CORE
Metadata, citation and similar papers at core.ac.uk
Provided by DSpace@MIT

Vignettes on Robust Combinatorial Optimization

by

Rajan Udwani

B.Tech, Indian Institute of Technology Bombay (2012)

Submitted to the Sloan School of Management
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Operations Research

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

© Massachusetts Institute of Technology 2018. All rights reserved.

Author .....
Sloan School of Management
August 10, 2018

Certified by .....
James B. Orlin
E. Pennell Brooks Professor of Management
Sloan School of Management
Thesis Supervisor

Certified by .....
Andreas S. Schulz
Alexander von Humboldt Professor
Department of Mathematics and School of Management
Technical University of Munich
Thesis Supervisor

Accepted by .....
Dimitris Bertsimas
Boeing Leaders for Global Operations
Co-director, Operations Research Center

Vignettes on Robust Combinatorial Optimization

by

Rajan Udwani

Submitted to the Sloan School of Management
on August 10, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Operations Research

Abstract

In this thesis, we design and analyze algorithms for robust combinatorial optimization in various settings.

First, we consider the problem of simultaneously maximizing multiple objectives, all monotone submodular, subject to a cardinality constraint. We focus on the case where the number of objectives is super-constant yet much smaller than the cardinality of the chosen set. We propose several algorithms (including one with the best achievable asymptotic guarantee for the problem). Experiments on synthetic data show that a heuristic based on our more practical and fast algorithm outperforms current practical algorithms in all cases considered.

Next, we study the problem of robust maximization of a single monotone submodular function in scenarios where after choosing a feasible set of elements, some elements from the chosen set are adversarially removed. Under some restriction on the number of elements that can be removed, we give the first constant factor approximation algorithms as well as the best possible asymptotic approximation in certain cases. We also give a black box result for the much more general setting of deletion-robust maximization subject to an independence system.

Lastly, we consider a robust appointment scheduling problem where the goal is to design simple appointment systems that try to achieve both high server utilization as well as short waiting times, under uncertainty in job processing times. When the order of jobs is fixed and one seeks to find optimal appointment duration for jobs, we give a simple heuristic that achieves the first constant factor (2) approximation. We also give closed form optimal solutions in various special cases that supersede previous work. For the setting where order of jobs is also flexible and under-utilization costs are homogeneous, it was previously shown that an EPTAS exists. We instead focus on simple and practical heuristics, and find a ratio based ordering which is 1.0604 approximate, improving on previous results for similarly practical heuristics.

Thesis Supervisor: James B. Orlin
Title: E. Pennell Brooks Professor of Management
Sloan School of Management

Thesis Supervisor: Andreas S. Schulz
Title: Alexander von Humboldt Professor
Department of Mathematics and School of Management
Technical University of Munich

Acknowledgments

There were several reasons that led me to pursue a PhD, one of which was the challenge of it. And like any challenging endeavour, I could only succeed because of the guidance, support, friendship and help from people around me.

First and foremost, to my advisers Prof. James B. Orlin and Prof. Andreas S. Schulz. I am deeply thankful for all your support over the years. You taught me everything I know about research. Looking back, I consider myself very lucky that you decided to take a chance on me after a broken Skype call. Your counsel and faith in me during the early years was key in building a foundation for my work, and essential in helping me gain some self-confidence. Your flexibility and encouragement to let me work on problems that interested me was another gift that always kept me motivated and interested in what I was doing. I apologize for all my various last minute antics, and am sincerely thankful for your patience with me through the years. I know I have a lot to learn yet, and I promise to continue to do so. It has been, and always will be, a privilege.

I would then like to thank Prof. Dimitris Bertsimas. You have always been a grandfatherly figure in the ORC, someone who cares deeply for the wellbeing of every student. Your presence in the ORC makes it not only a secure but also a lively environment! Thank you for all the advice you have given over the years, and for everything you have taught me. In a similar vein, I am also thankful to all the instructors who taught me over the years. In particular, Prof. David Karger for teaching an amazing course on Advanced Algorithms. Equally importantly, I would also like to thank the instructors for all the extra-curricular courses at MIT, as well the organizers and the broader MIT community that makes these things possible! Over the years, I learnt (or at least tried to learn) to snowboard, sail, row, sketch, do pottery, swim, SCUBA dive, surf, wind surf, ice skate, shoot a rifle, do kayak rolls, ballroom dance and also learnt archery, fencing, self-defense, glass blowing (!) and ASL. It was for me the biggest surprise, and a major part of what made this place truly truly special and unique. I am also thankful to all the seminar speakers over

the years. Listening to a broad range of talks and the opportunity to talk to speakers in person was invaluable. In fact, an in person remark by Jan Vondrák, who once spoke at the ORC seminar, helped extend a result in Chapter 4. These things would also be impossible without the involvement and support of the broader community, student volunteers, faculty and staff, who are at the heart of it all.

Next, I would like to thank my friends, for all the sparkling conversations, for everything you have taught me and introduced me to over the years, for tolerating my taste in music and perhaps even enjoying some of it, for lending me a friendly ear ever so often and most importantly, for your companionship. Your presence in my life here made it wholesome and meaningful, so thank you Pritish & Apoorvaa, Prashant, Matin, Daniel & Shiyun, Nishanth, SaiG, Murali, Reetik, Ananth, Srivatsan, Divya, Somya, Jackie & Will, Yiannis, Lee, Champ, Sharon & Ted, Sid, Jay, Anjuli & Joshua, Peng, Swati, Chiwei, Wang, He, Yongwhan, Ankit, Saurabh, Vashist, Gowtham, Hari, Miles, Illias, Lennart, Xiaoyue, Kris, Clark, Siong, Berk, Joel, Cong, Louis, Michael, Eeshit, Arkopal, Matt, Tamanna, Fiorella, Luisa, Tiziana, Audrey, Michel, Benedikt, Jari, Roxanna, Benedetta, Jannik ... and many others. Thanks also to my IBM Zurich mentors Marco Laumanns and Jacint Szabo, as well as Jamila for helping me feel at home during my visit to TU Munich. Thanks to all my numerous, numerous teachers and my friends from my life before MIT, I would never have gotten here without you. And thank you Aditi, for what I hope will be a new chapter in my (our) life, and for turning the last and otherwise frustrating months into a true wonder.

Finally, I would like to dedicate this thesis to my family. To my grandparents, aunts and uncles, who helped raise me and taught me many things. Most importantly though, the three anchors of my life; my parents and my little sister. They are the foundation on which I build everything, knowing that even if it all collapses they will always be there for me.

I gratefully acknowledge partial support from ONR Grant N00014-17-1-2194 for the duration of my PhD

Contents

List of Figures

Chapter 1

Introduction

Advances in computing, communication and data collection have created opportunities to optimize almost every aspect of modern life. To leverage these opportunities, one needs good algorithms. For absent well designed algorithms, problems that are currently solved in seconds would take eons even on a modern computer. But what constitutes a good algorithm? Two important criteria are accuracy – output should be feasible and *close* to optimal, and time – execution should take as little time as possible. Indeed, designing fast and approximately optimal algorithms continues to be a major area of study in Computer Science, Operations Research and many other fields today. In this thesis, we design and analyze algorithms for various combinatorial optimization problems, with a focus on both good approximation guarantee and practical runtime.

The optimization problems we consider are *robust* reformulations of settings related to two classical problems in combinatorial optimization – monotone submodular function maximization [NWF78, NW78], and scheduling [Smi56]. The motivation behind considering such reformulations stems from the fact that traditional optimization models often do not account for the uncertainty in model parameters which are input to algorithms. As a result, algorithms that are well designed in theory do not work well in practice. A traditional way of accounting for such uncertainty has been to remodel the problem through the lens of *stochastic* optimization. By approaching the issue from a different perspective, *robust* optimization has emerged as a versatile and

tractable alternative in the last couple of decades [BTEGN09, BS04a, BS03, BS04b]. In robust optimization, one models uncertainty in a deterministic sense by positing that uncertain parameters belong to a structured *uncertainty set*. Given this assumption, one seeks a solution that is optimal in the worst possible realization of uncertainty from the assumed set of possibilities.

We study three different instances of robust combinatorial optimization, in each of which the uncertainty manifests solely through the objective function. In all three settings, we find algorithms that have good theoretical guarantees and are also fast (nearly linear time) and easy to implement. We now introduce each of these settings and discuss motivations behind the specific formulations that we consider here. The first two settings involve maximization of monotone submodular functions and are presented in Section 1.1, followed by an appointment scheduling problem in Section 1.2.

1.1 Maximizing Monotone Submodular Functions under Uncertainty

First, recall that a set function $f : 2^N \to \mathbb{R}$ on the ground set N is called submodular when $f(A + a) - f(A) \leq f(B + a) - f(B)$ for all $B \subseteq A \subseteq N$ and $a \in N \setminus A$. The function is monotone when $f(B) \leq f(A)$ for all $B \subseteq A$. We assume $f(\emptyset) = 0$ w.l.o.g., and combined with monotonicity this implies that the function is non-negative.

Many well known objectives in combinatorial optimization exhibit two common properties: the marginal value of any given element is non-negative and it decreases as more and more elements are selected. The notions of submodularity and monotonicity nicely capture this property, resulting in the appearance of constrained monotone submodular maximization in a wide and diverse array of modern applications in machine learning and optimization, including feature selection ([KG05, TCG$^+$09]), network monitoring ([LKG$^+$07]), news article recommendation ([EAVSG09]), sensor placement and information gathering ([OUS$^+$08, GKS05, KGGK06, KLG$^+$08]), viral

marketing and influence maximization ([KKT03, HK16]), document summarization ([LB11]) and crowd teaching ([SB14]). In many of these settings, the objective is often uncertain and traditional algorithms have been demonstrated to perform poorly in practice (discussed further in Chapters 3 and 4). We study two formulations that deal with different forms and sources of uncertainty.

1.1.1 Multi-objective Maximization

Here we are interested in scenarios where multiple objectives, all monotone submodular, need to be maximized simultaneously, subject to a cardinality constraint. This problem has an established line of work in both machine learning [KMGG08] and the theory community [CVZ10]. As an example application, in robust experimental design one often seeks to maximize a function f_θ, which is monotone submodular for every value of θ. The function is very sensitive to the choice of θ but the parameter is unknown a priori and estimated from data. Therefore, one possible approach to finding a robust solution is to maximize the function $min_{\theta\in\Theta} f_\theta(.)$, where Θ is a set that captures the uncertainty in θ. If Θ is assumed to be a finite set of discrete values, like [KMGG08], we have an instance of multi-objective monotone submodular maximization. More generally, we consider the following problem,

$$MO_1 : \max_{A\subseteq N, |A|\leq k} \min_{i\in\{1,2,\ldots,m\}} f_i(A),$$

where $f_i(.)$ is monotone submodular for every $i \in \{1, \ldots, m\}$. The problem also has an alternative formulation due to [CVZ10], which we discuss is Chapter 3. Broadly speaking, there are two ways in which this has been applied:

When there are several natural criteria that need to be simultaneously optimized: such as in network monitoring, sensor placement and information gathering [OUS+08, LKG+07, KLG+08, KMGG08]. For example in the problem of intrusion detection [OUS+08], one usually wants to maximize the likelihood of detection while also minimizing the time until intrusion is detected, and the population affected by intrusion. The first objective is often monotone submodular and the latter objec-

tives are monotonically decreasing supermodular functions [LKG+07, KLG+08]. As a result the problem is often formulated as an instance of cardinality constrained maximization with a small number of submodular objectives.

When looking for solutions robust to the uncertainty in objective: such as in feature selection [KMGG08, GR06], variable selection and experimental design [KMGG08], robust influence maximization [HK16]. In these cases, there is often inherently just a single submodular objective which is highly prone to uncertainty either due to dependence on a parameter that is estimated from data, or due to multiple possible scenarios that each give rise to a somewhat different objective. Therefore one often seeks to optimize over the worst case realization of the uncertain objective, resulting in an instance of multi-objective submodular maximization.

In some applications the number of objectives is given by the problem structure and can be larger even than the cardinality parameter. However, in applications such as robust influence maximization, variable selection and experimental design, the number of objectives is a design choice that trades off optimality with robustness.

The regime $m = o(k)$ includes several of the applications we referred to above. For instance, in network monitoring and sensor placement, the number of objectives is usually a small constant [KMGG08, LKG+07]. For robust influence maximization, the number of objectives depends on the underlying uncertainty but often ends up being small [HK16]. And in settings like variable selection and experimental design [KMGG08], where the number of objectives considered is a design choice. We focus on this regime ($m = o(k)$) and show three algorithmic results with asymptotic approximation guarantees: (i) A practical $O(\frac{n}{\delta^3} \log m \log \frac{n}{\delta})$ time algorithm that has an asymptotic guarantee of $(1 - 1/e)^2 - \delta$. (ii) A $(1 - 1/e)$ approximation with increased runtime of $O(n^8)$ and (iii) A deterministic $(1 - 1/e) - \epsilon$ approximation for constant m, with runtime kn^{m/ϵ^4}. All other algorithms are randomized. Previously, [KMGG08] showed that MO_1 is inapproximable (to within any non-trivial factor) when $m = \Omega(k)$, unless P=NP. They also gave a bi-criterion approximation that violates the cardinality constraint. In contrast, [CVZ10] gave a $(1 - 1/e) - \epsilon$ approximation with query complexity $O(n^8 + n^{m/\epsilon^3})$. While this offers the best possible

approximation for constant m, the algorithm is infeasible to use in practice. Note that our results imply a sharp transition from constant factor approximability for $m = o(k)$ to inapproximability for $m = \Omega(k)$.

1.1.2 Deletion-robust Maximization

Consider the problem of maximizing a single monotone submodular function subject to a cardinality constraint,

$$SO : \max_{A \subseteq N, |A| \leq k} f(A).$$

Now, suppose that given our choice of a set A with cardinality k for the above maximization problem, τ elements are adversarially removed from A. Then the problem of finding a set that has maximum function value subject to removal of τ elements can be formulated as,

$$DRO : \max_{A \subseteq N, |A| \leq k} \min_{Z \subseteq A, |Z| \leq \tau} f(A - Z).$$

Note that the parameter τ controls the degree of robustness of the chosen set since the larger τ is, the larger the size of subset Z that can be adversarially removed from the chosen set A. For $\tau = 0$, DRO reduces to SO. This problem was also introduced in [KMGG08], and below we discuss two practical scenarios of SO that motivate DRO.

Sensor Placement [GKS05, KGGK06, LKG+07, KMGG08]: Given a large number of locations, we would like to place a relatively small number sensors at certain locations so as to maximize the *coverage*. Many commonly used coverage functions measure the cumulative information gathered in some form, and are thus monotone (more sensors is better) and submodular (decreasing marginal benefit of adding a new sensor).

As highlighted in [KMGG08], it is important to ask what happens if some sensors were to fail. Will the remaining sensors have good coverage regardless of which sensors failed, or is a small crucial subset responsible for most of the coverage?

Feature Selection [TCG+09, LWK+13, GR06, KMGG08]: In many machine learning models, adding a new feature to an existing set of features always improves the modeling power (monotonicity) and the marginal benefit of adding a new feature decreases as we consider larger sets (submodularity). Given a large set of features, we would like to select a small subset such that, we reduce the problem dimensionality while retaining most of the information.

However, as discussed in [GR06, KMGG08], in situations where the nature of underlying data is uncertain, leading to non-stationary feature distributions, it is important to not have too much dependence on a few features. Taking a concrete example from [GR06], in document classification, features may take not standard values due to small sample effects or in fact, the test and training data may come from different distributions. In other cases, a feature may even be deleted at some point, due to input sensors failures for example. Thus, similar questions arise here too and we would like to have an 'evenly spread' dependence on the set of chosen features.

DRO addresses the concerns above and in others applications, by seeking a set robust to the worst case manifestation of uncertainty. A natural variation is to optimize the average case failure scenario [GK11]. However, this is not suitable for some applications. For instance, we may have no prior on the failure/deletion mechanism. Moreover, in critical applications such as sensor placement for outbreak detection [KMGG08, LKG+07], we want protection against the worst case.

Here we give the first constant factor guarantees for *DRO* with combinatorial, 'greedy like' algorithms. To ease presentation, we usually ignore factors of the form $\left(1-\frac{O(1)}{k}\right)$ in the approximation guarantees but note that most of the results presented here are asymptotic in k. We first focus on $\tau = 1$, for which we propose a fast and practical 0.5547 approximation and later an asymptotically $(1 - 1/e) - 1/\Theta(m)$ algorithm (with runtime exponential in m, which is an input parameter). Then using ideas from the $\tau = 1$ case and our multi-objective maximization algorithm for $m = o(k)$, we give a $(1 - 1/e)$ approximation for $\tau = \frac{o(\log k)}{\log \log k}$. This algorithm gives the best possible approximation guarantee (asymptotically), but is impractical.

So, we give a fast algorithm but with worse approximation guarantee – 0.387. The algorithm is designed for $\tau = o\big(\sqrt{\frac{k}{c(k)}}\big)$ and gives a guarantee of $0.387\big(1 - \frac{1}{\Theta(c(k))}\big)$, where $c(k) \xrightarrow{k\to\infty} \infty$ is an input parameter that governs the trade off between how large τ can be and how fast the guarantee converges to 0.387. Finally, for more general constraints where we seek deletion-robust solutions that belong to an Independence system [1], we extend some of the ideas from the cardinality case into an enumerative procedure with runtime scaling as $n^{\tau+1}$. This yields an $\alpha/(\tau+1)$ approximation using an α approximation algorithm for $\tau = 0$ as a subroutine.

1.2 Robust Appointment Scheduling

Consider the problem of scheduling appointments in service operations where customers are served sequentially by a single server. Service times of customers are uncertain, and we wish to assign time slots for serving the customers in advance. A key practical setting where this problem arises is in health care services, where there are numerous instances that require efficient scheduling of appointments, such as scheduling outpatient appointments in primary care and specialty clinics, and scheduling surgeries for operating rooms.

Modern health care involves several high-cost devices and facilities such as MRI installations, CT scanners and operation rooms as well as highly trained personnel. An appointment scheduling system must ensure high utilization of these resources, while simultaneously offering small wait times and high quality of service to patients. For instance, if one assigns a very small time interval for a surgery, then a delay in the surgery could delay the start of the next surgery and so on. The costs and inconvenience due to delays, for both the patients and the staff, are captured in the *overage* cost of that surgery. On the other hand, if a surgery is assigned a large time duration but finishes much sooner, the hospital incurs an *underage* cost in the form of idle operation room and equipments until the next surgery commences. We

[1] An Independence system $\mathcal{I}$ over a ground set N is a collection of subsets such that the empty set belongs to $\mathcal{I}$, and every subset of an independent set (a set that belongs to $\mathcal{I}$) is also independent.

would like to design an appointment schedule that can achieve the desired trade-off between overage and underage costs. As discussed in [MRZ14], appointment systems in health care are often more involved and include a two-stage scheduling process. A preliminary booking stage where patients and surgeons select preferred dates and time windows for their appointments or surgeries. Then, given a group of appointments booked within a day (or equivalent a schedule block in the first stage), each appointment is assigned required resources (e.g., different operating rooms and surgeons) and alloted a starting time. The latter step is typically performed a few days in advance of the appointment dates from the first step, and it is this second step that our formulation addresses.

We consider a robust formulation of this problem that was first introduced in [MSS14]. For jobs $i \in \{1, \ldots, n\}$, let u_i, o_i denote the per unit time underage and overage costs respectively. Suppose that service time t_i for job i takes a value in the uncertainty set $[p_i - \hat{\delta}_i, p_i + \delta_i]$. For now let us assume that jobs are scheduled for service in a fixed order, which is the reverse order of their index. So job n is served first and job 1 last. We would like to appoint starting times A_i for $i \in \{0, n-1\}$. With C_i denoting the service completion time of job i, the problem can be formulated as,

$$RAS : \min_{A_0,\ldots,A_{n-1}} \quad \max_{t_i \in [p_i - \hat{\delta}_i, p_i + \delta_i] \, \forall i \in [n]} \quad \sum_{i=1}^{n} \max\{o_i(C_i - A_{i-1}), u_i(A_{i-1} - C_i)\}.$$

We further consider the problem where the order of jobs is not fixed a priori and also a part of the decision process, and call this the $RASS$ problem (Robust Appointment Scheduling and Sequencing).

Many existing models in the literature for appointment scheduling consider stochastic formulations of the problem. There are several issues related to solving these models in practice that [MSS14, MRZ14] discuss in great detail. For example, the models assume full knowledge of the distributions and the proposed algorithms often use sophisticated subroutines or sometimes lack analytical guarantees. The robust formulation and algorithms we consider here circumvent these issues. Previously

[MSS14] focused on the special case of RAS where underage costs u_i are identical/homogeneous for all i. We consider the general case and give a compact LP that achieves the first constant factor (2) approximation. We also give closed form optimal solutions in various special cases that supersede previous work. For the case of $RASS$ (where order of patients is interchangeable/flexible) with homogeneous underage costs, we show a simple ratio based heuristic that achieves a β approximation where $1.06036 < \beta < 1.06043$, improving the $2+\epsilon$ approximation in [MSS14]. Finally, for the fully general case of flexible job order and arbitrary costs, we show a $\Theta(n)$ approximation.

1.3 Outline

The outline for the rest of the thesis is as follows.

In Chapter 2, we establish some common background for Chapters 3 & 4 by reviewing relevant previous work on monotone submodular function maximization.

