: i.e., if the estimate is $\hat{\mathrm{OPT}}$, we get $\mathrm{OPT}/c \leq \hat{\mathrm{OPT}} \leq \mathrm{OPT}$. After obtaining this estimate, we go over the second half of the input, and pick an item j if and only if it is feasible to pick this item, and moreover, the ratio of its marginal value to w_j is at least $\hat{\mathrm{OPT}}/6$.

Lemma 3.4.3. *The above algorithm is a constant competitive algorithm for the non-monotone submodular secretary problem with one knapsack constraint.*

Proof. We give the proof for the monotone case. Extending it for the non-monotone requires the same idea as was used in the proof of Theorem 2. First suppose there is an item with value at least OPT/λ. With probability $1/2$, we try to pick the best item, and we succeed with probability $1/e$. Thus, we get an $O(1)$ competitive ratio in this case.

In the other case, all the items have small contributions to the solution, i.e., less than OPT/λ. In this case, with constant probability, both $f(T_1)$ and $f(T_2)$ are at least $\mathrm{OPT}/4$. Hence, $\hat{\mathrm{OPT}}$ is a constant estimate for OPT. Let T' be the set of items picked by the algorithm in this case. If the sum of the weights of the items in T' is at least $1/2$, we are done, because all these items have (marginal) value density at least $\hat{\mathrm{OPT}}/6$, so $f(T') \geq (1/2) \cdot (\hat{\mathrm{OPT}}/6) = \hat{\mathrm{OPT}}/12 \geq \mathrm{OPT}/48$.

Otherwise, the total weight of T' is less than $1/2$. Therefore, there are items in T_2 that are not picked. There might be two reasons for this. There was not enough room in the knapsack, which means that the weight of the items in T_2 is more than $1/2$. However, there cannot be more than one such item in T_2, and the value of this item is not more than OPT/λ. Let z be this single big item, for future reference. Therefore, $f(T') \geq f(T_2) - \mathrm{OPT}/\lambda$ in this case.

The other case is when the ratios of some items from T_2 are less than $\hat{\mathrm{OPT}}/6$, and thus we do not pick them. Since they are all in T_2, their total weight is at most 1. Because of submodularity, the total loss due to these missed items is at most $\hat{\mathrm{OPT}}/6$. Submodularity and non-negativity of f then gives $f(T') \geq f(T_2) - f(\{z\}) - \hat{\mathrm{OPT}}/6 \geq \hat{\mathrm{OPT}} - \mathrm{OPT}\,\lambda - \hat{\mathrm{OPT}}/6 = O(\mathrm{OPT})$. □

3.5 The subadditive secretary problem

In this section, we prove Theorem 3.1.4 by presenting first a hardness result for approximation subadditive functions in general. The result applies in particular to our online setting. Surprisingly, the monotone subadditive function that we use here is *almost submodular*; see Proposition 3.5.3 below. Hence, our constant competitive ratio for submodular functions is nearly the most general we can achieve.

Definition 4 (Subadditive function maximization). *Given a nonnegative subadditive function f on a ground set U, and a positive integer $k \leq |U|$, the goal is to find a subset S of U of size at most k so as to maximize $f(S)$. The function f is accessible through a value oracle.*

3.5.1 Hardness result

In the following discussion, we assume that there is an upper bound of m on the size of sets given to the oracle. We believe this restriction can be lifted. If the function f is not required to be monotone, this is quite easy to have: simply let the value of the function f be zero for queries of size larger than m. Furthermore, depending on how we define the online setting, this may not be an *additional* restriction here. For example, we may not be able to query the oracle with secretaries that have already been rejected.

The main result of the section is the following theorem. It shows the subadditive function maximization is difficult to approximate, even in the offline setting.

Theorem 3.5.1. *There is no polynomial time algorithm to approximate an instance of subadditive function maximization within $\tilde{O}(\sqrt{n})$ of the optimum. Furthermore, no algorithm with exponential time 2^t can achieve an approximation ratio better than $\tilde{O}(\sqrt{n/t})$.*

First, we are going to define our *hard* function. Afterwards, we continue with proving certain properties of the function which finally lead to the proof of Theorem 3.5.1.