In Chapter 3, we focus on the problem of multi-objective maximization of monotone submodular functions subject to a cardinality constraint. We show that as long as the number of functions is much smaller than the cardinality parameter, the problem is approximable up to a factor of $(1 - 1/e) \approx 0.632$. We also give a fast and practical algorithm with guarantee approaching $(1 - 1/e)^2 \approx 0.399$. Finally, we offer a derandomization result for constant m and outline some related open problems.

In Chapter 4, we focus on a robust formulation of the classical cardinality constrained single objective monotone submodular function maximization problem. Here we seek solutions that are robust to adversarial removal of some number of elements. We show the first constant factor approximation algorithms for the problem. In fact, when the number of elements removed is suitably smaller than the cardinality parameter, our algorithm approaches the best possible approximation guarantee of $(1 - 1/e)$. Importantly, we also give a fast and practical algorithm with guarantee 0.387. Finally, we present a generic black box result for robust maximization of a monotone submodular function subject to more general Independence system con-

straint. We conclude with some open problems and a brief summary of more recent work by others that addresses some of the open questions.

In Chapter 5, we study the robust version of an appointment scheduling problem relevant to service systems in health care. We consider two different settings. In the first setting, the order of jobs is fixed and the goal is to allocate time for the service of each job. Previous work considered a special case of the problem and gave a closed form optimal solution. We consider the general case and offer the first constant factor guarantee via a simple LP. Moreover our algorithm is optimal and reduces to a closed form optimal solution for special cases that supersede previous work. In the second setting, the order of jobs is flexible and also a part of the decision process. Here we show a simple ratio based heuristic is within $\approx 6\%$ of the optimal for an important special case, and we show a $\Theta(n)$ approximation for the much harder general case.

Finally, in Chapter 6 we finish with some concluding remarks.

Chapter 2

Monotone Submodular Function Maximization

In this chapter we review existing work on the maximization of monotone submodular functions [1]. Recall, the problem of maximizing a single monotone submodular function subject to a cardinality constraint,

$$SO := \max_{A \subseteq N, |A| \leq k} f(A).$$

Notice that if we remove the cardinality constraint, the unconstrained problem has a trivial optimal solution: the ground set N. Therefore, SO is perhaps the simplest non-trivial problem one could study in this setting. The first theoretical result for this problem goes back to [NWF78, NW78], where they showed that the *greedy algorithm* (which we review here) gives a guarantee of $(1 - 1/e)$, and this is best possible in the value-oracle model. Later, [Fei98] showed that this is also the best possible approximation unless P=NP. While this settled the hardness and approximability of the problem, finding faster approximations remained an open line of inquiry. Notably, [BV14] found a faster algorithm for SO that improved the quadratic $O(nk)$ query complexity of the classical greedy algorithm to nearly linear complexity, by trading

[1] Recall, a set function $f : 2^N \to \mathbb{R}$ on the ground set N is submodular if, $f(A+a)-f(A) \leq f(B+a)-f(B)$ for all $B \subseteq A \subseteq N$ and $a \in N \setminus A$. The function is monotone if $f(B) \leq f(A)$ for all $B \subseteq A$

off on the approximation guarantee. This was further improved by [MBK+15].

For the more general problem $\max_{A \in \mathcal{I}} f(A)$, where $\mathcal{I}$ is the collection of independent sets of a matroid [2]; the greedy algorithm is only $\frac{1}{2}$ approximate [NWF78]. In a breakthrough, [CCPV11, Von08] achieved a tight $(1 - 1/e)$ approximation by (approximately) maximizing the *multilinear extension* of submodular functions, followed by suitable rounding. Based on this framework, tremendous progress was made over the last decade for a variety of different settings [CCPV11, Von08, FNS11, Von13, VCZ11, CVZ10, DV12]. In the rest of this chapter, we review some of these algorithms and their analysis where relevant. Our review is by no means comprehensive, but geared towards enabling a better understanding of our contributions in Chapters 3 and 4.

2.1 The Greedy Algorithm and Variants

In this section we discuss two algorithms for SO, the greedy algorithm [NWF78, NW78] and the faster thresholding algorithm [BV14]. But first, we introduce some notation which will be useful in Chapters 3 and 4 as well.

Recall, we use N to denote our ground set of n elements. Given set A define the marginal increase in value of function f due to inclusion of set X as,

$$f(X|A) = f(A \cup X) - f(A).$$

We generally refer to singleton sets without braces $\{\}$ and use $+, -$ and $\cup, \setminus$ interchangeably. Let $\beta(\alpha) = 1 - \frac{1}{e^{\alpha}} \in [0, 1]$ for $\alpha \geq 0$. The function increases monotonically with α and note that $\beta(\alpha) \leq 1 - (1 - \frac{1}{k})^{\alpha k}$. This function appears naturally in our analysis and will be useful for expressing approximation guarantees. In fact, as shown in [NWF78, NW78], the natural *greedy algorithm* (below) achieves the best possible guarantee of $\beta(1) = (1 - 1/e)$.

[2] A matroid $\mathcal{M}$ over a ground set N is given by a collection of independent sets $\mathcal{I}$ such that the empty set belongs to $\mathcal{I}$, and every subset of an independent set (a set that belongs to $\mathcal{I}$) is also independent. Further, if A and B are two independent sets and $|A| > |B|$, then there exists $x \in A \setminus B$ such that $B + x \in \mathcal{I}$.

Algorithm 1 Greedy Algorithm

1: Input: k, f and initialize $A = \emptyset$

2: **while** $|A| < k$ **do** $A = A + \underset{x \in N-A}{\operatorname{argmax}} f(x|A)$

3: Output: A

Starting with $\emptyset$, at each step the algorithm chooses an element that adds the maximum marginal value to the current set, until k elements are chosen. We use the terms Algorithm 1 and the greedy algorithm interchangeably, both to refer to the algorithm above. Below we state and prove a generalized version of the approximation guarantee shown in [NWF78, NW78].

Lemma 1. *[NWF78, NW78] Given ground set N, monotone submodular function f, let OPT denote an optimal solution of SO with cardinality k. For all $\alpha \geq 0$, Algorithm 1 executed for αk steps yields a set A with $f(A) \geq \beta(\alpha) f(OPT)$.*

Proof. Let A_i be the set at iteration i of the greedy algorithm. Then by monotonicity we have, $f(A_i \cup OPT) \geq f(OPT)$. Further, by submodularity,

$$\begin{aligned} \textstyle\sum_{e \in OPT \setminus A_i} f(e|A_i) &\geq f(OPT|A_i) \\ &= f(A_i \cup OPT) - f(A_i) \\ &\geq f(OPT) - f(A_i) \end{aligned}$$

Hence, there exists an element e^* in $OPT \setminus A_i$ such that, $f(e^*|A_i) \geq \frac{f(OPT) - f(A_i)}{k}$. This gives us the recurring inequality,

$$f(A_{i+1}) \geq f(A_i + e^*) \geq f(A_i) + \frac{f(OPT) - f(A_i)}{k} \quad \forall i \in \{0, \ldots, \alpha k - 1\}.$$

Therefore, $f(OPT) - f(A_{i+1}) \leq (1 - \frac{1}{k})(f(OPT) - f(A_i))$. After αk iterations,

$$\begin{aligned} f(OPT) - f(A_{\alpha k}) &\leq \left(1 - \frac{1}{k}\right)^{\alpha k} f(OPT) \\ \implies \beta(\alpha) f(OPT) &\leq f(A_{\alpha k}) \end{aligned}$$

□

Notice that in each step, Algorithm 1 makes $O(n)$ queries to the oracle. Therefore, it has an overall query complexity of $O(nk)$, which is n^2 in the worst case. A remarkable improvement, that leads to a $(1-1/e)(1-\epsilon)$ approximation with runtime $O(\frac{n}{\epsilon}\log\frac{n}{\epsilon})$ was shown in [BV14]. Their *thresholding algorithm* is described below.

Algorithm 2 Thresholding Algorithm

Input: k, f, ϵ and set $A = \emptyset$
Find $w^* = \max_{x \in N} f(x)$
$w = w^*$
while $w \geq \frac{\epsilon}{n} w^*$ **do**
    for all $x \in N \backslash A$ **do**
        if $|A| == k$ **then** output A and terminate
        else if $f(x|A) \geq w$ **then** $A = A + x$
    end for
    $w = w(1-\epsilon)$
    end while

Similar to the greedy algorithm above, the thresholding algorithm is also $\beta(\alpha)$ approximate after αk elements are chosen. Unless mentioned otherwise, the thresholding algorithm can be used as a subroutine in the algorithms discussed in Chapters 3 and 4, in place of the greedy algorithm. This improves the runtime of our algorithms, at the cost of a multiplicative factor of $(1-\epsilon)$ in the guarantee.

2.2 Continuous Greedy Framework

Recall that Algorithm 1 is $\frac{1}{2}$ approximate for the more general problem $\max_{S \in \mathcal{I}} f(S)$, where $\mathcal{I}$ is the collection of independent sets of a matroid. The continuous greedy framework evolved as a means of improving this bound to $(1-1/e)$ for matroid constraints [CCPV11, Von08]. It has found numerous other applications since then, such as in multi-objective maximization of submodular functions [CVZ10], showing

hardness results for various settings [Von13, DV12] and also for maximizing non-monotone submodular functions [FNS11], to name a few. The overall idea is similar to the relax-and-round framework employed for finding approximations to hard optimization problems. However, unlike the usual application of relax-and-round, in this instance it is not obvious what problem one should relax to, and moreover the right relaxation is non-trivial to solve. To describe the framework we will need some more notation first.

We use $\mathbf{x}_S$ to denote the support vector of a set S (1 along dimension i if $i \in S$ and 0 otherwise). We also use the short hand $|\mathbf{x}|$ to denote the ℓ_1 norm of a vector $\mathbf{x}$. Let $P(\mathcal{M})$ denote the matroid polytope [3] corresponding to the matroid $\mathcal{M}$ with collection of independent sets $\mathcal{I}$ and rank[4] d. Given $f : 2^N \to \mathbb{R}$, its *multilinear extension* over $\mathbf{x} = \{x_1, \ldots, x_n\} \in [0,1]^n$ is defined as,

$$F(\mathbf{x}) = \sum_{S \subseteq N} f(S) \prod_{i \in S} x_i \prod_{j \notin S} (1 - x_j).$$

The function can also be interpreted as the expectation of function value over sets obtained by including element $i \in N$ independently with probability x_i, for every i.

2.2.1 The Continuous Greedy Algorithm

F acts as a natural replacement for the original function f in the *continuous greedy algorithm* [CCPV11]. The algorithm always moves in a feasible direction that best increases the value of function F, similar to the greedy algorithm. So starting at the origin with $\mathbf{x}_0 = 0$, the algorithm traces a trajectory $\{\mathbf{x}_t\}_{t=0}^{T}$ in the matroid polytope $P(\mathcal{M})$ such that,

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{1}{T} \operatorname*{argmax}_{\mathbf{y} \in P(\mathcal{M})} \mathbf{y}.\nabla F(\mathbf{x}_t).$$

Where $T = O(d^2)$ is a suitably chosen discretization. Note that while evaluating the exact value of F and its gradient is naturally hard in general, for the purpose of using

[3] This is the convex hull of the maximally independent sets of a matroid (also called the bases of a matroid).

[4] The size of a maximial independent set.

this function in optimization algorithms, approximations obtained using a sampling based oracle suffice [BV14, CVZ10, CCPV11]. When the algorithm terminates, we have a vector $\mathbf{x}_T$ in the matroid polytope such that $F(\mathbf{x}_T) \geq (1 - 1/e)OPT$, where $OPT = \max_{S \in \mathcal{I}} f(S)$. By using *pipage rounding* [AS04] the point $\mathbf{x}_T$ can be rounded to a set $X \in \mathcal{I}$ such that, $f(X) \geq F(\mathbf{x}_T)$. A key property that follows from monotonicity and submodularity, and enables the above result is that at each step, there exists some vector $\mathbf{y} \in P(\mathcal{M})$ such that $\mathbf{y}.\nabla F(\mathbf{x}_t) + F(\mathbf{x}_t) \geq OPT$.

Next, we briefly discuss how the framework enables simultaneous maximization of multiple multilinear extensions of monotone submodular functions.

2.2.2 Maximizing Multiple Functions

Consider multilinear extensions F_i for monotone submodular functions f_i, with $i \in \{1, \ldots, m\}$. Suppose there exists a set S such that $f_i(S) \geq OPT$ for every i. Then given arbitrary point $\mathbf{x} \in P(\mathcal{M})$, there also exists a vector $\mathbf{y} \in P(\mathcal{M})$ such that, $\mathbf{y}.\nabla F_i(\mathbf{x}) + F_i(\mathbf{x}) \geq OPT$ for every i. One can find such a vector by solving the LP,

$$LP(\mathbf{x}) := \{\max 0 \mid \mathbf{y}.\nabla F_i(\mathbf{x}) + F_i(\mathbf{x}) \geq OPT \quad \forall i, \quad \mathbf{y} \in P(\mathcal{M})\}$$

If no such vector exists, we get a certificate of infeasibility from the dual of the LP. Using this property, we have an algorithm for simultaneously maximizing multiple multilinear extensions of monotone submodular functions. As before, starting at the origin with $\mathbf{x}_0 = 0$, the algorithm traces a trajectory $\{\mathbf{x}_t\}_{t=0}^{T}$ in the matroid polytope $P(\mathcal{M})$ such that, $\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{1}{T}\mathbf{y}$, where $\mathbf{y}$ is now a solution to $LP(\mathbf{x}_t)$. The final point $\mathbf{x}_T$ is such that $F_i(\mathbf{x}_T) \geq (1 - 1/e)OPT$ for every i.

We formally summarize this the lemma below and refer the interested reader to [CVZ10] for further details (which will not be necessary for subsequent discussion).

Lemma 2. *([CVZ10] Lemma 7.3) Given matroid $\mathcal{M}$, submodular functions f_i and values V_i, the continuous greedy algorithm finds a point $\mathbf{x} \in P(\mathcal{M})$ such that $F_i(\mathbf{x}) \geq (1 - 1/e)V_i, \forall i$ or outputs a certificate of infeasibility.*

Note that rounding is not straightforward anymore. Using pipage rounding one find a set X such that $f_i(X) \geq F_i(\mathbf{x}_T)$ for a specific i, but not for every i. This is where *swap* rounding comes into the picture [CVZ10]. This technique is discussed further in Chapter 3.

2.2.3 Runtime

A straightforward implementation of the continuous greedy algorithm for a single objective can take up to $\Theta(n^8)$ steps [CCPV11]. This was reduced to $O(nk^4/\epsilon^3)$ in [FW14] and subsequently to $\tilde{O}(n^2)$ in [BV14]. However, the last two improvements do not obviously translate when solving for multiple functions. Further, unlike the case of a single submodular function, where the LP at each step is an instance of the max weight independent set problem over matroids (which lends itself to a fast greedy solution), with multiple functions, one needs to solve the LP with a matroid separation oracle, exacerbating the runtime problem.

Chapter 3

Multi-objective Maximization of Monotone Submodular Functions

In this chapter we study the following problem,

$$MO_1 : \max_{A \subseteq N, |A| \leq k} \min_{i \in \{1,2,\ldots,m\}} f_i(A).$$

Each function f_i is monotone submodular. We refer the reader to Chapter 1, Section 1.1.1 for a discussion on applications. Our focus will be on finding algorithms for the problem when the number of objectives m is super-constant but much smaller than the cardinality parameter k i.e., $m = o(k)$. We start by discussing related work.

3.1 Related Work

[KMGG08] amalgamated various applications and formally introduced MO_1. They refer to is as the Robust Submodular Observation Selection (RSOS) problem and show that when $m = \Omega(k)$ the problem is inapproximable (no non-trivial approximation possible) unless P=NP. Consequently, they proceeded to give a bi-criterion approximation algorithm which achieves the optimal answer by violating the cardinality constraint.

On the other hand, [CVZ10] showed a randomized $(1 - 1/e) - \epsilon$ approximation

for constant m in the more general case of matroid constraint, as an application of a new technique for rounding over a matroid polytope, called *swap rounding*. The runtime scales as $O(n^{m/\epsilon^3} + n^8)$ [1]. Note, [CVZ10] consider a different but equivalent formulation of the problem that stems from the influential paper on multi-objective optimization [PY00]. The alternative formulation, which we review in Section 3.3, is the reason we call this a multi-objective maximization problem (same as [CVZ10]). To the best of our knowledge, when $m = o(k)$ no constant factor approximation algorithms or inapproximability results were known prior to this work.

3.2 Our Contributions

Our focus here is on the regime $m = o(k)$, where we show three algorithmic results.

1. Asymptotically optimal approximation algorithm: We give a $(1 - 1/e - \epsilon)(1 - \frac{m}{k\epsilon^3})$ approximation, which for $m = o\big(\frac{k}{\log^3 k}\big)$ and $\epsilon = \min\{\frac{1}{8 \ln m}, \sqrt[4]{\frac{m}{k}}\}$ tends to $1 - 1/e$ as $k \to \infty$. The algorithm is randomized and outputs such an approximation w.h.p. Observe that this implies a steep transition around m, due to the inapproximability result (to within any non-trivial factor) for $m = \Omega(k)$.

We obtain this via extending the matroid based algorithm of [CVZ10], which relies on the *continuous greedy* approach, resulting in a runtime of $O(n^8)$. Note that there is no ϵ dependence in the runtime, unlike the result from [CVZ10]. The key idea behind the result is quite simple, and relies on exploiting the fact that we are dealing with a cardinality constraint, far more structured than matroids.

2. Fast and practical approximation algorithm: In practice, n can range from tens of thousands to millions [OUS+08, LKG+07], which makes the above runtime intractable. To this end, we develop a fast $O(\frac{n}{\delta^3} \log m \log \frac{n}{\delta})$ time $(1 - 1/e)^2(1 - m/k\epsilon^3) - \epsilon - \delta$ approximation. Under the same asymptotic conditions as above, the guarantee simplifies to $(1 - 1/e)^2 - \delta$. We achieve this via the Multiplicative-Weight-Updates (MWU) framework, which replaces the bottleneck continuous greedy process. This is what costs us the additional factor of $(1 - 1/e)$ in the guarantee but allows

[1] The n^8 term could potentially be improved to n^5 by leveraging subsequent work [BV14, FW14].

us to leverage the runtime improvements for SO achieved in [BV14, MBK$^+$15].

MWU has proven to be a vital tool in the past few decades [GK94, AK07, Bie06, Fle00, GK04, GK07, KY07, You95, You01, PST91, AHK12]. Linear functions and constraints have been the primary setting of interest in these works, but recent applications have shown its usefulness when considering non-linear and in particular submodular objectives [AG12, CJV15]. Unlike these recent applications, we instead apply the MWU framework in vein of the Plotkin-Shmoys-Tardos scheme for linear programming [PST91], essentially showing that the non-linearity only costs us a another factor of $(1-1/e)$ in the guarantee and yields a nearly linear time algorithm.

3. Finding a deterministic approximation for small m: While the above results are all randomized, we also show a simple greedy based deterministic $(1 - 1/e - \epsilon)$ approximation with runtime kn^{m/ϵ^4}. This follows by establishing an upper bound on the increase in optimal solution value as a function of cardinality k.

Outline: We start with definitions and preliminaries in Section 3.3, where we also review relevant parts of the algorithm in [CVZ10] that are essential for understanding the results here. In Section 3.4, we state and prove the main results. Since the guarantees we present are asymptotic and technically converge to the constant factors indicated as k becomes large, in Section 3.5 we test the performance empirically. A heuristic closely inspired by our MWU based algorithm achieves improved performance over previous heuristics, even for small k and large m. Finally, we conclude in Section 3.6 with some open problems.

3.3 Preliminaries

3.3.1 Definitions & review

Let us start by introducing the alternative formulation of the multi-objective maximization problem that we alluded to earlier. Introduced in [CVZ10], we call this formulation MO_2. Given a target value V_i (positive real) with every function f_i, the goal in MO_2 is to find a set S^* of size at most k, such that $f_i(S^*) \geq V_i$, $\forall i \in \{1, \ldots, m\}$

or certify that no S^* exists. More feasibly, one aims to efficiently find a set S of size k such that $f_i(S) \geq \alpha V_i$ for all i and some factor α, or certify that there is no set S^* of size k such that $f_i(S^*) \geq V_i$, $\forall i$. W.l.o.g., assume $V_i = 1$ for every i (since we can consider functions $f_i(.)/V_i$ instead). Therefore, MO_2 is equivalent to the decision version of MO_1: *Given $t > 0$, find a set S^* of size at most k such that $\min_i f_i(S^*) \geq t$, or give a certificate of infeasibility.*

In MO_2 we can always consider the modified submodular objectives $\min\{f_i(.), V_i\}$. So w.l.o.g., assume that $f_i(S) \leq V_i$ for every set S and every function f_i. To avoid any confusion we highlight this assumption below,

Assumptions. *W.l.o.g. given target values V_i, assume $f_i(S) \leq V_i$ for every set $S \subseteq N$, for every $i \in \{1, \ldots, m\}$.*

For both MO_1, MO_2 we use S_k to denote an optimal/feasible set (optimal for MO_1, and feasible for MO_2) to the problem and OPT_k to denote the optimal value for formulation MO_1. Given two vectors $\mathbf{x}, \mathbf{y} \in [0,1]^n$, let $\mathbf{x} \vee \mathbf{y}$ denote the component wise maximum. Then we define marginals for F as,

$$F(\mathbf{x}|\mathbf{y}) = F(\mathbf{x} \vee \mathbf{y}) - F(\mathbf{y}).$$

We now give an overview of the algorithm from [CVZ10], which is based on MO_2. To simplify the description we focus on cardinality constraint, even though it is designed more generally for matroid constraint. We refer to it as **CVZ**, and it has three stages. Recall, the algorithm runs in time $O(n^{m/\epsilon^3} + n^8)$.

Stage 1: This stage is necessary for technical reasons associated with the rounding procedure in Stage 3. Intuitively, this is a pre-processing stage with the purpose of picking a small initial set consisting of elements with 'large' marginal values, i.e. marginal value at least $\epsilon^3 V_i$ for some function f_i.

Given a set S of size k, index elements in $S = \{s_1, \ldots, s_k\}$ in the order in which the greedy algorithm would pick them. Now, there are at most $1/\epsilon^3$ elements such that $f_i(s_j|\{s_1, \ldots, s_{j-1}\}) \geq \epsilon^3 V_i$ for any fixed i, otherwise by monotonicity we have that $f_i(S) > V_i$ (violating our w.l.o.g. assumption that $f_i(S) \leq V_i \, \forall i$). In fact, due to

decreasing marginal values we have, $f_i(s_j|\{s_1,\ldots,s_{j-1}\}) < \epsilon^3 V_i$ for every $j > 1/\epsilon^3$.

Therefore, we focus on sets of size $\leq m/\epsilon^3$ (at most $1/\epsilon^3$ elements for each function) to find an initial set such that the remaining elements have marginal value $\leq \epsilon^3 V_i$ for f_i, for every i. In particular, one can try all possible initial sets of this size (i.e. run subsequent stages with different starting sets), leading to the n^{m/ϵ^3} term in the runtime. Stages 2 and 3 have runtime polynomial in m (in fact Stage 3 has runtime independent of m). Hence, Stage 1 is really the bottleneck. For the more general case of matroid constraint, it is not obvious at all if one can do better than brute force enumeration over all possible starting sets and still retain the approximation guarantee. However, we will show that for cardinality constraints one can easily avoid enumeration.

Stage 2: Given a starting set S from stage one, this stage works with the ground set $N - S$ and runs the continuous greedy algorithm. Suppose a feasible set S_k exists for the problem, then for the right starting set $S_1 \subset S_k$, this stage outputs a fractional point $\mathbf{x}(k_1) \in [0,1]^n$ with $|\mathbf{x}(k_1)| = k_1 = k - |S_1|$ such that $F_i(\mathbf{x}(k_1)|\mathbf{x}_{S_1}) \geq (1 - 1/e)(V_i - f_i(S_1))$ for every i. Where F_i is the multilinear extension of f_i and we used the notation for marginals $F_i(.|.)$, introduced in Chapter 2 Section 2.2. This stage is computationally expensive and takes time $O(n^8)$. For additional details, we refer the reader to Chapter 2 Section 2.2.2, and [CVZ10].

Stage 3: For the right starting set S_1 (if one exists), Stage 2 successfully outputs a point $\mathbf{x}(k_1)$. Stage 3 now follows a random process that converts $\mathbf{x}(k_1)$ into a set S_2 of size k_1 such that, $S_2 \in N - S_1$ and $f_i(S_1 \cup S_2) \geq (1 - 1/e)(1 - \epsilon)V_i, \forall i$ as long as $\epsilon < 1/8 \ln m$. The rounding procedure is called *swap rounding* and we include a specialized version of the formal lemma below.

Lemma 3. *([CVZ10] Theorem 1.4, Theorem 7.2) Given m monotone submodular functions $f_i(.)$ with the maximum value of singletons in $[0, \epsilon^3 V_i]$ for every i; a fractional point $\mathbf{x}$ with $|\mathbf{x}| \in \mathbb{Z}$ and $\epsilon < \frac{1}{8\gamma \ln m}$. Swap Rounding yields a set R with*

cardinality $|\mathbf{x}|$*, such that,*

$$\sum_i \Pr[f_i(R) < (1-\epsilon)F_i(\mathbf{x})] < me^{-1/8\epsilon} < 1/m^{\gamma-1}.$$

Remark: For any $\gamma > 1$, the above can be converted to a result w.h.p. by standard repetition. Also this is a simplified version of the matroid based result in [CVZ10].

The condition in Lemma 3 that requires that the value of every singleton is at most $\epsilon^3 V_i$ is the reason behind Stage 1 of the algorithm. For the right starting set S_1, all elements which could possibly violate the condition in Lemma 3 have been included in S_1 and are thus absent from the support of $\mathbf{x}(k_1)$.

3.3.2 Some Simple Heuristics

Before we present the main results, let us take a step back and examine some variants of the standard greedy algorithm. To design a greedy heuristic for multiple functions, what should the objective for greedy selection be?

One possibility is to split the selection of k elements into m equal parts. In part i, pick k/m elements greedily w.r.t. function f_i. It is not difficult to see that this is a (tight) $\beta(k/m)$ approximation. Second, recall that the convex combination of monotone submodular functions is also monotone and submodular. Therefore, one could run the greedy algorithm on a fixed convex combination of the m functions. It can be shown this does not lead to an approximation better than $1/\Theta(m)$. This is indeed the idea behind the bi-criterion approximation in [KMGG08]. Third, one could select elements greedily w.r.t. the objective function $h(.) = \min_i f_i(.)$. A naïve implementation of this algorithm can have arbitrarily bad performance even for $m = 2$. We show later in Section 3.4.3, that if one greedily (w.r.t. $h(.)$) picks sets of size k' instead of singletons at each step, for large enough k' one can get arbitrarily close to $(1 - 1/e)$.

3.4 Main Results

3.4.1 Asymptotic $(1-1/e)$ Approximation for $m = o\big(\frac{k}{\log^3 k}\big)$

We replace the enumeration in Stage 1 with a single starting set, obtained by scanning once over the ground set. The main idea is simply that for the cardinality constraint case, any starting set that fulfills the Stage 3 requirement of small marginals will be acceptable (not true for general matroids).

New Stage 1: Start with $S_1 = \emptyset$ and pass over all elements once in arbitrary order. *For each element e, add it to S_1 if for some i, $f_i(e|S_1) \geq \epsilon^3 V_i$.* Note that we add at most m/ϵ^3 elements (at most $1/\epsilon^3$ for each function). When the subroutine terminates, for every remaining element $e \in N \backslash S_1$, $f_i(e|S_1) < \epsilon^3 V_i, \forall i$ (as required by Lemma 3). Let $k_1 = k - |S_1|$ and note $k_1 \geq k - m/\epsilon^3$.

Stage 2 remains the same as CVZ and outputs a fractional point $\mathbf{x}(k_1)$ with $|\mathbf{x}(k_1)| = k_1$. While enumeration over all starting sets allowed us to find a starting set such that $F_i(\mathbf{x}(k_1)|\mathbf{x}_{S_1}) \geq (1-1/e)(V_i - f_i(S_1))$ for every i; with the new Stage 1 we will need to further exploit properties of the multilinear extension to show a similar lower bound on the marginal value of $\mathbf{x}(k_1)$.

Corollary 4. *Given a point $\mathbf{x} \in [0,1]^n$ with $|\mathbf{x}| = k$ and a multilinear extension F of a monotone submodular function, for every $k_1 \leq k$,*

$$F\Big(\frac{k_1}{k}\mathbf{x}\Big) \geq \frac{k_1}{k} F(\mathbf{x}).$$

Proof. Note that the statement is true for concave F. The proof now follows directly from the concavity of multilinear extensions in positive directions (Section 2.1 of [CCPV11]). □

Lemma 5. *$F_i(\mathbf{x}(k_1)|\mathbf{x}_{S_1}) \geq \beta(1)\frac{k_1}{k}(V_i - f_i(S_1))$ for every i.*

Proof. Recall that S_k denotes a feasible solution with cardinality k, and let $\mathbf{x}_{S_k}$ denote its characteristic vector. Clearly, $|\mathbf{x}_{S_k \backslash S_1}| \leq k$ and $F_i(\mathbf{x}_{S_k \backslash S_1}|\mathbf{x}_{S_1}) = f_i(S_k|S_1) \geq (V_i - f_i(S_1))$ for very i. And now from Corollary 4, we have that there exists a point

$\mathbf{x}'$ with $|\mathbf{x}'| = k_1$ such that $F_i(\mathbf{x}'|\mathbf{x}_{S_1}) \geq \frac{k_1}{k} F_i(\mathbf{x}_{S_k \setminus S_1}|\mathbf{x}_{S_1})$ for every i. Finally, using Lemma 2 we have $F_i(\mathbf{x}(k_1)|\mathbf{x}_{S_1}) \geq \beta(1) F_i(\mathbf{x}'|S_1)$, which gives the desired bound. □

Stage 3 rounds $\mathbf{x}(k_1)$ to S_2 of size k_1, and final output is $S_1 \cup S_2$. The following theorem now completes the analysis.

Theorem 6. *For $\epsilon = \min\{\frac{1}{8 \ln m}, \sqrt[4]{\frac{m}{k}}\}$ we have, $f_i(S_1 \cup S_2) \geq (1-\epsilon)(1-1/e)(1 - m/k\epsilon^3)V_i \, \forall i$ with constant probability. For $m = o\big(k/\log^3 k\big)$ the factor is asymptotically $(1-1/e)$.*

Proof. From Lemma 5 and applying Lemma 3 we have, $f_i(S_2|S_1) \geq (1-\epsilon)(1-1/e)(1-m/k\epsilon^3)(V_i - f_i(S_1)), \forall i$. Therefore, $f_i(S_1 \cup S_2) \geq (1-\epsilon)(1-1/e)(1-m/k\epsilon^3)V_i, \forall i$. Note that the runtime is now independent of ϵ. Stage 1 has runtime $O(n)$ and Stages 2 and 3 have runtime polynomial in n, m. To refine the guarantee, we choose $\epsilon = \min\{\frac{1}{8 \ln m}, \sqrt[4]{\frac{m}{k}}\}$, where the $\frac{1}{8 \ln m}$ is due to Lemma 3 and the $\sqrt[4]{\frac{m}{k}}$ term is to balance ϵ and $m/k\epsilon^3$. The resulting guarantee becomes $(1-1/e)(1-h(k))$, where the function $h(k) \to 0$ as $k \to \infty$, so long as $m = o\big(\frac{k}{\log^3 k}\big)$. □

3.4.2 Fast, Asymptotic $(1-1/e)^2 - \delta$ Approximation for $m = o\big(\frac{k}{\log^3 k}\big)$

While the previous algorithm achieves the best possible asymptotic guarantee, it is infeasible to use in practice. The main underlying issue was our usage of the continuous greedy algorithm in Stage 2 which has runtime $O(n^8)$, but the flexibility offered by continuous greedy was key to maximizing the multilinear extensions of all functions at once. To improve the runtime we avoid continuous greedy and find an alternative in Multiplicative-Weight-Updates (MWU) instead. MWU allows us to combine multiple submodular objectives together into a single submodular objective and utilize fast algorithms for SO at every step.

The algorithm consists of 3 stages as before. Stage 1 remains the same as the New Stage 1 introduced in the previous section. Let S_1 be the output of this stage as before. Stage 2 is replaced with a fast MWU based subroutine that runs for

$T = O(\frac{\ln m}{\delta^2})$ rounds and solves an instance of SO during each round. Here δ is an artifact of MWU and manifests as a subtractive term in the approximation guarantee. The currently fastest algorithm for SO, in [MBK+15], has runtime $O(n \log \frac{1}{\delta'})$ and an *expected* guarantee of $(1 - 1/e) - \delta'$. However, the slightly slower, but still nearly linear time $O(\frac{n}{\delta'} \log \frac{n}{\delta'})$ *thresholding* algorithm in [BV14], has (the usual) deterministic guarantee of $(1 - 1/e) - \delta'$. Both of these are known to perform well in practice and using either would lead to a runtime of $T \times \tilde{O}(n/\delta) = \tilde{O}(\frac{n}{\delta^3})$, which is a vast improvement over the previous algorithm.

Now, fix some algorithm $\mathcal{A}$ for SO with guarantee α, and let $\mathcal{A}(f, k)$ denote the set it outputs given monotone submodular function f and cardinality constraint k as input. Note that α can be as large as $1 - 1/e$, and we have $k_1 = k - |S_1|$ as before. Then the new Stage 2 is,

Algorithm 2 Stage 2: MWU

1: **Input:** $\delta, T = \frac{2 \ln m}{\delta^2}, \lambda_i^1 = 1/m, \tilde{f}_i(.) = \frac{f_i(.|S_1)}{V_i - f_i(S_1)}$
2: **while** $1 \leq t \leq T$ **do** $g^t(.) = \sum_{i=1}^m \lambda_i^t \tilde{f}_i(.)$
3: $X^t = \mathcal{A}(g^t, k_1)$
4: $m_i^t = \tilde{f}_i(X^t) - \alpha$
5: $\lambda_i^{t+1} = \lambda_i^t(1 - \delta m_i^t)$
6: $t = t + 1$
7: **Output:** $\mathbf{x}_2 = \frac{1}{T} \sum_{t=1}^T X^t$

The point $\mathbf{x}_2$ *obtained above is rounded to a set* S_2 *in Stage 3 (which remains unchanged). The final output is* $S_1 \cup S_2$. Note that we abuse notation and used X^t to denote sets as well as the respective support vectors. We continue to use X^t and $\mathbf{x}_{X^t}$ interchangeably in the below.

This application of MWU is unlike [AG12, CJV15], where broadly speaking the MWU framework is applied in a novel way to determine how an individual element is picked (or how a direction for movement is chosen in case of continuous greedy). In contrast, we use standard algorithms for SO and pick an entire set before changing

weights. Also, [CJV15] uses MWU along with the continuous greedy framework to tackle harder settings, but for our setting using the continuous greedy framework eliminates the need for MWU altogether and in fact, we use MWU as a replacement for continuous greedy.

Now, consider the following intuitive schema. We would like to find a set X of size k such that $f_i(X) \geq \alpha V_i$ for every i. While this seems hard, consider the combination $\sum_i \lambda_i f_i(.)$, which is also monotone submodular for non-negative λ_i. We can easily find a set X_λ such that $\sum_i \lambda_i f_i(X_\lambda) \geq \sum_i \lambda_i V_i$, since this is a single objective problem and we have fast approximations for SO. However, for a fixed set of scalar weights λ_i, solving the SO problem instance need not give a set that has sufficient value for every individual function $f_i(.)$. This is where MWU comes into the picture. We start with uniform weights for functions, solve an instance of SO to get a set X^1. Then we change weights to undermine the functions for which $f_i(X^1)$ was closer to the target value and stress more on functions for which $f_i(X^1)$ was small, and repeat now with new weights. After running many rounds of this, we have a collection of sets X^t for $t \in \{1, \ldots, T\}$. Using tricks from standard MWU analysis ([AHK12]) along with submodularity and monotonicity, we show that $\sum_t \frac{f_i(X^t|S_1)}{T} \gtrapprox (1 - 1/e)(V_i - f_i(S_1))$. Thus far, this resembles how MWU has been used in the literature for linear objectives, for instance the Plotkin-Shmoys-Tardos framework for solving LPs. However, a new issue now arises due to the non-linearity of functions f_i. As an example, suppose that by some coincidence $\mathbf{x}_2 = \frac{1}{T}\sum_{t=1}^{T} X^t$ turns out to be a binary vector, so we easily obtain the set S_2 from $\mathbf{x}_2$. We want to lower bound $f_i(S_2|S_1)$, and while we have a good lower bound on $\sum_t \frac{f_i(X^t|S_1)}{T}$, it is unclear how the two quantities are related. More generally, we would like to show that $F_i(\mathbf{x}_2|\mathbf{x}_{S_1}) \geq \beta \sum_t \frac{f_i(X^t|S_1)}{T}$ and this would then give us a $\beta\alpha = \beta(1 - 1/e)$ approximation using Lemma 3. Indeed, we show that $\beta \geq (1 - 1/e)$, resulting in a $(1 - 1/e)^2$ approximation. Now, we state and prove lemmas that formalize the above intuition.

Lemma 7. $g^t(X^t) \geq \frac{k_1}{k}\alpha \sum_i \lambda_i^t, \forall t.$

Proof. Consider the optimal set S_k and note that $\sum_i \lambda_i^t \tilde{f}_i(S_k) \geq \sum_i \lambda_i^t, \forall t$. Now

the function $g^t(.) = \sum_i \lambda_i^t \tilde{f}_i(.)$, being a convex combination of monotone submodular functions, is also monotone submodular. We would like to show that there exists a set S' of size k_1 such that $g^t(S') \geq \frac{k_1}{k} \sum_i \lambda_i^t$. Then the claim follows from the fact that $\mathcal{A}$ is an α approximation for monotone submodular maximization with cardinality constraint.

To see the existence of such a set S', greedily index the elements of S_k using $g^t(.)$. Suppose that the resulting order is $\{s_1, \ldots, s_k\}$, where s_i is such that $g^t(s_i|\{s_1, \ldots, s_{i-1}\}) \geq g^t(s_j|\{s_1, \ldots, s_{i-1}\})$ for every $j > i$. Then the truncated set $\{s_1, \ldots, s_{k-|S_1|}\}$ has the desired property, and we are done. □

Lemma 8.

$$\frac{\sum_t \tilde{f}_i(X^t)}{T} \geq \frac{k_1}{k}(1 - 1/e) - \delta \,, \forall i.$$

Proof. Suppose we have,

$$\frac{\sum_t \tilde{f}_i(X^t) - \alpha}{T} + \delta \geq \frac{1}{T} \sum_t \sum_i \frac{\lambda_i^t}{\sum_i \lambda_i^t} (\tilde{f}_i(X^t) - \alpha), \forall i. \tag{3.1}$$

Then assuming $\alpha = (1 - 1/e)$, the RHS above simplifies to,

$$\frac{1}{T} \sum_t \frac{g(X^t)}{\sum_i \lambda_i^t} - (1 - 1/e) \quad \geq (1 - 1/e)(\frac{k_1}{k} - 1) \quad \text{(using Lemma 7)}$$

And we have for every i,

$$\begin{aligned} \frac{\sum_t \tilde{f}_i(X^t) - (1 - 1/e)}{T} + \delta \quad &\geq (1 - 1/e)(\frac{k_1}{k} - 1) \\ \frac{\sum_t \tilde{f}_i(X^t)}{T} \quad &\geq \frac{k_1}{k}(1 - 1/e) - \delta. \end{aligned}$$

Now, the proof for (3.1) closely resembles the analysis of Theorem 3.3 and 2.1 in [AHK12]. We will use the potential function $\Phi^t = \sum_i \lambda_i^t$. Let $p_i^t = \lambda_i^t / \Phi^t$ and

$M^t = \sum_i p_i^t m_i^t$. Then we have,

$$\begin{aligned}\Phi^{t+1} &= \sum_i \lambda_i^t (1 - \delta m_i^t) \\ &= \Phi^t - \delta \Phi^t \sum_i p_i^t m_i^t \\ &= \Phi^t (1 - \delta M^t) \leq \Phi^t e^{-\delta M^t}\end{aligned}$$

After T rounds, $\Phi^T \leq \Phi^1 e^{-\delta \sum_t M^t}$. Further, for every i,

$$\begin{aligned}\Phi^T \geq w_i^T = \tfrac{1}{m} \textstyle\prod_t (1 - \delta m_i^t) \\ \ln(\Phi^1 e^{-\delta \sum_t M^t}) \geq \textstyle\sum_t \ln(1 - \delta m_i^t) - \ln m \\ \delta \textstyle\sum_t M^t \leq \ln m + \sum_t \ln(1 - \delta m_i^t)\end{aligned}$$

Using $\ln(\frac{1}{1-\epsilon}) \leq \epsilon + \epsilon^2$ and $\ln(1+\epsilon) \geq \epsilon - \epsilon^2$ for $\epsilon \leq 0.5$, and with $T = \frac{2 \ln m}{\delta^2}$ and $\delta < (1 - 1/e)$ (for a positive approximation guarantee), we have,

$$\frac{\sum_t M^t}{T} \leq \delta + \frac{\sum_t m_i^t}{T}, \forall i.$$

□

Lemma 9. *Given monotone submodular function f, its multilinear extension F, sets X^t for $t \in \{1, \ldots, T\}$, and a point $\mathbf{x} = \sum_t X^t/T$, we have,*

$$F(\mathbf{x}) \geq (1 - 1/e) \frac{1}{T} \sum_{t=1}^{T} f(X^t).$$

Proof. Consider the concave closure of a submodular function f,

$$f^+(\mathbf{x}) = \max_{\alpha} \{ \sum_X \alpha_X f(X) | \sum_X \alpha_X X = \mathbf{x}, \sum_X \alpha_X \leq 1, \alpha_X \geq 0 \, \forall X \subseteq N \}.$$

Clearly, $f_i^+(\mathbf{x}) \geq \frac{\sum_t f_i(X^t)}{T}$. So it suffices to show $F_i(\mathbf{x}) \geq (1 - 1/e) f_i^+(\mathbf{x})$, which in fact, follows from Lemmas 4 and 5 in [CCPV07].