Let n denote the size of the universe, i.e., $n := |U|$. Pick a random subset $S^* \subseteq U$ by sampling each element of U with probability k/n. Thus, the expected size of S^* is k.

Define the function $g : U \to N$ as $g(S) := |S \cap S^*|$ for any $S \subseteq U$. One can easily verify that g is submodular. We have a positive r whose value will be fixed below. Define the final function $f : U \to N$ as

$$f(S) := \begin{cases} 1 & \text{if } g(S) = 0 \\ \lceil g(S)/r \rceil & \text{otherwise.} \end{cases}$$

It is not difficult to verify the subadditivity of f; it is also clearly monotone.

In order to prove the core of the hardness result in Lemma 3.5.2, we now let $r := \lambda \cdot \frac{mk}{n}$, where $\lambda \geq 1 + \sqrt{\frac{3tn}{mk}}$ and $t = \Omega(\log n)$ will be determined later.

Lemma 3.5.2. *An algorithm making at most 2^t queries to the value oracle cannot solve the subadditive maximization problem to within k/r approximation factor.*

Proof. Note that for any $X \subseteq U$, $f(X)$ lies between 0 and $\lceil k/r \rceil$. In fact, the optimal solution is the set S^* whose value is at least k/r. We prove that with high probability the answer to all the queries of the algorithm is one. This implies that the algorithm cannot achieve an approximation ratio better than k/r.

Assume that X_i is the i-th query of the algorithm for $1 \leq i \leq 2^t$. Notice that X_i can be a function of our answers to the previous queries. Define E_i as the event $f(X_i) = 1$. This is equivalent to $g(X_i) \leq r$. We show that with high probability all events E_i occur.

For any $1 \leq i \leq 2^t$, we have

$$\Pr\left[E_i | \bigcap_{j=1}^{i-1} E_j\right] = \frac{\Pr[\bigcap_{j=1}^{i} E_j]}{\Pr[\bigcap_{j=1}^{i-1} E_j]} \geq \Pr\left[\bigcap_{j=1}^{i} E_j\right] \geq 1 - \sum_{j=1}^{i} \overline{E_j}.$$

Thus, we have $\Pr[\cap_{i=1}^{2^t} E_i] \geq 1 - 2^t \sum_{i=1}^{2^t} \Pr[\overline{E_i}]$ from union bound. Next we bound $\Pr[E_i]$. Consider a subset $X \subseteq U$ such that $|X| \leq m$. Since the elements of S^* are picked randomly with probability k/n, the expected value of $X \cap S^*$ is at most mk/n. Standard application of Chernoff bounds gives

$$\Pr[f(X) \neq 1] = \Pr[g(X) > r] = \Pr\left[|X \cap S^*| > \lambda \cdot \frac{mk}{n}\right] \leq \exp\{-(\lambda - 1)^2 \frac{mk}{n}\} \leq \exp\{-3t\} \leq \frac{2^-}{n}$$

where the last inequality follows from $t \geq \log n$. Therefore, the probability of all E_i events occurring simultaneously is at least $1 - 1/n$. $\square$

Now we can prove the main theorem of the section.

Theorem 3.5.1. We just need to set $k = m = \sqrt{n}$. Then, $\lambda = \sqrt{3t}$, and the inapproximability ratio is $\Omega(\sqrt{\frac{n}{t}})$. Restricting to polynomial algorithms, we obtain $t := O(\log^{1+\epsilon} n)$, and considering exponential algorithms with running time $O(2^{t'})$, we have $t = O(t')$, giving the desired results. $\square$

In case the query size is not bounded, we can define $f(X) := 0$ for large sets X, and pull through the same result; however, the function f is no longer monotone in this case.

We now show that the function f is almost submodular. Recall that a function g is submodular if and only if $g(A) + g(B) \geq g(A \cup B) + g(A \cap B)$.