Alternatively, we now give a novel and direct proof for the statement. We abuse

notation and use $\mathbf{x}_{X^t}$ and X^t interchangeably. Let $\mathbf{x} = \sum_{t=1}^{T} X^t/T$ and w.l.o.g., assume that sets X^t are indexed such that $f(X^j) \geq f(X^{j+1})$ for every $j \geq 1$. Further, let $f(X^t)/T = a^t$ and $\sum_t a^t = A$.

Recall that $F(\mathbf{x})$ can be viewed as the expected function value of the set obtained by independently sampling element j with probability x_j. Instead, consider the alternative random process where starting with $t = 1$, one samples each element in set X^t independently with probability $1/T$. The random process runs in T steps and the probability of an element j being chosen at the end of the process is exactly $p_j = 1 - (1 - 1/T)^{Tx_j}$, independent of all other elements. Let $\mathbf{p} = (p_1, \ldots, p_n)$, it follows that the expected value of the set sampled using this process is given by $F(\mathbf{p})$. Observe that for every j, $p_j \leq x_j$ and therefore, $F(\mathbf{p}) \leq F(\mathbf{x})$. Now in step t, suppose the newly sampled subset of X^t adds marginal value Δ^t. From submodularity we have, $\mathbb{E}[\Delta^1] \geq \frac{f(X^1)}{T} = a^1$ and in general, $\mathbb{E}[\Delta^t] \geq \frac{f(X^t) - \mathbb{E}[\sum_{j=1}^{t-1} \Delta_j]}{T} \geq a^t - \frac{1}{T}\sum_{j=1}^{t-1} \mathbb{E}[\Delta^j]$.

To see that $\sum_t \mathbb{E}[\Delta^t] \geq (1 - 1/e)A$, consider a LP where the objective is to minimize $\sum_t \gamma^t$ subject to $b^1 \geq b^2 \cdots \geq b^T \geq 0$; $\sum b^t = A$ and $\gamma^t \geq b^t - \frac{1}{T}\sum_{j=1}^{t-1} \gamma^j$ with $\gamma^0 = 0$. Here A is a parameter and everything else is a variable. Observe that the extreme points are characterized by j such that, $\sum b^t = A$ and $b^t = b^1$ for all $t \leq j$ and $b^{j+1} = 0$. For all such points, it is not difficult to see that the objective is at least $(1 - 1/e)A$. Therefore, we have $F(\mathbf{p}) \geq (1 - 1/e)A = (1 - 1/e)\sum_t f(X^t)/T$, as desired.

□

Theorem 10. *For $\epsilon = \min\{\frac{1}{8 \ln m}, \sqrt[4]{\frac{m}{k}}\}$, the algorithm makes $O(\frac{n}{\delta^3} \log m \log \frac{n}{\delta})$ queries, and with constant probability outputs a feasible $(1-\epsilon)(1-1/e)^2(1-\frac{m}{k\epsilon^3})-\delta$ approximate set. Asymptotically, $(1 - 1/e)^2 - \delta$ approximate for $m = o\big(k/\log^3 k\big)$.*

Proof. Combining Lemmas 8 & 16 we have, $\tilde{F}_i(\mathbf{x}_2) \geq (1 - 1/e)\frac{\sum_t \tilde{f}_i(X^t)}{T} \geq \frac{k_1}{k}(1 - 1/e)^2 - \delta\,, \forall i$. The asymptotic result follows just as in Theorem 6. For runtime, note that Stage 1 takes time $O(n)$. Stage 2 runs an instance of $\mathcal{A}(.)$, T times, leading to an upper bound of $O((\frac{n}{\delta}\log\frac{n}{\delta}) \times \frac{\log m}{\delta^2}) = O(\frac{n}{\delta^3}\log m \log\frac{n}{\delta})$, if we use the thresholding algorithm in [BV14] (at the cost of a multiplicative factor of $(1-\delta)$ in the

approximation guarantee). Finally, since we have a decomposition of $\mathbf{x}_2$ into T sets of size k_1 each from Stage 2, swap rounding takes time Tk_1, leading to total runtime $O(\frac{k}{\delta^2}\log m)$ for Stage 3. Combining all three we get a runtime of $O(\frac{n}{\delta^3}\log m \log \frac{n}{\delta})$. $\square$

3.4.3 Variation in Optimal Solution Value and Derandomization

Consider the problem SO with cardinality constraint k. Given an optimal solution S_k with value OPT_k for the problem, it is not difficult to see that for arbitrary $k' \leq k$, there is a subset $S_{k'} \subseteq S_k$ of size k', such that $f(S_{k'}) \geq \frac{k'}{k}OPT_k$. For instance, indexing the elements in S_k using the greedy algorithm, and choosing the set given by the first k' elements gives such a set. This implies $OPT_{k'} \geq \frac{k'}{k}OPT_k$, and the bound is easily seen to be tight.

This raises a natural question: Can we generalize this bound on variation of optimal solution value with varying k, for multi-objective maximization? A priori, this isn't obvious even for modular functions. In particular, note that indexing elements in order they are picked by the greedy algorithm doesn't suffice since there are many functions and we need to balance values amongst all. We show below that one can indeed derive such a bound.

Lemma 11. *Given that there exists a set S_k such that $f_i(S_k) \geq V_i, \forall i$ and $\epsilon < \frac{1}{8\ln m}$. For every $k' \in [m/\epsilon^3, k]$, there exists $S_{k'} \subseteq S_k$ of size k', such that,*

$$f_i(S_{k'}) \geq (1-\epsilon)\Big(\frac{k'-m/\epsilon^3}{k-m/\epsilon^3}\Big)V_i, \forall i.$$

Proof. We restrict our ground set of elements to S_k and let S_1 be a subset of size at most m/ϵ^3 such that $f_i(e|S_1) < \epsilon^3 V_i, \forall e \in S_k \backslash S_1$ and $\forall i$ (recall, we discussed the existence of such a set in Section 2.1, Stage 1). The rest of the proof is similar to the proof of Lemma 5. Consider the point $\mathbf{x} = \frac{k'-|S_1|}{k-|S_1|}\mathbf{x}_{S_k \backslash S_1}$. Clearly, $|\mathbf{x}| = k' - |S_1|$, and from Corollary 4, we have $F_i(\mathbf{x}|\mathbf{x}_{S_1}) \geq \frac{k'-|S_1|}{k-|S_1|}F_i(\mathbf{x}_{S_k\backslash S_1}|\mathbf{x}_{S_1}) = \frac{k'-|S_1|}{k-|S_1|}f_i(S_k\backslash S_1|S_1) \geq \frac{k'-|S_1|}{k-|S_1|}(V_i - f_i(S_1)), \forall i$. Finally, using swap rounding Lemma 1, there exists a set S_2

of size $k' - |S_1|$, such that $f_i(S_1 \cup S_2) \geq (1-\epsilon)\frac{k'-|S_1|}{k-|S_1|}V_i, \forall i$.

□

For large k, $\frac{k'-m/\epsilon^3}{k-m/\epsilon^3}$ tends to $\frac{k'}{k}$. As we show next, this implies that for large enough k', we can choose sets of size k' (k'-tuples) at each step to get a deterministic (asymptotically) $(1-1/e)-\epsilon$ approximation with runtime $O(kn^{m/\epsilon^4})$ for the multi-objective maximization problem, when m is constant (all previously known approximation algorithms, as well as the ones presented earlier, are randomized).

Theorem 12. *For $k' = \frac{m}{\epsilon^4}$, choosing k'-tuples greedily w.r.t. $h(.) = \min_i f_i(.)$ yields approximation guarantee $(1-1/e)(1-2\epsilon)$ for $k \to \infty$, while making kn^{m/ϵ^4} queries.*

Proof. The analysis generalizes that of the standard greedy algorithm in [NW78, NWF78]. Let S_j denote the set at the end of iteration j. $S_0 = \emptyset$ and let the final set be $S_{\lfloor k/k' \rfloor}$. Then from Theorem 11, we have that at step $j+1$, there is some set $X \in S_k \backslash S_j$ of size k' such that

$$f_i(X|S_j) \geq (1-\epsilon)\frac{k'-m/\epsilon^3}{k-m/\epsilon^3}\big(V_i - f_i(S_j)\big), \forall i.$$

To simplify presentation let $\eta = (1-\epsilon)\frac{k'-m/\epsilon^3}{k-m/\epsilon^3}$ and note that $\eta \leq 1$. Further, $1/\eta \to \infty$ as $k \to \infty$ for fixed m and $k' = o(k)$. Now, we have for every i, $f_i(S_{j+1}) \geq f_i(S_j) + \eta(V_i - f_i(S_j))$. Therefore, $V_i - f_i(S_{j+1}) \leq (1-\eta)(V_i - f_i(S_j))$, which gives us $V_i - f_i(S_{\lfloor k/k' \rfloor}) \leq (1-\eta)^{\lfloor k/k' \rfloor}V_i$. Rearranging terms,

$$\begin{aligned} f_i(S_{\lfloor k/k' \rfloor}) \quad &\geq (1-(1-\eta)^{\lfloor k/k' \rfloor})V_i \\ &\geq \beta(\eta\lfloor k/k' \rfloor)V_i \quad \geq (1-1/e)(\eta\lfloor k/k' \rfloor)V_i. \end{aligned}$$

Where the last inequality follows from the fact that for $\alpha \leq 1$, $\beta(\alpha) = (1-e^{1-\alpha}/e) \geq (1-1/e)\alpha$. Let $\epsilon = \sqrt[4]{\frac{m}{k'}}$, then we have,

$$\eta\lfloor k/k' \rfloor \geq (1-\epsilon)\frac{1-m/k'\epsilon^3}{1-m/k\epsilon^3}\Big(1-\frac{k'}{k}\Big) \geq \frac{\Big(1-\sqrt[4]{\frac{m}{k'}}\Big)^2}{1-\frac{1}{k}\sqrt[4]{\frac{m}{(k')^3}}}\Big(1-\frac{k'}{k}\Big)$$

As $k \to \infty$ we get the asymptotic guarantee $(1-1/e)\left(1-\sqrt[4]{\frac{m}{k'}}\right)^2 = (1-1/e)(1-\epsilon)^2$. □

3.5 Experiments on Kronecker Graphs

We choose synthetic experiments where we can control the parameters to see how the algorithm performs in various scenarios, esp. since we would like to test how the MWU algorithm performs for small values of k and $m = \Omega(k)$. We work with formulation MO_1 of the problem and consider a multi-objective version of the *max-k-cover* problem on graphs. Random graphs for our experiments were generated using the Kronecker graph framework introduced in [LCK+10]. These graphs exhibit several natural properties and are considered a good approximation for real networks (esp. social networks [HK16]).

We compare three algorithms: (i) A baseline greedy heuristic, labeled GREEDY, which focuses on one objective at a time and successively picks k/m elements greedily w.r.t. each function (formally stated below). (ii) The bi-criterion approximation called SATURATE from [KMGG08], to the best of our knowledge this is considered state-of-the-art for the problem. (iii) We compare these algorithms to a heuristic inspired by our MWU algorithm. This heuristic differs from the algorithm discussed earlier in two ways. Firstly, we eliminate Stage 1 which was key for technical analysis but in practice makes the algorithm perform similar to GREEDY. Second, instead of simply using the the swap rounded set S_2, we output the best set out of $\{X^1, \ldots, X^T\}$ and S_2. Also, for both SATURATE and MWU we estimate target value t using binary search and consider capped functions $\min\{f_i(.), t\}$. Also, for the MWU stage, we use $\delta = 0.5$.

Algorithm 3 GREEDY

1: **Input:** $k, m, f_i(.)$ for $i \in [m]$

2: $S = \emptyset, i = 1$

3: **while** $|S| \leq k - 1$ **do**

4: $S = S + \arg\max_{x \in N-S} f_i(x|S)$

5: $i = i + 1 \mod m + 1$

6: **Output:** S

Algorithm 4 SATURATE

1: Input: $k, t, f_1, \ldots, f_m$ and set $A = \emptyset$

2: $g(.) = \sum_i \min\{f_i(.), t\}$

3: **while** $|A| < k$ **do** $A = A + \underset{x \in N-A}{\operatorname{argmax}}\ g(x|A)$

4: Output: A

We pick Kronecker graphs of sizes $n \in \{64, 512, 1024\}$ with random initiator matrix [2] and for each n, we test for $m \in \{10, 50, 100\}$. Note that each graph here represents an objective, so for a fixed n, we generate m Kronecker graphs to get m *max-cover* objectives. For each setting of n, m we evaluate the solution value for the heuristics as k increases and show the average performance over 30 trials for each setting. All experiments were performed using MATLAB.

[2] To generate a Kronecker graph one needs a small initiator matrix. Using [LCK+10] as a guideline we use random matrices of size 2×2, each entry chosen uniformly randomly (and independently) from $[0, 1]$. Matrices with sum of entries smaller than 1 are discarded to avoid highly disconnected graphs.

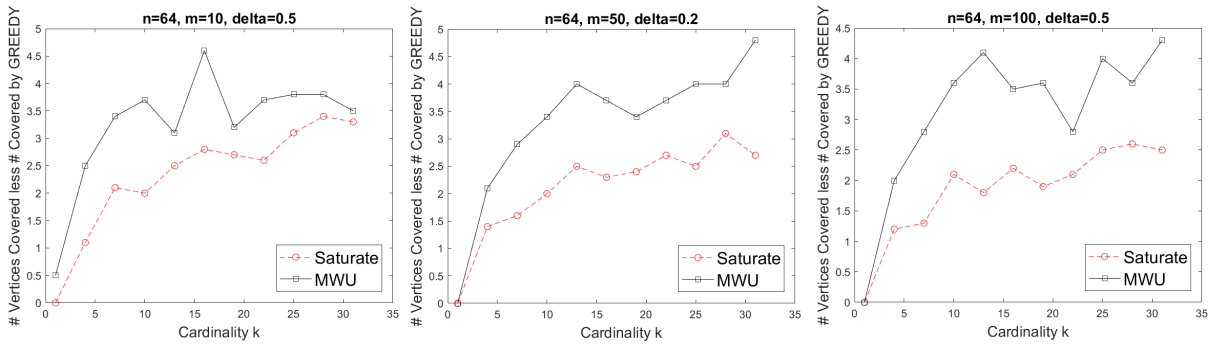

Figure 3-1: Plots for graphs of size 64. Number of objectives increases from left to right. The X axis is the cardinality parameter k and Y axis is difference between # vertices covered by MWU and SATURATE minus the # vertices covered by GREEDY for the same k. MWU outperforms the other algorithms in all cases, with a max. gain (on SATURATE) of 9.80% for $m = 10$, 12.14% for $m = 50$ and 16.12% for $m = 100$.

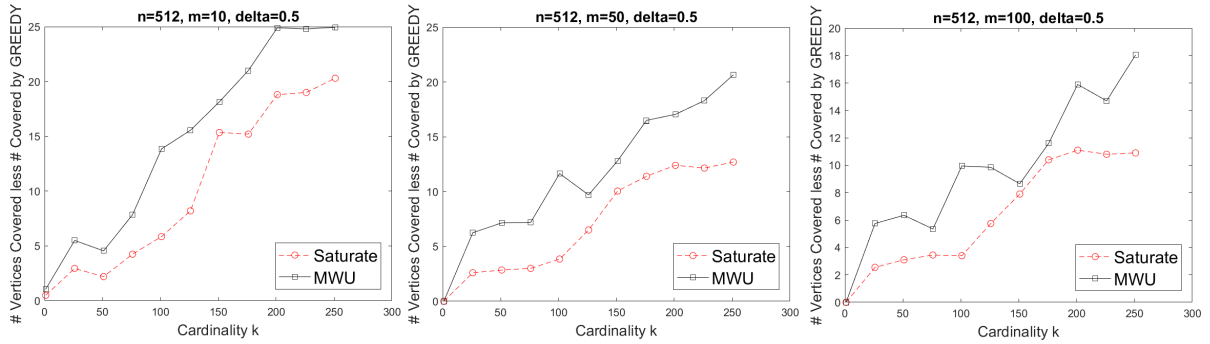

Figure 3-2: Plots for graphs of size 512. MWU outperforms SATURATE in all cases with a max. gain (on SATURATE) of 7.95% for $m = 10$, 10.08% for $m = 50$ and 10.01% for $m = 100$.

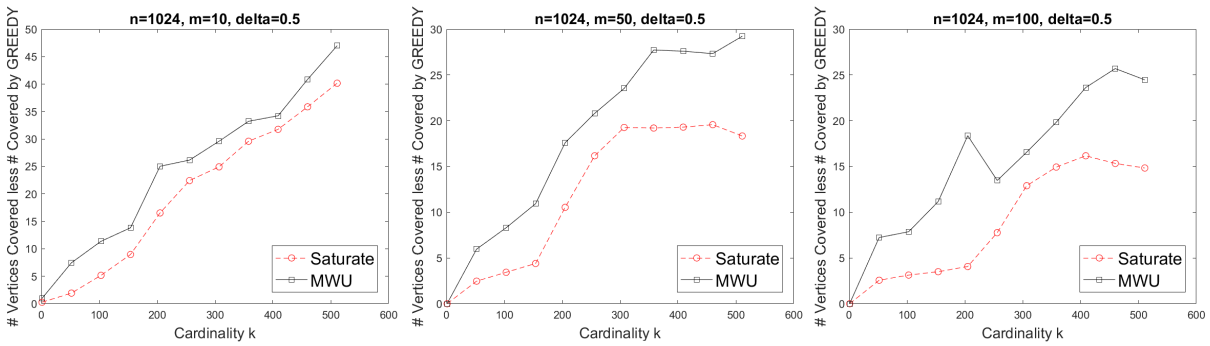

Figure 3-3: Plots for graphs of size 1024. MWU outperforms SATURATE in all cases, with max. gain (on SATURATE) of 6.89% for $m = 10$, 5.02% for $m = 50$ and 7.4% for $m = 100$.

3.6 Conclusion and Open Problems

In summary, we consider the problem of multi-objective maximization of monotone submodular functions subject to a cardinality constraint, when $m = o\big(\frac{k}{\log^3 k}\big)$. No polynomial time constant factor approximations or strong inapproximability results were known for the problem, though it was known that the problem is inapproximable when $m = \Omega(k)$ and admitted a nearly $1 - 1/e$ approximation for constant m. We showed that when $m = o\big(\frac{k}{\log^3 k}\big)$, one can indeed approach the best possible guarantee of $1 - 1/e$ and further also gave a nearly-linear time $(1 - 1/e)^2$ approximation for the same. Finally, we established a natural bound on how the optimal solution value increases with increasing cardinality k of the set, leading to a simple deterministic algorithm

A natural open question here is whether one can achieve approximations right up to $m = o(k)$. Additionally, it also of interest to ask if there are fast algorithms with guarantee closer to $1 - 1/e$, in contrast to the guarantee of $(1 - 1/e)^2$ shown here. Further, it is unclear if similar results can also be shown for a general matroid constraint.

Chapter 4

Deletion-robust Monotone Submodular Function Maximization

In this chapter we study the problem of maximizing monotone submodular functions subject to adversarial removal of elements, formulated as follows,

$$DRO := \max_{A \subseteq N, |A| \leq k} \quad \min_{Z \subseteq A, |Z| \leq \tau} f(A - Z).$$

W.l.o.g., we can turn the inequalities $|A| \leq k$ and $|Z| \leq \tau$ to equalities. The parameter τ controls the degree of robustness of the chosen set, and for $\tau = 0$ the problem reduces to the classical monotone submodular maximization problem SO. It also follows (and is shown formally later) that there is no algorithm with guarantee strictly better than $(1 - 1/e)$ for DRO.

Observe that one can easily cast DRO as a multi-objective maximization problem MO_1 from Chapter 3. For every set $Z \subseteq N$, the function $f_Z(.) = f(.\backslash Z)$ is monotone submodular. Letting $\mathcal{Z} = \{Z \mid Z \subseteq N, |Z| = \tau\}$ we have an equivalent formulation of DRO, $\max_{A \subseteq N, |A| = k} \min_{Z \in \mathcal{Z}} f_Z(A)$. The number of objectives here is $O(n^\tau)$, and due to intractability of MO_1 for $m = \Omega(n)$ this does not yield any approximation results. The approach however, raises the following question – *is it possible to reduce DRO to*

MO_1 with far fewer objectives? Indeed, one of the key ideas in this chapter achieves exactly this, resulting in a $(1-1/e)$ approximation for DRO for $\tau = \frac{o(\log k)}{\log \log k}$. We start here by discussing related work, and refer the reader to Chapter 1 Section 1.1.2 for a discussion on applications of the formulation.

4.1 Related Work

For a discussion of past work on SO we refer the interested reader to Section 3.1, except to recall that the *greedy* algorithm (Algorithm 1) guarantees a $(1-1/e)$ approximate solution, and this is best possible guarantee for SO. However, the greedy algorithm as well as the *continuous* greedy approach for $\tau = 0$ can perform arbitrarily poorly for DRO, even for $\tau = 1$. We show an example of this in Section 4.4, along with a discussion of some other natural approaches that fail to yield a constant factor guarantee for $\tau \geq 1$.

DRO was formally introduced in Krause et al. [KMGG08], where they used the multi-objective maximization framework to solve the problem. Their bi-criterion algorithm for solving MO_1 (SATURATE) when specialized to DRO guarantees optimality by allowing sets up to size $k(1 + \Theta(\log(\tau k \log n)))$ and has runtime exponential in τ. To the best of our knowledge, no stronger/constant factor approximation results were known for DRO prior to our work. Further progress has been made on the problem after the work discussed in this chapter first appeared (in [OSU15]), and we discuss this in Section 4.8.