Proposition 3.5.3. *For the hard function f defined above, $f(A) + f(B) \geq f(A \cup B) + f(A \cap B) - 2$ always holds; moreover, $f(X)$ is always positive and attains a maximum value of $\tilde{\Theta}(\sqrt{n})$ for the parameters fixed in the proof of Theorem 3.5.1.*

Proof. The function $h(X) := g(X)/r$ is clearly submodular, and we have $h(X) \leq f(X) \leq h(X) + 1$. We obtain $f(A) + f(B) \geq h(A) + h(B) \geq h(A \cup B) + h(A \cap B) \geq f(A \cup B) + f(A \cap B) - 2$. $\square$

3.5.2 Algorithm

An algorithm that only picks the best item clearly gives a k-competitive ratio. We now show how to achieve an $O(n/k)$ competitive ratio, and thus by combining the two, we obtain an $O(\sqrt{n})$-competitive algorithm for the monotone subadditive secretary problem. This result complements our negative result nicely.

Partition the input stream S into $\ell := n/k$ (almost) equal-sized segments, each of size at most k. Randomly pick all the elements in one of these segments. Let the segments be denoted by $S_1, S_2, \ldots, S_\ell$. Subadditivity of f implies $f(S) \leq \sum_i f(S_i)$. Hence, the expected value of our solution is $\sum_i \frac{1}{\ell} f(S_i) \geq \frac{1}{\ell} f(S) \geq \frac{1}{\ell}\,\mathrm{OPT}$, where the two inequalities follow from subadditivity and monotonicity, respectively.

3.6 Conclusions and further results

In this paper, we consider the (non-monotone) submodular secretary problem for which we give a constant-competitive algorithm. The result can be generalized when we have a matroid constraint on the set that we pick; in this case we obtain an $O(\log^2 r)$-competitive algorithm where r is the rank of the matroid. However, we show that it is very hard to compete with the optimum if we consider subadditive functions instead of submodular functions. This hardness holds even for "almost submodular" functions; see Proposition 3.5.3.

One may consider special non-submodular functions which enjoy certain structural results in order to find better guarantees. For example, let $f(T)$ be the minimum individual value in T which models a bottle-neck situation in the secretary problem, i.e., selecting a group of k secretaries to work together, and the speed (efficiency) of the group is limited to that of the slowest person in the group (note that unlike the submodular case here the condition for employing exactly k secretaries is enforced.) In this case, we present a simple $O(k)$-competitive ratio for the problem as follows. Interview the first $1/k$ fraction of the secretaries without employing anyone. Let α be the highest efficiency among those interviewed. Employ the first k secretaries whose efficiency surpasses α.

Theorem 3.6.1. *Following the prescribed approach, we employ the k best secretaries with probability at least $1/e^2k$.*

Indeed we believe that this $O(k)$ competitive ratio for this case should be almost tight. One can verify that provided individual secretary efficiencies are far from each other, say each two consecutive values are farther than a multiplicative factor n, the problem of maximizing the expected value of the minimum efficiency is no easier than being required to employ all the k best secretaries. Theorem **??** in Appendix .3 provides evidence that the latter problem is hard to approximate.

Another important aggregation function f is that of maximizing the performance of the secretaries we employ: think of picking k candidate secretaries and finally hiring the best. We consider this function in Appendix **??** for which we present a near-optimal solution. In fact, the problem has been already studied, and an optimal strategy appears in [23]. However, we propose a simpler solution which features certain "robustness" properties (and thus is of its own interest): in particular, suppose we are given a vector $(\gamma_1, \gamma_2, \ldots, \gamma_k)$ such that $\gamma_i \geq \gamma_{i+1}$ for $1 \leq i < k$. Sort the elements in a set R of size k in a non-increasing order, say $a_1, a_2, \ldots, a_k$. The goal is to maximize the efficiency $\sum_i \gamma_i a_i$. The algorithm that we propose maximizes this more general objective obliviously; i.e., the algorithm runs irrespective of the vector γ, however, it can be shown the resulting solution approximates the objective for all vectors γ at the same time. The reader is referred to Appendix **??** for more details.

.1 Hardness Results

Here we show some matching hardness results to show that our algorithms are optimal unless $P = NP$. Surprisingly the problem we studied does not have better than $\log n$ approximation even in very simple cases, namely, one interval scheduling with nonuniform parallel machines, or multi-interval scheduling with only one processor.

It is proved in [13] that the multi-interval scheduling problem with only one processor and simple cost function is Set-Cover hard, and therefore the best possible approximation factor for this problem is $\log n$. We note that in the simple cost function the cost of an interval is equal to its length plus a fixed amount of energy (the restart cost). All previous work studies the problem with this cost function. In fact, Theorem 7 of [13] shows that the problem does not have a $o(\log N)$-approximation even when the number of time intervals of each job is at most 2 (each job has a set of time intervals in which it can execute).

Theorem .1.1. *It is NP-hard to approximate 2-interval gap scheduling within a $o(\log N)$ factor, where N is the size of input.*

Now we show that the one-interval scheduling problem, for which there exists a polynomial-time algorithm in [13], does not have any $o(\log N)$-approximation when only a subset of processors are capable of executing a job. Assume that each job has one time interval in which it can execute, and for each job, we have a subset of processors that can execute this job in its time interval, i.e., the other processors do not have necessary resources to execute the job. We also consider the generalized cost function in which the cost of an interval is not necessarily equal to its length plus a fixed amount. We call this problem *one-interval scheduling with nonuniform processors*.

Theorem .1.2. *It is NP-hard to approximate one-interval scheduling with nonuniform processors problem within a $o(\log N)$ factor, where N is the size of input.*

Proof. Like previous hardness results for these scheduling problems, we give an approximation-preserving reduction from Set Cover, which is not $o(\log n)$-approximable unless $\mathrm{P} = \mathrm{NP}$ [43]. Let $E = \{e_1, e_2, \ldots, e_n\}$ be the set of all elements in the Set-Cover instance. There are also m subsets of E, $S_1, S_2, \ldots, S_m$ in the instance. We construct our scheduling problem instance as follows. For each set S_j, we put a processor P_j in our instance. For each element e_i, we put a job j_i. Only jobs in set S_j can be done in processor P_j. The time interval of all jobs is $[1, n]$. The cost of keeping each processor alive during a time interval is 1. Note that the cost a time interval is not a function of its length in this case, i.e., the cost of an interval is almost equal to a fixed cost which might be the restart cost. So the optimum solution to our scheduling problem is a minimum size subset of processors in which we can schedule all jobs because we can assume that when a processor is alive in some time units, we can keep that processor alive in the whole interval $[1, n]$ (it does not increase our cost). In fact we want to find the minimum number of subsets among the input subset such that their union is E. This is exactly the Set Cover problem. □

.2 Polynomial-Time Algorithm for Prize-Collecting One-interval Gap Minimization Problem

The simple cost function version of our problem is studied in [9,13] as the gap-minimization problem. Each job has a time interval, and we want to schedule all jobs on P machines with the minimum number of gaps. (A gap is a maximal period of time in which a processor is idle, which can be associated with a restart for one of the machines.) There are many cases in which we can not schedule all jobs according to our limitation in resources: number of machines, deadlines, etc. So we define the prize-collecting version of this simple problem. Assume that each job has some value for us, and we get its value if we schedule it. We want to get the maximum possible value according to some cost limits. Formally, we want to schedule a subset of jobs with maximum total value and at most g gaps. The variable g is given in the input. Now we show how to adapt the sophisticated dynamic program in [13] to solve this problem.

Theorem .2.1. *There is a (n^7p^5g)-time algorithm for prize-collecting p-processor gap scheduling of n jobs with budget g, the number of gaps should not exceed g.*

Proof. In the proof of Theorem 1 of [13], C_{t_1,t_2,k,q,l_1,l_2} is defined to be the number of gaps in the optimal solution for a subproblem defined there. If we define $C'_{t_1,t_2,k,q,l_1,l_2,g'}$ to be the maximum value we can get in the same subproblem using at most $g' \leq g$ gaps, we can update this new dynamic program array in the same way. The rest of the proof is similar; we just get an extra g in the running time. □

.3 Omitted proofs and theorems

Lemma 3.2.1. Let $k := |B| - |A|$. Then, define in an arbitrary manner sets $\{B_i\}_{i=0}^k$ such that

- $B_0 = A$,
- $|B_i \setminus B_{i-1}| = 1$ for $i : 1 \leq i \leq k$,
- and $B_k = B$.