4.2 Our contributions

We work in the value oracle model and give constant factor guarantees for DRO with combinatorial, 'greedy like' algorithms. To ease presentation, we will usually ignore factors of the form $\left(1 - \frac{O(1)}{k}\right)$ in the approximation guarantees and thus, most of the results presented here are asymptotic in k.

We first study a special case where we construct relatively simple algorithms and

obtain a $(1-1/e)$ approximation for $\tau = o(k)$. Using insights obtained for the special case we then find constant factor approximations for the general case. We start with the case of $\tau = 1$ and find a fast and practical 0.5547 approximation. This is subsequently improved to a $(1-1/e) - 1/\Theta(m)$ algorithm (with runtime exponential in input parameter m), which relies on a subroutine for bi-objective maximization of monotone submodular functions. Reinterpreting this result, we find that an instance of DRO with $\tau = 1$ can be cast as an instance of MO_1 with $m = 3$. This is a vast improvement over the trivial reduction from DRO to $MO_!$ discussed earlier, where $m = n$ for $\tau = 1$.

Leveraging the insights obtained for $\tau = 1$, we show that DRO for larger τ can be cast as MO_1 with $m = 2^{O(\tau \log \tau)}$ objectives. Using our results for MO_1 with $m = o(k)$ from Chapter 3, we have an asymptotic $(1 - 1/e)$ approximation for $\tau = \frac{o(\log k)}{\log \log k}$ (so that $m = 2^{O(\tau \log \tau)} = o(k)$). Therefore, for $\tau = \frac{o(\log k)}{\log \log k}$ we find an algorithm with the best possible asymptotic guarantee.

In an effort ot find a fast and practical algorithm, for $\tau = o\big(\sqrt{\frac{k}{c(k)}}\big)$ we give a nearly linear time $0.387\big(1 - \frac{1}{\Theta(c(k))}\big)$ algorithm, where $c(k) \xrightarrow{k\to\infty} \infty$ is an input parameter that governs the trade off between how large τ can be and how fast the guarantee converges to 0.387.

In the more general case, where we wish to find a deletion-robust set that belongs to an independence system (of which cardinality constraint is a very special case), we extend some of the ideas from the cardinality case into an enumerative procedure that yields an $\alpha/(\tau+1)$ approximation using an α approximation algorithm for $\tau = 0$ as a subroutine. The runtime of our black box procedure scales as $n^{\tau+1}$.

Outline: The outline for the rest of this chapter is as follows: In Section 4.3, we introduce some key notation and show a useful elementary lemma. In Section 4.4 we discuss how several natural ideas fail to give any approximation guarantees and show an elementary hardness of approximation result. In Section 4.4, we start with a special case and slowly build up to an algorithm with asymptotic guarantee $(1 - 1/e)$ for $\tau = \frac{o(\log k)}{\log \log k}$, covering the other results in the process. Finally, in Section 4.7, we extend some of the ideas to more general constraints. Section 4.8 concludes

with some open questions and a brief discussion of more recent work.

4.3 Preliminaries

We denote an instance of DRO on ground set N with cardinality constraint parameter k and robustness parameter τ by (k, N, τ). Subsequently, we use $OPT(k, N, \tau)$ to denote an optimal set for the instance (k, N, τ). Notice that $(k, N, 0)$ denotes an instance of SO over ground set N with cardinality parameter k.

Given a set A, we call a subset Z_τ a *minimizer* of A when $f(A-Z_\tau) = \min_{B\subseteq A;|B|=\tau} f(A-B)$. Also, let $\mathcal{Z}_\tau(A)$ be the set of minimizers of A. When $\tau = 1$, we often use the letter z for minimizers and when τ is otherwise clear from the context and fixed during the discussion we use the shorthand $Z, \mathcal{Z}(.)$. Based on this we also introduce a key function $g_\tau(A) = f(A - Z_\tau)$. Again, we simply use $g(.)$, when τ is clear from context. Similar to the marginal value function for f, define the marginal increase in value of function g due to a set X added to the set A as $g(X|A) = g(A \cup X) - g(A)$.

We generalize the function $\beta(\alpha)$ from Chapters 2 and 3 to $\beta(\eta, \alpha) = \frac{e^\alpha - 1}{e^\alpha - \eta} \in [0, 1]$ for $\eta \in [0, 1]$, $\alpha \geq 0$. Note that for $\eta = 0$ we recover the old $\beta(.)$ function and in particular, $\beta(0, 1) = (1 - 1/e)$. This function appears naturally in our analysis and will be useful for expressing approximation guarantees of the algorithms.

Now, suppose we run Algorithm 1 for n steps and denote the element added at the i-th iteration by a_i. Using this, index the elements in N in the order they were added so, $N = \{a_1, a_2, \ldots, a_n\}$.

Next, we state and prove an elementary but useful lemma, which compares the optimal value of (k, N, τ) with $(k - \tau, N, 0)$.

Lemma 13. *For instances (k, N, τ), we have:*

$$g_\tau(OPT(k, N, \tau)) \leq f(OPT(k - \tau, N - X, 0)) \leq f(OPT(k - \tau, N, 0)),$$

for all $X \subseteq N, |X| \leq \tau$.

Proof. We focus on the first inequality since the second follows by definition. Let $x =$

$\left|X \cap OPT(k, N, \tau)\right| \leq \tau$, then note that $g_\tau(OPT(k, N, \tau)) \leq g_{\tau-x}(OPT(k, N, \tau) - X)$, since the RHS represents the value of some subset of $OPT(k, N, \tau)$ of size $k - \tau$, which upper bounds the LHS by definition. Now, $g_{\tau-x}(OPT(k, N, \tau) - X) \leq g_{\tau-x}(OPT(k - x, N - X, \tau - x))$ by definition. Finally, note that $g_{\tau-x}(OPT(k - x, N - X, \tau - x)) \leq f(OPT(k - \tau, N - X, 0))$ since the LHS represents the value of a set of size $k - x - (\tau - x) = k - \tau$ that does not include any element in X, giving us the desired. □

In the following section, we discuss why some straightforward algorithms fail for DRO, and show results on hardness of approximating DRO.

4.4 Negative Results

The example below demonstrates why the greedy algorithm that gives the best possible guarantee for SO, fails for DRO even when $\tau = 1$. However, the weakness will also indicate a property which will guide us towards better guarantees later.

Example: Consider a ground set N of size $2k$ such that $f(a_1) = 1$, $f(a_i) = 0$, $\forall\, 2 \leq i \leq k$ and $f(a_j) = \frac{1}{k}$, $\forall j \geq k + 1$. Also, for all $j \geq k + 1$, let $f(a_j|X) = \frac{1}{k}$ if $X \cap \{a_1, a_j\} = \emptyset$ and 0 otherwise. Consider the set $S = \{a_{k+1}, \cdots, a_{2k}\}$ and let the set picked by the greedy algorithm (with arbitrary tie-breaking) be $A = \{a_1, \cdots, a_k\}$. Then we have that $f(A - a_1) = 0$ and $f(S - a_j) = 1 - \frac{1}{k}$ for every $a_j \in S$. The insight here is that greedy may select a set where only the first few elements contribute most of the value in the set, which makes it non-robust. However, as we discuss more formally later, such a concentration of value implies that only the first two elements, $\{a_1, a_2\}$, are critical and protecting against removal of those suffices for best possible guarantees.

In fact, many natural variations fail to give an approximation ratio better than $1/(k - 1)$. Indeed, a guarantee of this order, i.e. $1/(k - \tau)$, is achievable for any τ by the following naïve algorithm: *Pick the k largest value elements.* It is also important to examine if the function g is super/sub-modular, since that would make

existing techniques useful. While g is monotonic, it is neither super nor sub modular. Nonetheless, it is interesting to examine a natural variant of the greedy algorithm, where we greedily pick w.r.t. function g instead of f. This variant can also be arbitrarily bad if we pick just one element at each iteration. We refer the reader to Appendix A.1 for some more details.

Finally, it is natural to expect that for any τ we cannot approximate DRO better than SO (which is approximable up to a factor of $\beta(0,1)$) and this is indeed true, as shown below.

Lemma 14. *There exists no polytime algorithm with approximation ratio greater than $(1-1/e)$ for DRO unless P=NP. For the value oracle model, we have the same threshold, but for algorithms that make only a polynomial number of queries.*

Proof. We will give a strict reduction from SO (for which the above hardness result holds [NWF78, Fei98]) to DRO for arbitrary τ. Consider an instance of SO, denoted by $(k, N, 0)$. We intend to reduce this to an instance of DRO on an augmented ground set $N \cup X$ i.e. $(k+\tau, N \cup X, \tau)$.

The set $X = \{x_1, \cdots, x_\tau\}$ is such that $f(x_i) = (k+1)f(a_1)$ (and recall that $f(a_1) \geq f(a_i), \forall a_i \in N$) and $f(x_i|S) = f(x_i)$ for every i and $S \subset N \cup X$ not containing x_i. We will show that $g(OPT(k+\tau, N \cup X, \tau)) = f(OPT(k, N, 0))$.

First, note that for an arbitrary set $S = S_N \cup S_X$, such that $|S| = k+\tau$ and $S_X = S \cap X$, we have that every minimizer contains S_X. This follows by definition of X, since for any two subsets P, Q of S with $|P| = |Q| = k$ and P disjoint with X but $Q \cap X \neq \emptyset$, we have by monotonicity $f(Q) \geq f(x_i) = (k+1)f(a_1) > kf(a_1)$ and by submodularity $kf(a_1) \geq f(P)$. This implies that X is the minimizer of $OPT(k, N, 0) \cup X$ and hence $f(OPT(k, N, 0)) \leq g(OPT(k+\tau, N \cup X, \tau))$

For the other direction, consider the set $OPT(k+\tau, N \cup X, \tau)$ and define,

$$M = OPT(k+\tau, N \cup X, \tau) \cap X.$$

Next, observe that carving out an arbitrary set B of size $\tau - |M|$ from $OPT(k+$

$\tau, N \cup X, \tau) - M$ will give us the set

$$C = OPT(k + \tau, N \cup X, \tau) - M - B,$$

of size $k + \tau - (|M| + \tau - |M|) = k$. Also note that by design, $C \subseteq N$ and hence $f(C) \leq f(OPT(k, N, 0))$, but by definition, we have that $g(OPT(k+\tau, N \cup X, \tau)) \leq f(C)$. This gives us the other direction and we have $g(OPT(k + \tau, N \cup X, \tau)) = f(OPT(k, N, 0))$.

To complete the reduction we need to show how to obtain an α-approximate solution to $(k, N, 0)$ given an α-approximate solution to $(k + \tau, N \cup X, \tau)$. Let $S = S_N \cup S_X$ be such a solution i.e. a set of size $k + \tau$ with $S_X = S \cap X$, such that $g(S) \geq \alpha g(OPT(k + \tau, N \cup X, \tau))$. Now consider an arbitrary subset S'_N of S_N of size $\tau - |S_X|$. Observe that $|S_N - S'_N| = |S| - |S_X| - (\tau - |S_X|) = k$ and further $f(S_N - S'_N) \geq g(S) \geq \alpha g(OPT(k+\tau, N \cup X, \tau)) = \alpha f(OPT(k, N, 0))$, by definition. Hence the set $S_N - S'_N \subseteq N$ is an α-approximate solution to $(k, N, 0)$ which given S, can be obtained in polynomial time/queries.

□

Main Results

Before we start, we would like to remind the reader that the focus of these results is on asymptotic performance guarantee (for large k). In some cases, the results can be improved for small k but we generally ignore those details.

Additionally, in every algorithm that uses the greedy algorithm (Algorithm 1) as a subroutine, especially the fast and practical algorithms 5 ,7 and 10, we can replace the greedy addition rule of adding $x = \arg\max_{x \in S_1} f(x|S_2)$ for some S_1, S_2, by the more efficient thresholding rule from [BV14], reviewed in Chapter 2 Section 2.1. This improves the query/run time to $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$, at the cost of a factor of $(1 - \epsilon)$ in the guarantee.

4.5 Special Case of "copies"

We first consider a special case, which will serve two purposes. First, it will simplify our results and the insights gained for this case generalize. Second, this case may arise in practical scenarios so it is worthwhile to discuss the special algorithms which are much simpler than the general algorithms discussed later.

Given an element $x \in N$, we call another element x' a *copy* of element x if,

$$f(x') = f(x) \text{ and } f(x'|x) = 0.$$

This implies $f(x|x') = f(\{x, x'\}) - f(x') = f(x) + f(x'|x) - f(x') = 0$. In fact, more generally, $f(x'|A) = f(x|A)$ for every $A \subseteq N$, since $f(A + \{x, x'\}) = f(A + x) = f(A + x')$. *This is a useful case for robust sensor placement, if we were allowed to place/replicate multiple sensors at certain locations that are critical for coverage.* Assume that each element in N has τ copies. In the next section we discuss algorithms for this special case when $\tau = 1$. This will help build a foundation, even though the results for $\tau = 1$ are later superseded by the result for $\tau = o(k)$.

4.5.1 Algorithms for $\tau = 1$ in Presence of "copies"

Let a_i' denote the copy of element a_i. As briefly indicated previously, we would like to make our set robust to removal of critical elements. In the presence of copies, adding a copy of these elements achieves this. So as a first step, consider a set that includes a copy of each element, and so is unaffected by single element removal. One way to do this is to run the greedy algorithm for $k/2$ iterations and then add a copy of each of the $k/2$ elements. Then, it follows from Lemma 1 that $g(S) = f(S) \geq \beta(0, \frac{k/2}{k-1}) f(OPT(k-1, N, 0))$ and then from Lemma 13 we have, $g(S) \geq (1 - \frac{1}{\sqrt{e}})\, g(OPT(k, N, 1))$, where we use the fact that $\beta(0, \frac{k/2}{k-1}) > \beta(0, 0.5) = (1 - \frac{1}{\sqrt{e}})$. Hence, we have ≈ 0.393-approximation and the bound is tight. We can certainly improve upon this, one way to do so is to consider if one needs to copy all $k/2$ elements. We show that in fact, copying just a_1 and a_2 is enough. Intuitively, if the

greedy set has value nicely spread out, we could get away without copying anything but nevertheless, in such a case copying just two elements does not affect the value much. Otherwise, as in the example from Section 4.4, if greedy concentrates its value on the first few elements, then copying them is enough.

Before we state and prove this formally, consider the below corollary:

Corollary 15. *Let A be the final set obtained by running the greedy algorithm for l steps on an initial set S. Then we have,*

$$f(A - S|S) \geq \beta\Big(0, \frac{l}{k}\Big) f(OPT(k, N, 0)|S) \geq \beta\Big(0, \frac{l}{k}\Big)\Big(f\big(OPT(k, N, 0)\big) - f(S)\Big)$$

Proof. Follows from Lemma 1, since $f(.|S)$ is monotone submodular for any fixed set S and union of greedy on $f(.|S)$ with S is the same as doing greedy on $f(.)$ starting with S. □

Suppose A, referred to as the greedy set, is the output of Algorithm 1. We now state and prove a simple yet key lemma, which will allow us to quantify how the ratio $\frac{f(A)}{f(OPT(k,N,0))}$ improves over $(1 - 1/e)$, as a function of how concentrated the value of the greedy set is on the first few elements.

Lemma 16. *Starting with initial set S, run l iterations of the greedy algorithm and let A be the output (so $|A| = l + |S|$). Then given $f(S) \geq cf(A)$ for some $c \leq 1$ (high concentration of value on S implies large c), we have,*

$$f(A) \geq \beta\Big(c, \frac{l}{k}\Big) f(OPT(k, N, 0)), \text{ for all } k \geq 1$$

In a typical application of this lemma, we will have S be the first $s = |S|$ elements of the greedy algorithm on $\emptyset$ and $c = s\eta$ for some $\eta \leq 1/s$. Additionally, $|A|$ can be greater than k.

Proof. Denote $f(OPT(k, N, 0))$ by OPT and with the sets A, S as defined above, let $\delta = \frac{f(A)}{OPT}$, which implies $f(S) \geq c\delta OPT$ (by assumption). Also, since we allow $|A|$ to be larger than k, assume $\delta \leq 1$ (otherwise statement is true by default).

Now from Corollary 15 we have, $f(A) = f(S)+f(A-S|S) \geq f(S)+\beta(0,\frac{l}{k})(OPT-f(S))$ and thus,

$$\begin{aligned}
\delta OPT &\geq (1-\beta(0,\tfrac{l}{k})) \times c\delta f(OPT) + \beta(0,\tfrac{l}{k}) f(OPT) \quad \text{[substitution]} \\
\delta(1-c(1-\beta(0,\tfrac{l}{k}))) &\geq \beta(0,\tfrac{l}{k}) \\
\delta(1-c\tfrac{1}{e^{l/k}}) &\geq \tfrac{e^{l/k}-1}{e^{l/k}} \\
\delta &\geq \tfrac{e^{l/k}-1}{e^{l/k}-c} = \beta\Big(c,\tfrac{l}{k}\Big) \\
\implies f(A) &\geq \beta\Big(c,\tfrac{l}{k}\Big) f(OPT(k,N,0))
\end{aligned}$$

□

Let us understand the above lemma with a quick example. Consider the greedy set of the first k elements $A = \{a_1, \ldots, a_k\}$ and let $f(OPT(k,N,0)) = 1$. Now, clearly $f(a_1) \geq \frac{f(A)}{k}$. So using the lemma with $S = \{a_1\}$ and $c = 1/k$ gives us simply that $f(A) \geq \beta\Big(\frac{1}{k},\frac{k-1}{k}\Big) \approx (1-1/e)$, as expected. Next, if $f(a_1) \geq f(A)/2$, then we have that $f(A) \geq \beta\Big(\frac{1}{2},\frac{k-1}{k}\Big) \approx 0.77$, asymptotically much better than $(1-1/e) \approx 0.63$. Similarly, if $f(\{a_1,a_2\}) \geq f(A)/2$ we again have $f(A) \geq \beta\Big(\frac{1}{2},\frac{k-2}{k}\Big) \approx 0.77$. Additionally, we could compare the value $f(A)$ to $f(OPT(k-1,N,0))$ instead, and then replacing k by $k-1$ in the denominator, we have $f(A) \geq \beta\Big(\frac{1}{2},\frac{k-2}{k-1}\Big) f(OPT(k-1,N,0))$.

Now, consider the algorithm that copies the first two elements and thus outputs: $\{a_1, a_1', a_2, a_2', a_3, \ldots, a_{k-2}\}$. Call this the 2-Copy algorithm. Using the above lemma, we show that this algorithm gives us the best possible guarantee asymptotically, aligning with the intuitive argument we presented earlier. Preceding the actual analysis, we show an elementary lemma that we will use frequently.

Lemma 17. *For $0 \leq \eta \leq \frac{1}{3}$ and arbitrary α, $\underset{\eta}{\text{argmin}}\ (1-\eta)\beta(3\eta,\alpha) = 0$.*

Proof.

$$\begin{aligned}
(1-\eta)\beta(3\eta,\alpha) &= \frac{1}{3}(3-3\eta)\frac{e^\alpha-1}{e^\alpha-3\eta} = \frac{1}{3}(e^\alpha-1)\Big(1+\frac{3-e^\alpha}{e^\alpha-3\eta}\Big) \\
&\geq \frac{1}{3}(e^\alpha-1)\Big(1+\frac{3-e^\alpha}{e^\alpha}\Big) = (1-1/e^\alpha) = \beta(0,\alpha)
\end{aligned}$$

□

Theorem 18. *For the case with copies, 2-Copy is* $\beta\big(0, \frac{k-5}{k-1}\big) \xrightarrow{k\to\infty} (1-1/e)$ *approximate.*

Proof. First, denote the output as $A = \{a_1, a_1', a_2, a_2', a_3, \ldots, a_{k-2}\}$ and notice that $f(A) = f(\{a_1, a_2, \ldots, a_{k-2}\})$. As a warm-up, using Lemma 1 we get,

$$f(A) \geq \beta\Big(0, \frac{k-2}{k-1}\Big) f(OPT(k-1, N, 0)) \geq \beta\Big(0, \frac{k-2}{k-1}\Big) g(OPT(k, N, 1)), \quad (4.1)$$

where the second inequality follows from Lemma 13.

Let z be a minimizer of A. We can assume that $z \notin \{a_1, a_1', a_2, a_2'\}$ since all of these have 0 marginal. Now let $f(z|A-z) = \eta f(A)$. We have due to greedy additions and submodularity, $f(a_3|\{a_1, a_2\}) \geq f(z|\{a_1, a_2\}) \geq f(z|A-z)$ and similarly, $f(\{a_1, a_2\}) \geq 2f(z|A-z)$. This implies that $f(\{a_1, a_2, a_3\}) \geq 3\eta f(A)$, which relates the value removed by a minimizer z to the value concentrated on the first 3 elements $\{a_1, a_2, a_3\}$. The higher the value removed, the higher the concentration and closer the value of $f(A)$ to $f(OPT(k-1, N, 0))$. More formally, with $S = \{a_1, a_1', a_2, a_2', a_3\}$, k replaced by $k-1$, $l = k-5$ and $c = 3\eta$ ($\eta \leq 1/3$), we have from Lemma 16,

$$f(A) \geq \beta\Big(3\eta, \frac{k-5}{k-1}\Big) f(OPT(k-1, N, 0)) \geq \beta\Big(3\eta, \frac{k-5}{k-1}\Big) g(OPT(k, N, 1)).$$

Which implies that $g(A) = (1-\eta)f(A) \geq (1-\eta)\beta(3\eta, \frac{k-5}{k-1}) g(OPT(k, N, 1))$. Applying Lemma 17 with $\alpha = \frac{k-5}{k-1}$ finishes the proof. As an example, for $k \geq 55$ the value of the ratio is ≥ 0.6. Additionally, we can use more precise bounds of the greedy algorithm for small k to get better guarantees in that regime. □

The result also implies that for large k, if the output set A, of Algorithm 1 (the greedy algorithm), has a minimizer a_i with $i \geq 3$, then $g(A) = f(A - a_i) \geq (1-1/e) g(OPT(k, N, 1))$, i.e. the greedy algorithm is $(1-1/e)$ approximate for the robust problem in this situation. This is because, for such an instance we can assume that the set contains copies of a_1, a_2 without changing anything and then Theorem 18 applies.

Moreover, if instead of just the first two, we copy the first i elements for $i \geq 3$, we get the same guarantee but with worse asymptotics, so copying more than first two does not result in a gain. On the other hand, copying just one element, a_1, gives a tight guarantee of 0.5 (proof omitted).

Since $(1 - 1/e)$ is the best possible guarantee achievable asymptotically, we now shift focus to case of $\tau > 1$ but $\ll k$, and generalize the above ideas to get an asymptotically $(1 - 1/e)$ approximate algorithm in presence of copies.

4.5.2 $(1-1/e)$ Algorithms for $\tau = o(k)$ in Presence of "copies"

Assume we have τ copies available for each $a_i \in N$. As we did for $\tau = 1$, we would like to determine a small critical set of elements, copy them (possibly many times) and then add the rest of the elements greedily to get a set of size k. In order to understand how large the critical set should be, recall that in the proof of Theorem 18, we relied on the fact that $f(\{a_1, a_2\})$ is at least twice as much as the value removed by the minimizer, and then we could use Lemma 16 to get the desired ratio. To get a similar concentration result on the first few elements for larger τ, we start with an initial set of size 2τ and in particular, we can start with $A_{2\tau} = \{a_1, a_2, \ldots, a_{2\tau}\}$. Additionally, similar to the 2-Copy algorithm, we also want the set to be unaffected by removal of up to τ elements from $A_{2\tau}$. We do this by adding τ copies of each element in $A_{2\tau}$. More concretely, consider the algorithm that greedily picks the set $A_{k-2\tau^2} = \{a_1, a_2, \ldots, a_{k-2\tau^2}\}$, and copies each element in $A_{2\tau} = \{a_1, a_2, \ldots, a_{2\tau}\} \subseteq A_{k-2\tau^2}$, τ times. Denote the set of copies by $C(A_{2\tau})$ and we have $|C(A_{2\tau})| = 2\tau^2$. To summarize, the algorithm outputs the set

$$A = A_{2\tau} \cup C(A_{2\tau}) \cup \{a_{2\tau+1}, \ldots, a_{k-2\tau^2}\}.$$

Observe that when $\tau = 1$, this coincides with the 2-Copy algorithm.

We next show that this algorithm is $\beta(0, \frac{k-2\tau^2-3\tau}{k-\tau}) \xrightarrow{k\to\infty} (1 - 1/e)$ approximate.

Proof. We can assume that $Z \cap (A_{2\tau} \cup C(A_{2\tau})) = \emptyset$ or alternatively $Z \subseteq A_{k-2\tau^2} - A_{2\tau}$, since there are $\tau + 1$ copies of every element (counting the element itself) and Z

cannot remove all. Recall that in the analysis of the 2-Copy algorithm, we showed $f(\{a_1, a_2, a_3\}) \geq 3f(z|A-z)$ and then used Lemmas 16 and 17 to get the desired. Analogously, here we would like to show and $f(A_{3\tau}) \geq 3f(Z|A-Z)$ and do the same.

Next, index elements in Z from 1 to τ, with the mapping $\pi : \{1, \ldots, \tau\} \to \{2\tau+1, \ldots, k-2\tau^2\}$, such that $\pi(i) > \pi(j)$ for $i > j$ and $a_{\pi(i)}$ is the i-th element in Z. Then with $A_i = \{a_1, \ldots, a_i\}$ for all i, we have by submodularity, the following set of inequalities,

$$\begin{aligned} f(a_{\pi(i)}|A_{\pi(i)-1}) \quad &= f(a_{\pi(i)}|A - \{a_{\pi(i)}, \ldots, a_{k-2\tau^2}\}) \\ &\geq f(a_{\pi(i)}|A - \{a_{\pi(i)}, a_{\pi(i+1)}, \ldots, a_{\pi(\tau)}\}) \quad \forall i \\ \implies \sum_{i=1}^{\tau} f(a_{\pi(i)}|A_{\pi(i)-1}) \quad &\geq f(Z|A-Z) \end{aligned} \tag{4.2}$$

where the RHS in (4.2) is by definition.

Note that $\pi(i) > 2\tau$, and for arbitrary $i \in \{1, \ldots, \tau\} = [\tau]$, $j \in \{1, \ldots, 2\tau\} = [2\tau]$, due to greedy iterations we have that $f(a_{\pi(i)}|A_{\pi(i)-1}) \leq f(a_j|A_{j-1})$. So consider any injective mapping from $i \in [\tau]$ to 2 distinct elements $i_1, i_2 \in [2\tau]$, for instance $i_1 = i, i_2 = i + \tau$. We rewrite the previous inequality as,

$$2f(a_{\pi(i)}|A_{\pi(i)-1}) \leq f(a_{i_1}|A_{i_1-1}) + f(a_{i_2}|A_{i_2-1})$$

Summing over all i, along with (4.2) above gives,

$$2f(Z|A-Z) \leq f(A_{2\tau}) \tag{4.3}$$

where the RHS is by injective nature of mapping and definition of $f(.|.)$.

In fact, we have $\pi(i) \geq 2\tau + i$ and thus, $f(a_{\pi(i)}|A_{\pi(i)-1}) \leq f(a_{2\tau+i}|A_{2\tau+i-1})$. From this, we have for the set $A_{3\tau} - A_{2\tau} = \{a_{2\tau+1}, \ldots, a_{3\tau}\}$ that,

$$f(Z|A-Z) \leq f(A_{3\tau} - A_{2\tau}|A_{2\tau}) \tag{4.4}$$

(4.3) and (4.4) combined give us,

$$f(A_{3\tau}) \geq 3f(Z|A-Z)$$

We have from Lemma 13 that $f(OPT(k-\tau, N, 0)) \geq g(OPT(k, N, \tau))$ and combined with using Lemma 16, with k replaced by $k-\tau$, $S = A_{3\tau} \cup C(A_{2\tau})$, $s = 3$ and $l = k - |S| = k - 2\tau^2 - 3\tau$ and $f(Z|A-Z) = \eta f(A)$ gives us,

$$f(A) \geq \beta\Big(3\eta, \frac{k-2\tau^2-3\tau}{k-\tau}\Big) g(OPT(k, N, \tau)).$$

Thus $g(A) = (1-\eta)f(A) \geq \beta\Big(0, \frac{k-2\tau^2-3\tau}{k-\tau}\Big) g(OPT(k, N, \tau))$, follows using Lemma 17. □

Note that while we could ignore the asymptotic factors in approximation guarantee for $\tau = 1$, here we cannot, and this is where the upper bound on τ comes in. Recall that we compare the value $g(OPT(k, N, \tau))$, which is the $f(.)$ value of a set of size $k-\tau$, to the $f(.)$ value of a set of size $k - \Theta(\tau^2)$ (since the $\Theta(\tau^2)$ elements added as copies do not contribute any real value). Now $\frac{k-\Theta(\tau^2)}{k}$ converges to 1 only for $\tau \ll \sqrt{k}$ and it is this degradation that creates the threshold of $o(\sqrt{k})$.

However, it turns out that we don't need to add τ copies of each element in $A_{2\tau}$. Intuitively, the first few elements in $A_{2\tau}$ are more important and for those we should add τ copies, but the later elements are not as important and we can add fewer copies. In fact, we can geometrically decrease the number of copies we add from τ, to 1, over the course of the 2τ elements, resulting in a total of $\Theta(\tau \log \tau)$ copies. The resulting approximation ratio converges to $(1 - 1/e)$ for $\tau = o(k)$.

More concretely, consider the following algorithm,

Algorithm 5 $(1-1/e)$ Algorithm for copies when $\tau = o(k)$

1: Initialize $A = A_{2\tau}, i = 1$

2: **while** $i \leq \lceil \log 2\tau \rceil$ **do**

3: $\quad A = A \cup \left(\lceil \tau/2^{i-1} \rceil \text{ copies for each of } \{a_{2^i-1}, \ldots, a_{2^{i+1}-2}\} \cap A_{2\tau} \right)$; $i = i+1$

4: **while** $|A| < k$ **do** $A = A + \underset{x \in N-A}{\operatorname{argmax}} f(x|A)$

5: Output: A

Essentially, we start with the set $A_{2\tau}$, add τ copies each for $\{a_1, a_2\}$, $\tau/2$ copies for each of $\{a_3, \ldots, a_6\}$, $\tau/4$ copies for each of $\{a_7, \ldots, a_{14}\}$ and so on, finally adding the rest of the $k - \Theta(\tau \log \tau)$ elements greedily. Notice that we could get the best possible guarantee with a minimizer oblivious algorithm i.e., the output is independent of the minimizer at any stage of the algorithm.

Theorem 19. *Algorithm 5 is* $\beta\left(0, \frac{k-\Theta(\tau \log \tau)}{k-\tau}\right) \xrightarrow{k \to \infty} (1-1/e)$ *approximate for* $\tau = o(k)$.

Proof. The basic outline of the analysis is similar to that of the previous one for $\tau = o(\sqrt{k})$, so we focus only on the differences here.

Let A_1 denote the subset of A obtained in the final greedy phase (step 4 in Algorithm 5). Note that if $Z \cap A_{2\tau} = \emptyset$, then we have (4.3) and (4.4) as before and thus $f(A_{3\tau}) \geq 3f(Z|A-Z)$. By applying Lemma 16 with $l = k - \Theta(\tau \log \tau)$ this time, we get the desired. However, unlike the previous analysis, here $Z \cap A_{2\tau}$ need not be empty since we have less than τ copies for many elements. We would like to show that $f(A_{2\tau} - Z) \geq 2f(Z|A-Z)$ regardless. Let $m = \lceil \log 2\tau \rceil - 1$ and in fact, for ease of presentation we assume that $\tau = 2^m$. Also let $B_i = \{a_{2^i-1}, \ldots, a_{2^{i+1}-2}\} \cap A_{2\tau}$ i.e., the elements for which we add $\tau/2^{i-1}$ copies in the algorithm and let C_i denote the set of copies of these elements. Note that $|B_i| = 2^i$ and $|C_i| = 2\tau \geq 2|B_i|$ for all $i \leq m$, for $i = m+1$, $B_{m+1} = \{a_{2\tau-1}, a_{2\tau}\}$ and $|C_{m+1}| = 2$. We can assume that for every element in $A_{2\tau}$ included in minimizer Z, all copies of the element are also present in Z, hence Z removes at most $\lfloor \frac{\tau}{1+\tau/2^{i-1}} \rfloor = \lfloor \frac{|B_i|}{2} \frac{\tau}{\tau+2^{i-1}} \rfloor \leq |B_i|/2 - 1$ elements from $B_i, \forall i \leq m$. So we assume w.l.o.g. $|Z \cap B_1| = 0$.

Let us first examine the case where $Z \subset B_i \cup C_i$ for some $2 \leq i \leq m$ (the case of $i = m+1$ follows rather easily). Notice that $\sum_{j=1}^{i-1} |B_j| = 2(|B_i|/2 - 1)$, then from the observation above we have $2|Z \cap B_i| \leq \sum_{j=1}^{i-1} |B_j|$. This implies that we can injectively map any $|B_i|/2 - 1 = 2^{i-1} - 1$ elements in B_i, to two distinct elements with lower indices in $\cup_{j=1}^{i-1} B_j$. Then, just as we showed in (4.4), we have an injective mapping from every element in $Z \cap A_{2\tau}$ to two distinct elements in $\cup_{j=1}^{i-1} B_j$, and this gives us $f(\cup_{j=1}^{i-1} B_j) \geq 2f(Z|A-Z)$ which further implies $f(A_{2\tau}) \geq f(\cup_{j=1}^{i-1} B_j) + f(Z|\cup_{j=1}^{i-1} B_j) \geq 3f(Z|A-Z)$ and applying Lemmas 16 and 17 completes the case.

For the general case, let $x_j = |Z \cap B_j|, \forall j \in \{2, \ldots, m+1\}$ (with $x_{m+1} \leq 2$) and let $x_{m+2} = |Z \cap A_1|$. From $|Z| = \tau$ and the fact that Z contains all copies of every element in $|Z \cap A_{2\tau}|$, we have $\sum_{j=2}^{m+1} x_j(1 + \frac{\tau}{2^{j-1}}) + x_{m+2} \leq \tau$. Observe that to show the existence of the desired injective mapping, it suffices to show,

$$\begin{aligned} 2x_i \quad &\leq |\cup_{j=1}^{i-1} B_j - Z| - \sum_{j=1}^{i-1} 2x_j \\ &= |\cup_{j=1}^{i-1} B_j| - \sum_{j=1}^{i-1} 3x_j, \forall i \in \{2, \ldots, m+2\} \end{aligned} \tag{4.5}$$

Consider the polytope given by $X = (x_2, \ldots, x_{m+2})$ and constraints $0 \leq x_j \leq \lfloor \frac{|B_i|}{2} \frac{\tau}{\tau + 2^{i-1}} \rfloor \, \forall j$ and $\sum_{j=1}^{m+1} x_j(1 + \frac{\tau}{2^{j-1}}) + x_{m+2} \leq \tau$. Then the extreme points correspond exactly to the special cases we considered so far i.e. (i)$Z \cap A_{2\tau} = \emptyset$ and (ii) $Z \subset B_i \cap C_i$, and we showed that the conditions (4.5) are satisfied for these cases. This implies the (linear) conditions (4.5) are satisfied for every point in the polytope and that completes the proof.

□

4.6 Algorithms in Absence of "copies"

To recap, thus far we chose a set of elements greedily and treated a suitably large subset of the initial few elements as 'critical' and added 'enough' copies of these elements to ensure that we keep a copy of each critical element in the set, even after

adversarial removal. We aim to follow a similar scheme even in the absence of copies. In the general case however, we need to figure out a new way to ensure that our set is robust to removal of the first few critical elements chosen greedily. We first discuss how to approach this for $\tau = 1$.

4.6.1 Algorithms for $\tau = 1$

We start by discussing how one could construct a greedy set that is robust to the removal of a_1. In the case of copies, we would simply add a copy a_1' of a_1, to accomplish this. Here, one approach would be to pick a_1 and then pick the rest of the elements greedily while ignoring a_1. Note that this would add a copy of a_1, if it were available, and if not it will permit the selection of elements which have small marginal on any set containing a_1, but possibly large marginal value in the absence of a_1. Such an element need not be selected by the standard Algorithm 1 that doesn't ignore a_1. Formally,

Algorithm 6 0.387 Algorithm

1: Initialize $A = \{a_1\}$

2: **while** $|A| < k$ **do** $A = A + \underset{x \in N - A}{\operatorname{argmax}} f(x|A - \{a_1\})$

3: Output: A

This simple algorithm is in fact, asymptotically 0.387 approximate and the bound is tight (proof in Appendix A.2). However, a possible issue with the algorithm is that it is oblivious to the minimizers of the set at any iteration. It ignores a_1 throughout, even if a_1 stops being the minimizer after a few iterations. Thus, if we check the minimizer after every iteration and stop ignoring a_1 once it is not a minimizer (i.e. proceed with standard greedy iterations after such a point), we achieve a performance guarantee of 0.5 (proof omitted). Note that this matches the guarantee obtained by copying a_1 in presence of copies. In this sense, we can build a set that is robust to removal of a_1 in the general setting.

As we saw for the case of copies, in order to get even better guarantees, we need to consider the set of the first two elements, $\{a_1, a_2\}$. A direct generalization of line 2

in Algorithm 6, to a rule that ignores both a_1 and a_2, i.e. $\underset{x \in N-A}{\text{argmax}}\ f(x|A - \{a_1, a_2\})$, can be shown to have a performance bound less than 0.5. Algorithm 7 avoids looking at both elements simultaneously. It instead ignores a_1 until its marginal becomes sufficiently small, and then does the same for a_2, if required.

Algorithm 7 $0.5547{-}\Omega(1/k)$ Algorithm

1: Initialize $A = \{a_1, a_2\}$

Phase 1:

2: **while** $|A| < k$ **and** $f(a_1|A - a_1) > \frac{f(A)}{3}$ **do** $A = A + \underset{x \in N-A}{\text{argmax}}\ f(x|A - a_1)$

Phase 2:

3: **while** $|A| < k$ **and** $f(a_2|A - a_2) > \frac{f(A)}{3}$ **do** $A = A + \underset{x \in N-A}{\text{argmax}}\ f(x|A - a_2)$

Phase 3:

4: **while** $|A| < k$ **do** $A = A + \underset{x \in N-A}{\text{argmax}}\ f(x|A)$

5: Output:A

The algorithm is asymptotically 0.5547-approximate (as an example, guarantee > 0.5 for $k \geq 50$) and note that it is minimizer oblivious and easy to implement.

We wish to make the final set robust to removal of either one of a_1, a_2. Algorithm 7 deals with the two elements one at a time, first ignoring a_1 and then a_2 if required. In order to improve on it, we devise a way to add new elements while paying attention to both a_1 and a_2 simultaneously. To this end, consider the algorithm that iteratively adds a set X of size m while one of a_1, a_2 is a minimizer, where X is,

$$X = \underset{|S| \leq m; S \subseteq N-A}{\text{argmax}}\ g(S|A) = \underset{|S| \leq m; S \subseteq N-A}{\text{argmax}} \Big[\min\big\{f(S + A - a_1), f(S + A - a_2)\big\} - \min\big\{f(A - a_1), f(A - a_2)\big\}\Big],$$

i.e., while $z \in \{a_1, a_2\}$, greedily adding m tuples but w.r.t. to the $g(.)$ function now instead of $f(.)$, for suitable $m \geq 1$. We need to resort to m-tuples instead of singletons because, for $m = 1$ we cannot guarantee improvements at each iteration, as there need not be any element that adds marginal value on both a_1 and a_2. However, for larger m we can show improving guarantees. More concretely, consider an instance where

$f(a_1) = f(a_2) = 1$, a_1 has a copy a_1' and additionally both a_1 and a_2 have 'partial' copies, $f(a_i^j) = \frac{1}{k}$ and $f(a_i^j|a_i) = 0$ for $j \in \{1, \ldots, k\}$, $i \in \{1, 2\}$. Also, let there be a set G of $k-2$ garbage elements with $f(G) = 0$. Finally, let $f(\{a_1, a_2\}) = 2$ and $f(a_i^j|X) = \frac{1}{k}$ if $a_i \notin X$ (and also $a_1' \notin X$ for $i = 1$). Running the algorithm with: (i) $m = 1$ outputs $\{a_1, a_2\} \cup G$ in the worst case, with (ii) $m = 2$ outputs a_1^j, a_2^j on step j of Phase 1 and thus, 'partially' copies both a_1 and a_2. Instead, if we run the algorithm with (iii) $m = 3$, the algorithm picks up a_1' and then copies a_2 almost completely with $\{a_2^1, \ldots, a_2^{k-3}\}$. In fact, we will show that while $\{a_1, a_2\}$ are minimizers, adding m-tuples in this manner allows us to guarantee that at each step we increase $g(A)$ by $\frac{m-1}{m}\frac{1}{k}$ times the difference from optimal. Thus, when m is large enough that $\frac{m-1}{m} \approx 1$, we effectively add value at the 'greedy' rate of $\frac{1}{k}$ times the difference from optimal (ref. Lemma 1). However, this is while $z \in \{a_1, a_2\}$, so we need to address the case when $\{a_1, a_2\}$ are not minimizers. One approach would be to follow along the lines of Algorithm 7 by adding singletons greedily w.r.t. f (similar to Phase 3) once the minimizer falls out of $\{a_1, a_2\}$. This is what we do in the algorithm below, recall that $\mathcal{Z}(A)$ is the set of minimizers of set A.