Let $b_i := B_i \setminus B_{i-1}$ for $i : 1 \leq i \leq k$. We can write $f(B) - f(A)$ as follows

$$\begin{aligned} f(B) - f(A) &= \sum_{i=1}^{k} [f(B_i) - f(B_{i-1})] \\ &= \sum_{i=1}^{k} [f(B_{i-1} \cup \{b_i\}) - f(B_{i-1})] \\ &\leq \sum_{i=1}^{k} [f(A \cup b_i) - f(A)], \end{aligned}$$

where the last inequality follows from the non-increasing marginal profit property of submodular functions. Noticing that $b_i \in B \setminus A$ and they are distinct, namely $b_i \neq b_{i'}$ for $1 \leq i \neq i' \leq k$, finishes the argument. $\square$

Bibliography

[1] *The submodular secretary problem and its extensions*, 2009.

[2] A. A. Ageev and M. I. Sviridenko. An 0.828-approximation algorithm for the uncapacitated facility location problem. *Discrete Appl. Math.*, 93(2-3):149–156, 1999.

[3] Miklos Ajtai, Nimrod Megiddo, and Orli Waarts. Improved algorithms and analysis for secretary problems and generalizations. *SIAM J. Discrete Math.*, 14(1):1–27, 2001.

[4] Arash Asadpour, Hamid Nazerzadeh, and Amin Saberi. Stochastic submodular maximization. In *Proceedings of the 4th International Workshop on Internet and Network Economics (WINE)*, pages 477–489, 2008.

[5] John Augustine, Sandy Irani, and Chaitanya Swamy. Optimal power-down strategies. *SIAM J. Comput.*, 37(5):1499–1516, 2008.

[6] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. A knapsack secretary problem with applications. In *Proceedings of the 10th International Workshop on Approximation and Combinatorial Optimization (APPROX'07)*, pages 16–28, 2007.

[7] Moshe Babaioff, Nicole Immorlica, David Kempe, and Robert Kleinberg. Online auctions and generalized secretary problems. *SIGecom Exch.*, 7(2):1–11, 2008.

[8] Moshe Babaioff, Nicole Immorlica, and Robert Kleinberg. Matroids, secretary problems, and online mechanisms. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms (SODA'07)*, pages 434–443, 2007.

[9] Philippe Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 364–367, New York, NY, USA, 2006. ACM.

[10] Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *Integer Programming and Combinatorial Optimization, 12th International IPCO Conference (IPCO'07)*, pages 182–196, 2007.

[11] Gerard Cornuejols, Marshall Fisher, and George L. Nemhauser. On the uncapacitated location problem. In *Studies in integer programming (Proc. Workshop, Bonn. 1975)*, pages 163–177. Ann. of Discrete Math., Vol. 1. North-Holland, Amsterdam, 1977.

[12] Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: an analytic study of exact and approximate algorithms. *Management Sci.*, 23(8):789–810, 1976/77.

[13] Erik D. Demaine, Mohammad Ghodsi, Mohammad Taghi Hajiaghayi, Amin S. Sayedi-Roshkhar, and Morteza Zadimoghaddam. Scheduling to minimize gaps and power consumption. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, pages 46–54, New York, NY, USA, 2007. ACM.

[14] E. B. Dynkin. The optimum choice of the instant for stopping a markov process. *Sov. Math. Dokl.*, 4:627–629, 1963.

[15] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. In *Combinatorial Structures and their Applications (Proc. Calgary Internat. Conf., Calgary, Alta., 1969)*, pages 69–87. Gordon and Breach, New York, 1970.

[16] Eyal Even Dar, Vahab S. Mirrokni, S. Muthukrishnan, Yishay Mansour, and Uri Nadav. Bid optimization for broad match ad auctions. In *WWW '09: Proceedings of the*

18th international conference on World wide web, pages 231–240, New York, NY, USA, 2009. ACM.