Algorithm 8 A $(1-1/e) - 1/\Theta(m)$ Algorithm for $\tau = 1$

input: m

1: Initialize $A = \{a_1, a_2\}$

Phase 1:

2: **while** $|A| < k$ **and** $\mathcal{Z}(A) \subseteq \{a_1, a_2\}$ **do**

3: $\quad l = \min\{m, k - |A|\}$

4: $\quad A = A \cup \underset{|S|=l; S \subseteq N-A}{\operatorname{argmax}} g(S|A)$

Phase 2:

5: **while** $|A| < k$ **do** $A = A + \underset{x \in N-A}{\operatorname{argmax}} f(x|A)$

6: Output:A

Before analyzing the approximation guarantee of Algorithm 8, we first need to show that in Phase 1, at each step we greedily add an m-tuple with marginal value $\frac{m-1}{k}$ times the difference from optimal. We do this by showing a more general property

below,

Lemma 20. *Given two monotone submodular functions, f_1, f_2 on ground set N. If there exists a set S of size k, such that $f_i(S) \geq V_i, \forall i$, then for every m ($2 \leq m \leq k$), there exists a set $X \subseteq S$ with size m, such that,*

$$f_i(X) \geq \frac{m-1}{k} V_i, \forall i.$$

Proof. First, we show that it suffices to prove this statement for two modular functions h_1, h_2. Note that we can reduce our ground set to S. Now, consider an arbitrary indexing of elements in $S = \{s_1, \ldots, s_k\}$ and let $S_j = \{s_1, \ldots, s_j\}, \forall j \in [k]$. Consider modular functions such that value of element s_j is $h_i(s_j) := f_i(s_j|S_{j-1})$. Note that $h_i(S) = f_i(S)$ and additionally, by submodularity, we have that $h_i(X)$ is a lower bound on $f_i(X)$ i.e. for every set $X \subseteq S$, $f_i(X) = \sum_{j:s_j \in X} f_i(s_j|X \cap S_{j-1}) \geq h_i(X)$. Also, we can assume w.l.o.g. that $h_i(S) = 1, \forall i$ and so it suffices to show that $h_i(X) \geq \frac{m-1}{k}, \forall i$.

We proceed by induction on m. For the base case of $m = 2$, we can pick elements $e_1, e_2 \in S$ such that $h_i(e_i) \geq \frac{1}{k}$ for $i \in \{1, 2\}$, and we are done. Now assume that the property holds for $m \leq p$ and we show it for $m = p + 1$ by contradiction. Consider an arbitrary set X_0 of size p, such that $h_i(X_0) \geq \frac{p-1}{k}$. Such a set exists by assumption, and note that if for some i, say $i = 1$, $h_1(X_0) \geq \frac{p}{k}$, we are done, since we can add an element $e \in S$ to X_0 such that $h_2(e + X_0) \geq \frac{p}{k}$. So we assume that $\frac{p-1}{k} \leq h_i(X_0) < \frac{p}{k}, \forall i$ to set up the contradiction.

Now, consider the reduced ground set $S - X_0$. Then we have that $h_i(S - X_0) > 1 - \frac{p}{k}, \forall i$ and since $|S - X_0| = k - p$, we have for the reduced ground set $S - X_0$, that there exists a set X_1 of size p such that $h_i(X_1) \geq \frac{p-1}{k-p} h_i(S - X_0) > \frac{p-1}{k}$, by the induction assumption. Using our second assumption (for contradiction), we have that $h_i(X_1) < \frac{p}{k}$. We repeat this until $|S - \cup_j X_j| \leq p$. Let $X' = S - \cup_j X_j$, then since $h_i(S) = 1 \, \forall i$, we have $0 < |X'| = p' \leq p$ and $h_i(X') > \frac{p'}{k} \, \forall i$. Now using the induction assumption for $m = p + 1 - p'$ and ground set $S - X'$, we have a set Y with $|Y| = p + 1 - p'$, such that $h_i(X' \cup Y) \geq h_i(X') + \frac{p-p'}{k-p'}(1 - h_i(X')) > \frac{p}{k}, \forall i$, yielding a contradiction. □

Observe that this is a stronger version of Lemma 11 in Chapter 3 for the special case of two objective functions. In fact, we conjecture that the following strengthened version of Lemma 11 is true in general.

Conjecture 21. *Given $m \geq 1$ (treated as a constant) monotone submodular functions $f_1, \ldots, f_m$ on ground set N, a set $S \subseteq N$ of size k, such that $f_i(S) \geq V_i$ for all $i \in [m]$ and an arbitrary l with $m \leq l \leq k$, there exists a set $X \subseteq S$ of size l, such that,*

$$f_i(X) \geq \frac{l - \Theta(1)}{k} V_i, \quad \forall i \in [m]$$

Based on this lemma, consider the below generalized greedy algorithm,

Algorithm 9 Generalized Greedy Algorithm

input: m, V_1, V_2

1: Initialize $A = \emptyset$
2: **while** $|A| < k$ **do**
3: $\quad m = \min\{m, k - |A|\}$
4: $\quad$ Find $\left\{X |\, f_i(X|A) \geq \frac{m-1}{k}[V_i - f_i(A)], X \subseteq N - A, |X| = m\right\}$ by enumeration
5: $\quad A = A \cup X$
6: Output:A

Just as we showed in Chapter 3 Theorem 12, we have that Algorithm 9 is a $(1 - 1/e) - 1/\Theta(m)$ approximation for bi-objective maximization of monotone submodular functions subject to cardinality constraints. As we saw in Chapter 3, the problem of maximizing the minimum of monotone submodular functions, MO_1, is equivalent to the formulation with target values V_i above (MO_2). Hence, for Algorithm 8, we have after l iterations in Phase 1,

$$\begin{aligned}
&g(A) = \min\{f(A - a_1), f(A - a_2)\} \\
&\geq \left(\beta\left(0, \frac{l}{k}\right) - 1/\Theta(m)\right) \min\{f(OPT(k, N, 1) - a_1), f(OPT(k, N, 1) - a_2)\} \\
&\geq \left(\beta\left(0, \frac{l}{k}\right) - 1/\Theta(m)\right) g(OPT(k, N, 1)) \qquad (4.6)
\end{aligned}$$

Note that for Algorithm 9, Lemma 16 applies, albeit with the additive $-1/\Theta(m)$ term. We now present the analysis of Algorithm 8.

Theorem 22. *Given $m \geq 2$, Algorithm 8 is $\beta(0, \frac{k-2m-2}{k}) - 1/\Theta(m) \xrightarrow{k\to\infty} (1-1/e) - 1/\Theta(m)$ approximate, and makes $O(n^{m+1})$ queries.*

Proof. Let $A_0 = \{a_1, a_2\}$. Consider the function $g^0(S) = \min_{i\in\{1,2\}}\{f(S-a_i)\}$ and let z be a minimizer of output set A as usual and define $z^0(S) = \arg\min_{i\in\{1,2\}}\{f(S-a_i)\}$. Note that if $z^0(S) \in \mathcal{Z}(S)$ then $g^0(S) = g(S)$. Also, with the standard definition of marginal, note that for any set $S \cap A_0 = \emptyset$, $g^0(S|X) \geq \min_{i\in\{1,2\}}\{f(S|X - a_i)\} \geq f(S|X)$. Let U be the set added during Phase 1 and similarly W during Phase 2. Also, let $U = \{u_1, \ldots, u_p\}$, where each u_i is a set of size m and similarly $W = \{w_1, \ldots, w_r\}$, where each w_i is a singleton. Let $OPT = g(OPT(k, N, 1))$ and note that if $r = 0$ i.e., Phase 2 doesn't occur, we have that $z \in A_0$ and using (4.6), $g(A) \geq \big[\beta(0, \frac{k-2}{k-1}) - 1/\Theta(m)\big]OPT$. So assume $r > 0$ and also $p \geq 2$, since the case $p = 1$ will be easy to handle later. Now, let $f(z|A - z) = \eta f(A)$, and so $g(A) = (1 - \eta)f(A)$. Using analysis similar to Lemma 16, we will focus on showing that,

$$f(A) \geq \Big[\beta\Big(3\eta, \frac{k-2m}{k-1}\Big) - 1/\Theta(m)\Big]OPT \tag{4.7}$$

Lemma 17 then gives $g(A) \geq \big[\beta(0, \frac{k-2m}{k-1}) - 1/\Theta(m)\big]OPT$.

During the rest of the proof, we sometimes ignore the $1/\Theta(m)$ term with the understanding that it is present by default. To show (4.7), let $a_i = z^0(A_0 \cup U)$. Then, observe that,

$$\begin{aligned} f(A) = \;& g^0(A_0 \cup (U - u_p)) + g^0(u_p|A_0 \cup (U - u_p)) \\ & + f(a_i|(A_0 \cup U) - a_i) + f(W|A_0 \cup U) \end{aligned} \tag{4.8}$$

For the first term in (4.8), we have

$$g^0(A_0 \cup (U - u_p)) = g^0(A_0 \cup u_1) + g^0(\{u_2, \ldots, u_{p-1}\}|A_0 \cup u_1).$$

Then, due to the greedy nature of Phase 1, we have using Theorem **??** (ignoring $1/\Theta(m)$ term),

$$g^0(\{u_2,\ldots,u_{p-1}\}|A_0\cup u_1) \geq \beta\Big(0,\frac{mp-2m}{k-1}\Big)(OPT - g^0(A_0\cup u_1)) \tag{4.9}$$

As usual, the k in the denominator was replaced by $k-1$ because we compare the value to a set of size $k-1$ ($OPT = f(OPT(k,N,1)-z)$). Similarly, for the last term in (4.8), we have,

$$f(W|A_0\cup U) \geq \beta\big(0,\frac{r}{k-1}\big)(OPT - f(A_0\cup U)) \tag{4.10}$$

Now we make some substitutions, let $\Delta = g^0(A_0\cup u_1) + g^0(u_p|A_0\cup(U-u_p)) + f(a_i|(A_0\cup U)-a_i)$, $\alpha_p = \frac{(m-1)(p-2)}{k-1}, \alpha_r = \frac{r}{k-1}$. Then using $k = mp+r+2$ we get, $\alpha_p+\alpha_r = \frac{k-2m}{k-1} - 1/\Theta(m)$. Also, $f(A_0\cup U) = \Delta + g^0(\{u_2,\ldots,u_{p-1}\}|A_0\cup u_1)$. This gives us,

$$\begin{aligned}
&f(A) = \Delta + g^0(\{u_2,\ldots,u_{p-1}\}|A_0\cup u_1) + f(W|A_0\cup U)\\
&\overset{(a)}{\geq} \Delta + g^0(\{u_2,\ldots,u_{p-1}\}|A_0\cup u_1) + \beta(0,\alpha_r)[OPT - f(A_0\cup U)]\\
&\geq \Delta + (1-\beta(0,\alpha_r))g^0(\{u_2,\ldots,u_{p-1}\}|A_0\cup u_1) + \beta(0,\alpha_r)(OPT-\Delta)\\
&\overset{(b)}{\geq} \Delta + (1/e^{\alpha_r})\beta(0,\alpha_p)(OPT - g^0(A_0\cup u_1)) + \beta(0,\alpha_r)(OPT-\Delta)\\
&\geq \Delta + (\beta(0,\alpha_r)+\beta(0,\alpha_p)/e^{\alpha_r})[OPT-\Delta]\\
&= \Delta + \Big[\beta\Big(0,\frac{k-2m}{k-1}\Big) - 1/\Theta(m)\Big][OPT-\Delta]
\end{aligned}$$

where the last equality holds asymptotically, (a) comes from (4.10) and (b) from (4.9). Assume for the time being, that for any x, $3f(x|A-x)\leq\Delta$, which implies that $\Delta \geq 3\eta f(A)$. Armed with these inequalities, the same simple algebra as in the proof of Lemma 16, gives us (4.7). More concretely, ignoring the $1/\Theta(m)$ term, we

have,

$$\begin{aligned} f(A) &\geq \Delta + \beta\Big(0, \frac{k-2m}{k-1}\Big)[OPT - \Delta] \\ &\geq 3\eta f(A)\Big(1 - \beta\Big(0, \frac{k-2m}{k-1}\Big)\Big) + \beta\Big(0, \frac{k-2m}{k-1}\Big)OPT \end{aligned}$$

$$\begin{aligned} (1 - 3\eta e^{-\frac{k-2m}{k-1}})f(A) &\geq (1 - e^{-\frac{k-2m}{k-1}})OPT \\ \implies f(A) &\geq \beta\Big(3\eta, \frac{k-2m}{k-1}\Big)OPT \end{aligned}$$

To finish the proof, we need to show $3f(x|A-x) \leq \Delta$. We break this down by first showing in two steps that for all x, $2f(x|A-x) \leq g^0(A_0 \cup u_1) = f(a_2) + g^0(u_1|A_0)$, followed by proving that $f(x|A-x) \leq g^0(u_p|A_0 \cup (U-u_p)) + f(a_i|(A_0 \cup U) - a_i)$.

Step 1: for all x, $f(x|A-x) \leq f(a_2)$. To see this for $x \neq a_1$, note that $f(x|A-x) \leq f(x|A_0 - x)$ and further, $f(x|A_0 - x) \leq f(a_2|a_1) \leq f(a_2)$, where the first inequality is because a_2 adds maximum marginal value to a_1. For $x = a_1$, since Phase 1 ends, we have that $f(a_1|A - a_1) \leq f(a_1|a_2 + U) \leq f(y|(A_0 \cup U) - y)$ for some y not in A_0 and then we have $f(y|(A_0 \cup U) - y) \leq f(a_2)$.

Step 2: $f(x|A-x) \leq g^0(u_1|A_0)$. For $x \notin A_0$, we have that $f(x|A-x) \leq f(x|A_0) \leq g^0(x|A_0) \leq g^0(u_1|A_0)$. For $x \in A_0$, from the fact that $A_0 \cup U$ has a minimizer $y \notin A_0$, we have that $f(x|A - x) \leq f(x|(A_0 \cup U) - x) \leq f(y|(A_0 \cup U) - y)$ and further $f(y|(A_0 \cup U) - y) \leq f(y|A_0) \leq g^0(y|A_0) \leq g^0(u_1|A_0)$.

Finally, we show that $f(x|A - x) \leq g^0(u_p|A_0 \cup (U - u_p)) + f(a_i|(A_0 \cup U) - a_i)$, for all x. Let $a_j = z^0(A_0 \cup (U - u_p))$ and observe that,

$$\begin{aligned} &g^0(A_0 \cup U) - g^0(A_0 \cup (U - u_p)) + f(a_i|(A_0 \cup U) - a_i) \\ &= f((A_0 \cup U) - a_i) - f((A_0 - a_j) \cup (U - u_p)) + f(a_i|(A_0 \cup U) - a_i) \\ &= f(A_0 \cup U) - f((A_0 - a_j) \cup (U - u_p)) \\ &\geq f(a_j|(A_0 - a_j) \cup (U - u_p)) \end{aligned}$$

Now, before adding u_p, we have that $g^0(.) = g(.)$ and in fact, a_j is a minimizer of $A_0 \cup (U - u_p)$, so clearly, for all $x \in A_0 \cup (U - u_p)$, the desired is true. For $x \in u_p$, it is true since $f(u_p|A_0 \cup (U - u_p)) \leq g^0(u_p|A_0 \cup (U - u_p))$. For $x \in W$, we have that $f(x|A - x) \leq f(x|A_0 \cup (U - u_p)) \leq g^0(x|A_0 \cup (U - u_p)) \leq g^0(u_p|A_0 \cup (U - u_p)))$, and we are done.

The case $p = 1$ can be dealt with same as above, except that now $\Delta = g^0(A_0 \cup u_1) + f(a_i|(A_0 \cup u_1) - a_i) + f(w_1|A_0 \cup u_1) = f(A_0 \cup u_1 \cup w_1)$. □

Before improving and extending the above result to show an asymptotic $(1 - 1/e)$ approximation for $\tau = o(\sqrt{\log k})$, we first give a fast 0.387 approximation for $\tau = o(\sqrt{k})$.

4.6.2 0.387 Algorithm for $\tau \ll \sqrt{k}$

The first algorithm in Section 4.5.2, which greedily chooses $\{a_1, \ldots, a_{k-2\tau^2}\}$ elements and adds τ copies for each of the first 2τ elements. can be recast as greedily choosing 2τ elements, ignoring them and choosing another 2τ greedily (which will be copies of the first 2τ) and repeating this τ times in total, leading to a set which contains $A_{2\tau}$ and $\tau - 1$ copies of each element in $A_{2\tau}$. Then, ignoring this set, we greedily add till we have k elements in total. Thus, the algorithm essentially uses the greedy algorithm as a sub-routine $\tau + 1$ times. Based on this idea, we now propose an algorithm for $\tau = o(\sqrt{k})$, which can also be viewed as an extension of the 0.387 algorithm for $\tau = 1$. To be more precise, it achieves an asymptotic guarantee of 0.387 for $\tau = o(\sqrt{\frac{k}{c(k)}})$, where $c(k)$ is an input parameter that governs the trade off between how fast the guarantee approaches 0.387 as k increases and how large τ can be for the guarantee to still hold. In fact, the guarantee is $0.387\Big(1 - \frac{1}{\Theta(c(k))}\Big)$ with $c(k)$ being a function monotonically increasing in k and approaching ∞ as $k \to \infty$. The factor also degrades proportionally to $1 - \frac{\tau^2 c(k)}{k}$, as τ approaches $\sqrt{\frac{k}{c(k)}}$.

Algorithm 10 Algorithm for $\tau = o\Big(\sqrt{\frac{k}{c(k)}}\Big)$

1: Initialize $\tau' = c(k)\tau^2, A_0 = A_1 = X = \emptyset$.

2: **while** $|A_0| < \tau'$ **do**

3: **while** $|X| < \tau'/\tau$ **do** $X = X + \underset{x \in N-(A_0 \cup X)}{\operatorname{argmax}} f(x|X)$

4: $A_0 = A_0 \cup X$; $X = \emptyset$

5: **while** $|A_1| < k - \tau'$ **do** $A_1 = A_1 + \underset{x \in N-(A_0 \cup A_1)}{\operatorname{argmax}} f(x|A_1)$

6: Output: $A_0 \cup A_1$

Theorem 23. *Algorithm 10 has an approximation ratio of* $\frac{e-1}{2e-1+\frac{e-1}{c(k)}} = \frac{e-1}{2e-1}\big(1 - \frac{1}{\Theta(c(k))}\big) \xrightarrow{k\to\infty} 0.387$ *for* $\tau = o\big(\sqrt{\frac{k}{c(k)}}\big)$.

Proof. Let $A = A_0 \cup A_1$ be the output with A_0, A_1 as in the algorithm. Define $Z_0 = A_0 \cap Z$ and $Z_1 = Z - Z_0 = A_1 \cap Z$. Let $OPT(k-\tau, N - Z_0, 0) = A_0' \cup X$ where $A_0' = OPT(k-\tau, N-Z_0, 0) \cap A_0$ and $X \cap A_0' = \emptyset$. Now note that,

$$\begin{aligned} f(A_0 - Z_0) + f(OPT(k-\tau, N - A_0, 0)) \quad &\geq f(A_0') + f(X) \\ &\geq f(OPT(k-\tau, N - Z_0, 0)) \\ &\geq g(OPT(k, N, \tau)) \quad [\because \text{Lemma 13}] \end{aligned}$$

Which implies,

$$f(A_0 - Z_0) \geq g(OPT(k, N, \tau)) - f(OPT(k-\tau, N - A_0, 0)) \tag{4.11}$$

In addition,

$$f(A_1) \geq \beta\Big(0, \frac{k-\tau'}{k-\tau}\Big) f(OPT(k-\tau, N - A_0, 0)) \tag{4.12}$$

Next, index disjoint subsets of A_0 based on the loop during which they were added. So the subset added during loop i is denoted by A_0^i, where $i \in \{1, \ldots, \tau\}$. So the last subset consisting of $\tau c(k)$ elements is A_0^τ.

Now, if Z_0 includes at least one element from each A_0^i then $Z_1 = \emptyset$ and for this case we have from (4.11) and (4.12) above,

$$\begin{aligned} f(A-Z) \quad &\geq \max\{f(A_0 - Z_0), f(A_1)\} \\ &\geq \max\{g(OPT(k,N,\tau)) - f(OPT(k-\tau, N-A_0, 0)), \\ &\qquad \beta\Big(0, \frac{k-\tau'}{k-\tau}\Big) f(OPT(k-\tau, N-A_0, 0))\} \\ &\geq \frac{\beta\Big(0, \frac{k-\tau'}{k-\tau}\Big)}{1+\beta\Big(0, \frac{k-\tau'}{k-\tau}\Big)} g(OPT(k,N,\tau)) \\ &\xrightarrow{k\to\infty} \frac{e-1}{2e-1} g(OPT(k,N,\tau)) \end{aligned}$$

Next, suppose that $|Z_1| > 0$, then there is some A_0^j such that $A_0^j \cap Z = \emptyset$. Further let $f(Z_1|A-Z) = \eta f(A_1)$, then since $|A_0^j| \geq c(k)|Z_1|$, similar to (4.3), we have due to greedy iterations and submodularity, $f(A_0^j) \geq c(k) f(Z_1|A-Z) = c(k)\eta f(A_1)$. Also note that,

$$f(A-Z) \geq f(A_1 - Z_1) \geq f(A_1) - f(Z_1|A-Z) \geq (1-\eta) f(A_1) \tag{4.13}$$

Moreover, let A_1' be the set of first τ elements of A_1. Then, due to greedy iterations we have $f(A_1') \geq f(Z_1|A_1 - Z_1) \geq \eta f(A_1)$. Thus, from Lemma 16, with N replaced by $N - A_0$, k replaced by $k-\tau$, $S = A_1'$ with $c = \eta$ and $l = k - |S| = k - \tau' - \tau$, we have,

$$f(A_1) \geq \beta(\eta, \frac{k-\tau'-\tau}{k-\tau}) f(OPT(k-\tau, N-A_0, 0)) \tag{4.14}$$

From (4.11), (4.13) and (4.14),

$$
\begin{aligned}
f(A-Z) \quad & \geq \max\{f(A_0 - Z_0), (1-\eta)f(A_1), f(A_0^j)\} \\
& \geq \max\{f(A_0 - Z_0), (1-\eta)f(A_1), c(k)\eta f(A_1)\} \\
& \overset{(a)}{\geq} \max\{g(OPT(k,N,\tau)) - f(OPT(k-\tau, N-A_0, 0)), \\
& \qquad \frac{c(k)}{1+c(k)}\beta(\frac{1}{1+c(k)}, \frac{k-\tau'-\tau}{k-\tau}) f(OPT(k-\tau, N-A_0, 0))\} \\
& \geq \frac{\frac{c(k)}{1+c(k)}\beta\left(\frac{1}{1+c(k)}, \frac{k-\tau'-\tau}{k-\tau}\right)}{1+\frac{c(k)}{1+c(k)}\beta\left(\frac{1}{1+c(k)}, \frac{k-\tau'-\tau}{k-\tau}\right)} g(OPT(k,N,\tau)) \\
& \xrightarrow{k\to\infty} \frac{e-1}{2e-1} g(OPT(k,N,\tau))
\end{aligned}
$$

where (a) follows by substituting $\eta = \frac{1}{1+c(k)}$. □

4.6.3 $(1-1/e) - \epsilon$ Algorithm for $\tau = \frac{o(\log k)}{\log\log k}$

For $\tau = 1$, inspired by the 2-Copy algorithm and using a phase wise approach, we derived a $(1-1/e) - 1/\Theta(m)$ approximation for the general case in Section 4.6.1. In the first phase, where the minimizers are restricted to the set $A_0 = \{a_1, a_2\}$, we build a set robust to removal of either of these elements by using an algorithm for bi-objective maximization of monotone submodular functions. In the second and final phase, we filled in the rest of the set with standard greedy iterations (like Algorithm 1).