[17] U. Feige and M. X. Goemans. Approximating the value of two power proof systems, with applications to max 2sat and max dicut. In *Proceedings of the 3rd Israel Symposium on the Theory of Computing Systems (ISTCS'95)*, page 182, Washington, DC, USA, 1995. IEEE Computer Society.

[18] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[19] Uriel Feige. On maximizing welfare when utility functions are subadditive. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC'06)*, pages 41–50. ACM, 2006.

[20] Uriel Feige, Vahab S. Mirrokni, and Jan Vondrák. Maximizing non-monotone submodular functions. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 461–471. IEEE Computer Society, 2007.

[21] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions. II. *Math. Programming Stud.*, (8):73–87, 1978. Polyhedral combinatorics.

[22] P. R. Freeman. The secretary problem and its extensions: a review. *Internat. Statist. Rev.*, 51(2):189–206, 1983.

[23] John P. Gilbert and Frederick Mosteller. Recognizing the maximum of a sequence. *J. Amer. Statist. Assoc.*, 61:35–73, 1966.

[24] Kenneth S. Glasser, Richard Holzsager, and Austin Barron. The d choice secretary problem. *Comm. Statist. C—Sequential Anal.*, 2(3):177–199, 1983.

[25] Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. Assoc. Comput. Mach.*, 42(6):1115–1145, 1995.

[26] Mohammad T. Hajiaghayi, Robert Kleinberg, and Tuomas Sandholm. Automated online mechanism design and prophet inequalities. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence (AAAI'07)*, pages 58–65, 2007.

[27] Mohammad Taghi Hajiaghayi, Robert Kleinberg, and David C. Parkes. Adaptive limited-supply online auctions. In *Proceedings of the 5th ACM conference on Electronic Commerce (EC '04)*, pages 71–80, New York, NY, USA, 2004. ACM.

[28] Eran Halperin and Uri Zwick. Combinatorial approximation algorithms for the maximum directed cut problem. In *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms (SODA'01)*, pages 1–7, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[29] Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859 (electronic), 2001.

[30] Nicole Immorlica, Robert D. Kleinberg, and Mohammad Mahdian. Secretary problems with competing employers. In *Proceedings of the 2nd Workshop on Internet and Network Economics (WINE'06)*, volume 4286, pages 389–400. Springer, 2006.

[31] Sandy Irani, Sandeep K. Shukla, and Rajesh Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3(4), 2007.

[32] Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777 (electronic), 2001.

[33] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[34] Subhash Khot, Guy Kindler, Elchanan Mossel, and Ryan O'Donnell. Optimal inapproximability results for max-cut and other 2-variable csps? In *Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS'04)*, pages 146–154, Washington, DC, USA, 2004. IEEE Computer Society.

[35] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

[36] Robert 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.

[37] Nitish Korula and Martin Pal. Algorithms for secretary problems on graphs and hypergraphs. *CoRR*, abs/0807.1139, 2008.

[38] Ariel Kulik, Hadas Shachnai, and Tami Tamir. Maximizing submodular set functions subject to multiple linear constraints. In *SODA '09: Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 545–554, Philadelphia, PA, USA, 2009. Society for Industrial and Applied Mathematics.

[39] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discrete Math.*, 23(4):2053–2078, 2010.

[40] L. Lovász. Submodular functions and convexity. In *Mathematical programming: the state of the art (Bonn, 1982)*, pages 235–257. Springer, Berlin, 1983.

[41] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. I. *Math. Programming*, 14(3):265–294, 1978.

[42] Maurice Queyranne. A combinatorial algorithm for minimizing symmetric submodular functions. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms (SODA'95)*, pages 98–101, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[43] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *STOC*, pages 475–484, 1997.

[44] Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *J. Combin. Theory Ser. B*, 80(2):346–355, 2000.

[45] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004.

[46] R. J. Vanderbei. The optimal choice of a subset of a population. *Math. Oper. Res.*, 5(4):481–486, 1980.

[47] Jan Vondrák. Symmetry and approximability of submodular maximization problems. In *Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09)*. IEEE Computer Society, 2009. to appear.

[48] John G. Wilson. Optimal choice and assignment of the best m of n randomly arriving items. *Stochastic Process. Appl.*, 39(2):325–343, 1991.