However, this phase wise approach doesn't generalize well for $\tau > 1$ since we can have a minimizer that intersects with the initial set but is not a subset of the initial set. Unlike for $\tau = 1$, where a minimizer z is either in $\{a_1, a_2\}$ or not. An alternative approach comes from reinterpreting the result for $\tau = 1$ as follows. We want to build a set that has a large value on both $f_1(.) = f(.|a_1)$ and $f_2(.) = f(.|a_2)$ simultaneously, to deal with the scenarios when either of these elements is a minimizer. Further, we can capture the notion of continuing greedily w.r.t. $f(.)$ once the set becomes robust to removal of either of a_1 or a_2, by considering a third function $f_3(.) = f(.|\{a_1, a_2\})$. Thus, instead of separate phases, we can think about a single multi-objective problem

over three monotone submodular functions f_1, f_2, f_3, and try to add a set A_1 to $A_0 = \{a_1, a_2\}$, such that $f_i(A_1) \geq (1 - 1/e) f_i(OPT(k, N, 1)), \forall i$. To see why this serves our purposes, consider the scenario where a_2 is a minimizer for the final set A,

$$\begin{aligned} g(A) = f(A_1 + A_0 - a_2) &= f_1(A_1) + f(a_1) \\ &\geq f(a_1) + (1 - 1/e)(f(OPT(k, N, 1)) - f(a_1)) \\ &\geq (1 - 1/e) g(OPT(k, N, 1)). \end{aligned}$$

Generalizing this for larger τ, we start with the set A_0 obtained by running Algorithm 10 with $\tau' = 3\tau^2$, implying $|A_0| = 3\tau^2$. Now, consider the monotone submodular functions $f_i(.) = f(.|Y_i)$ for every possible subset Y_i ($|Y_i| \geq 3\tau^2 - \tau$) of A_0 and denote the set of functions by $\mathcal{L}$. Observe that, $|\mathcal{L}| \leq \tau(3\tau^2)^\tau = 2^{O(\tau \log \tau)}$.

Assuming there exists a set S of size $k - 3\tau^2$ such that,

$$f_i(S) \geq \Big(1 - \Theta\Big(\frac{1}{k}\Big)\Big)\big[g(OPT(k, N, \tau)) - f(Y_i)\big].$$

We would like to solve an instance of MO_2 to find a set A_1 of size $k - 3\tau^2$ such that $f_i(A_1) \geq \beta(0, 1)(1 - \Theta(\frac{1}{k}))\big[g(OPT(k, N, \tau)) - f(Y_i)\big]$. We know that $OPT(k, N, \tau)$ is a set such that, $f_i(OPT(k, N, \tau)) \geq f(OPT(k, N, \tau)) - f(Y_i) \geq g(OPT(k, N, \tau)) - f(Y_i)$. So for $\tau = o(\sqrt{k})$, the existence of such a set S follows directly from Lemma 11 in Chapter 3. Now we use the $(1-1/e)$ algorithm for when the number of objectives is $o(k)$, that we developed in Chapter 3, to find set A_1 that $f_i(A_1) \geq (1-1/e) f_i(S)$. The number of objectives here is given by $|\mathcal{L}| \leq 2^{O(\tau \log \tau)}$. Therefore, as long as $\tau = \frac{o(\log k)}{\log \log k}$, we have that the number of functions is $o(k)$ and we can use the algorithms from Chapter 3. Denote the algorithm by $\mathcal{A}$ and let the set output be $\mathcal{A}(f_i, V_i)$, for inputs $(f_i, V_i)_{i=1}^l$.

A final hurdle in using the algorithm for MO_2 is that we need to input the values $V_i = g(OPT(k, N, \tau)) - f(Y_i)$ and hence, we need an estimate of $OPT = g(OPT(k, N, \tau))$. We can overestimate OPT as long as the problem remains feasible. However, underestimating OPT results in a loss in guarantee. Using Algorithm 10, we

can quickly find lower and upper bounds lb, ub such that $lb \leq OPT \leq ub = lb/0.387$ and then run the multi-objective maximization algorithm above with a geometrically increasing sequence of $O(1/\log(1+\delta))$ many values to get an estimate OPT' within factor $(1 \pm \delta)$ of OPT.

In summary, our scheme starts with the set A_0 of size $3\tau^2$, obtained by running Algorithm 10 with $\tau' = 3\tau^2$, then uses the algorithm for multi-objective optimization as a subroutine to find the estimate $OPT' \geq (1-\delta)OPT$ and simultaneously a set $A_1 = \mathcal{A}(\{OPT' - f(Y_i)\}_i, \mathcal{L})$ of size $k - 3\tau^2$, such that $f_i(A_1) \geq (1-1/e)(1 - \Theta(1/k))(OPT' - f(Y_i))$.

The final output is $A = A_0 \cup A_1$ and we next show that this is asymptotically $(1 - 1/e - \epsilon)$ approximate.

Proof. We ignore the ϵ and $\Theta(1/k)$ terms to ease notation. First, note that if $Z \subseteq A_0$, then $f(.|A_0 - Z) \in \mathcal{L}$ gives us, $f(A_1|A_0 - Z) \geq (1-1/e)(OPT' - f(A_0 - Z))$. Hence,

$$\begin{aligned} g(A) \quad &= f(A_0 - Z) + f(A_1|A_0 - Z) \\ &\geq f(A_0 - Z) + (1-1/e)(OPT' - f(A_0 - Z)) \\ &\geq (1-1/e)OPT'. \end{aligned}$$

If $Z \not\subseteq A_0$, then let $Z_1 = Z \cap A_1$ and $Z_0 = Z - Z_1$. Similar to the proof of Theorem 23, let A_0^i denote the ith set of 3τ elements greedily chosen for constructing A_0, $i \leq \tau$. Since $|Z_0| < \tau$, $\exists i$ such that $A_0^i \cap Z_0 = \emptyset$. Then analogous to the proof of Theorem 23, we have due to greedy additions,

$$f(A_0 - Z_0) \geq f(A_0^i) \geq 3f(Z_1|A - Z) \tag{4.15}$$

(in contrast with $c(k)f(Z_1|A - Z)$ in Theorem 23). Now, since $f(.|A_0 - Z_0)$ is one of the functions in $\mathcal{L}$, we have $f(A_1|A_0 - Z_0) \geq (1-1/e)(OPT' - f(A_0 - Z_0))$ which implies,

$$f(A - Z_0) = f(A_0 - Z_0) + f(A_1|A_0 - Z_0) \geq f(A_0 - Z_0) + (1-1/e)(OPT' - f(A_0 - Z_0)).$$

Further, letting $f(Z_1|A-Z) = \eta f(A-Z_0)$ and using (4.15),

$$\begin{aligned} f(A-Z_0) &\geq \frac{3\eta}{e} f(A-Z_0) + (1-1/e)OPT' \\ &\geq \beta(3\eta, 1)OPT'. \end{aligned}$$

Now, using Lemma 17 we have $g(A) = f(A-Z) \geq (1-\eta)f(A-Z_0) \geq \beta(0,1)OPT'$. □

Remark: Note that if one could find a way to reduce the number of objectives considered from $2^{O(\tau \log \tau)}$ to say τ^2, then the above result would extend directly for $\tau = o(\sqrt{k})$. To go further, to say $\tau = o(k)$, one would also need to change the starting set construction as accomplished in [BMSC17]. Also, since we use the randomized $(1-1/e)$ approximation from Chapter 3 for number of functions $m = o(k)$, the above scheme is a randomized approximation. We could instead use the deterministic approximation for constant number of objectives from Chapter 3 to get a deterministic $(1-1/e)(1-\epsilon)$ approximation for constant τ.

4.7 Extension to General Constraints

So far, we have looked at a robust formulation of SO, where we have a cardinality constraint. However, there are more sophisticated applications where we find instances of budget or even matroid constraints. In particular, consider the generalization $\max\limits_{A \in \mathcal{I}} \min\limits_{|B| \leq \tau} f(A \backslash B)$, for some independence system $\mathcal{I}$. By definition, for any feasible set $A \in \mathcal{I}$, all subsets of the form $A \backslash B$ are feasible as well, so the formulation is sensible. Let's briefly discuss the case of $\tau = 1$ and suppose that we are given an α approximation algorithm $\mathcal{A}$, with query/run time $O(R)$ for the $\tau = 0$ case. Let G_0 denote its output and z_0 be a minimizer of G_0. Consider the restricted system $\mathcal{I}_{z_0} = \{A : z_0 \in A, A \in \mathcal{I}\}$. Now, in order to be able to pick elements that have small marginal on z_0 but large value otherwise, we can generalize the notion of ignoring z_0 by maximizing the monotone submodular function $f(. \backslash z_0)$ subject to the independence system I_{z_0}. However, unlike the cardinality constraint case, where this

algorithm gives a guarantee of 0.387, the algorithm can be arbitrarily bad in general (because of severely restricted I_{z_0}, for instance). We tackle this issue by adopting an enumerative procedure.

Let $\mathcal{A}_j$ denote the algorithm for $\tau = j$ and let $\mathcal{A}_j(N, Z)$ denote the output of $\mathcal{A}_j$ on ground set N and subject to restricted system $\mathcal{I}_Z$. Finally, let $\hat{z}(A) = \underset{x \in A}{\operatorname{argmax}}\ f(x)$. With this, we have for general constraints:

Algorithm 11 $\mathcal{A}_\tau : \frac{\alpha}{\tau+1}$ for General Constraints

Initialize $i = 0, Z = \emptyset$
while $N - Z \neq \emptyset$ **do**
    $G_i = \mathcal{A}_0(N - Z, \emptyset)$
    $z_i \in \hat{z}(G_i); \quad Z = Z \cup z_i$
    $M_i = z_i \cup \mathcal{A}_{\tau-1}(N - Z, z_i); \quad i = i + 1$
Output: $\operatorname{argmax}\{g_\tau(S) | S \in \{G_j\}_{j=0}^{i} \cup \{M_j\}_{j=0}^{i}\}$

To understand the basic idea behind the algorithm, assume that z_0 is in an optimal solution for the given τ. Then, given the algorithm $\mathcal{A}_{\tau-1}$, if a minimizer of the set $M_0 = z_0 \cup \mathcal{A}_{\tau-1}(N - z_0, z_0)$ includes z_0, it only removes $\tau - 1$ elements from $\mathcal{A}_{\tau-1}(N - z_0, z_0)$. On the other hand, if a minimizer doesn't include z_0, $g_\tau(M_0) \geq f(z_0) \geq \frac{f(M_0) - g_\tau(M_0)}{\tau}$. These two cases yield the desired ratio, however, since z_0 need not be in an optimal solution, we systematically enumerate.

Theorem 24. *Given an α approximation algorithm $\mathcal{A}$ for $\tau = 0$ with query time $O(R)$, algorithm $\mathcal{A}_\tau$ described above guarantees ratio $\frac{\alpha}{\tau+1}$ for general τ with query time $O(n^\tau R + n^{\tau+1})$*

Proof. We proceed via induction on $j \in \{0, \ldots, \tau\}$. Clearly, for $j = 0$, $\mathcal{A}_0 \equiv \mathcal{A}$, and the statement holds. Assume true for $j \in \{0, 1, \ldots, \tau - 1\}$, then we show validity of the claim for $\mathcal{A}_\tau$. The query time claim follows easily since the while loop runs for at most n iterations and each iteration makes $O(n^\tau + n^{\tau-1}R)$ queries (by assumption on query time of $\mathcal{A}_{\tau-1}$) and updating the best solution at the end of each iteration (counts towards the final output step) takes $O(n^\tau)$ time to find the minimizer by brute force for two sets G_i and M_i.

Now, let $OPT(\mathcal{I}, N, \tau)$ denote an optimal solution to $\max_{A\in\mathcal{I}} \min_{|B|=\tau} f(A-B)$ on ground set N and assume that $z_0 \in OPT(\mathcal{I}, N, \tau)$. For any minimizer B of A, we have for every element $z \in \hat{z}(A)$, $f(z) \geq \frac{f(B)}{\tau} \geq \frac{f(A)-f(A-B)}{\tau}$. Let Z_0 denote a minimizer of G_0. Hence, if $z_0 \notin Z_0$, we have that $g_\tau(G_0) \geq f(z_0) \geq \frac{f(G_0)-g_\tau(G_0)}{\tau}$, giving us $g_\tau(G_0) \geq \frac{f(G_0)}{\tau+1}$. Instead if z_0 is in the minimizer of G_0 and if $f(G_0 - Z_0) < \frac{f(G_0)}{\tau+1}$, then we have that $f(Z_0|G_0 - Z_0) \geq \frac{\tau}{\tau+1} f(G_0)$, implying that $f(z_0) \geq \frac{f(G_0)}{\tau+1}$. Now, let Z_0' denote the minimizer of M_0 and note that if $z_0 \notin Z_0'$, we are done. Else, we have that,

$$\begin{aligned} g_\tau(M_0) = g_{\tau-1}(M_0 - z_0) \quad &\geq \frac{\alpha}{\tau} g_{\tau-1}(OPT(\mathcal{I}_{z_0}, N - z_0, \tau - 1)) \\ &\geq \frac{\alpha}{\tau} g_{\tau-1}(OPT(\mathcal{I}, N, \tau) - z_0) \\ &\geq \frac{\alpha}{\tau} g_\tau(OPT(\mathcal{I}, N, \tau)) > \frac{\alpha}{\tau+1} g_\tau(OPT(\mathcal{I}, N, \tau)) \end{aligned}$$

Where the first inequality stems from the induction assumption, the second and third by our assumption on z_0. This was all true under the assumption that $z_0 \in OPT(\mathcal{I}, N, \tau)$, if that is not the case, we remove z_0 from the ground set and repeat the same process. The algorithm takes the best set out of all the ones generated, and hence there exists some iteration l such that $z_l \in OPT(\mathcal{I}, N, \tau)$ and analyzing that iteration as we did above, gives us the desired.

Finally, for the cardinality constraint case, we can avoid enumeration altogether and the simplified algorithm has runtime polynomial in (n, τ) and guarantee that scales as $\frac{1}{\tau}$, which for $\Omega(\sqrt{k}) \leq \tau = o(k)$, is a better guarantee than the naïve one of $\frac{1}{k-\tau}$ from Section 4.4. □

4.8 Conclusion, Open Problems and Further Work

We looked at a robust version of the classical monotone submodular function maximization problem, where we want sets that are robust to the removal of any τ elements. We introduced the special, yet insightful case of copies, for which we gave a fast and asymptotically $(1 - 1/e)$ approximate algorithm for $\tau = o(k)$.

For the general case, where we may not have copies, we gave a randomized $(1-1/e)$ approximation for $\tau = \frac{o(\log k)}{\log \log k}$. Additionally, we also gave a fast and practical 0.387 algorithm for $\tau = o(\sqrt{k})$. Note that here, unlike in the special case of copies, we could not tune the algorithm to work for larger τ and in fact, there has been further work in this direction since the appearance of this work in [OSU15]. Most notably, [BMSC17] generalizes the notion of geometrically reducing the number of copies from Section 4.5.2, and achieves a 0.387 approximation for $\tau = o(k)$.

Finally, similar robustness versions can be considered for maximization subject to independence system constraints and we gave an enumerative black box approach that leads to an $\frac{\alpha}{\tau+1}$ approximation algorithm with query time scaling as $n^{\tau+1}$, given an α approximation algorithm for the non-robust case.

Chapter 5

Robust Appointment Scheduling

In this chapter we study a robust formulation of the appointment scheduling problem and find simple, nearly optimal heuristics. As briefly discussed in Chapter 1, we will consider two different settings. The first, called *RAS* (for Robust Appointment Scheduling), is about finding optimal appointment start times for jobs that are to served in a given (fixed) order. In the second, called *RASS* (Robust Appointment Scheduling and Sequencing), we must decide the sequence/order of jobs in addition to appointing start times.

Let us review the model and these formulations in some more detail. Suppose we are given n jobs with uncertain service times, all to be served by a single server. We first focus on *RAS*. So the service order is fixed a priori and we would like to appoint start times for every job, so that each job arrives at its appointed start time and is served as soon as previous jobs have been served. While we would like to minimize the time a job has to wait to be served post arrival, there is also a cost associated with keeping the server idle. Therefore, we need to balance these two costs when accounting for the uncertain service times. Index jobs $i \in [n]$ in order opposite to the schedule (i.e. job n is scheduled first followed by job $n-1$ and so on, job 1 is scheduled last). Since we intend to model the uncertainty in service time using a deterministic uncertainty set, we take the simplest uncertainty set and assume that it takes time in the range $[p_i - \hat{\delta}_i, p_i + \delta_i]$ to serve job i for every i. If job i is delayed and ends after the appointed start time for job $i-1$, we have a per unit delay overage cost given by o_i

and similarly, if job i ends before the appointed start time for job $i-1$ we have a per unit underage cost given by u_i. Let $o(i) = \sum_{j=1}^{i} o_j$ and $S_i = [i]$ be the subset of the last i jobs in the schedule. We focus on variables x_i that determine how much time is allocated for a job. W.l.o.g., assume that the last job in the schedule, job 1, also suffers from overage and underage based on its assigned end time (even though no job succeeds it). Let A_i be the allocated appointment/starting time for job i. Then $A_i = \sum_{j=i+1}^{n} x_j$ for $i < n$ and let $A_n = 0$ and $A_0 = \sum_{i=1}^{n} x_i$. Further, let B_i be the allocated end time for job i, and note that $B_i = A_{i-1}$ and $x_i = B_i - A_i$ for every i. Given an arbitrary realization where job i takes time t_i, let C_i denote the completion time for job $i \in [n]$. Letting $C_{n+1} = 0$, we have $C_i = \max\{C_{i+1} + t_i, A_i + t_i\}$. The cost due to job i is given by $\max\{o_i(C_i - A_{i-1}), u_i(A_{i-1} - C_i)\}$, where the first term denotes the overage cost and the second term the underage cost (only one of the two can be strictly positive). As introduced in Chapter 3 Section 1.2, our optimization problem can now be stated as,

$$RAS: \min_{x_1,\ldots,x_n} \quad \max_{t_i \in [p_i - \hat{\delta}_i, p_i + \delta_i]\ \forall i \in [n]} \quad \sum_i \max\{o_i(C_i - A_{i-1}), u_i(A_{i-1} - C_i)\}.$$

The inner problem (maximization), which is also the adversary's problem, finds the worst possible scenario/time profile for jobs given an allocation. It is not difficult to see that an optimal allocation needs to allocate at least $p_i - \hat{\delta}_i$ time for job i, so we can assume w.l.o.g. $p_i - \hat{\delta}_i = 0$ for every job i. Therefore, to simplify notation we let jobs take time between 0 to $\Delta_i = \hat{\delta}_i + \delta_i$.

In the second setting we relax the assumption that order of jobs is fixed a priori. So the sequence of jobs is now a decision variable, and we seek the optimal order as well as time allocation. Let π represent a permutation over the set of first n natural numbers. Then,

$$RASS: \min_{\pi:[n]\leftrightarrow[n]} \min_{x_i} \quad \max_{t_i \in [0, \Delta_i]} \quad \sum_{i \in [n]} \max\{o_{\pi(i)}(C_{\pi(i)} - A_{\pi(i)-1}), u_{\pi(i)}(A_{\pi(i)-1} - C_{\pi(i)})\}. \tag{5.1}$$

Here $A_{\pi(i)}$ and $C_{\pi(i)}$ denote the appointment start time and completion time for

job i when jobs are sequenced according to $\pi(.)$. So $A_{\pi(i)} = \sum_{j|\pi(j)\geq\pi(i)+1} x_j$ and $C_{\pi(i)} = \max\{C_{\pi(i)+1} + t_i, A_{\pi(i)} + t_i\}$. Next, we discuss previous work on this and other related problems.

5.1 Related work

The existing literature on appointment scheduling is quite diverse and includes stochastic optimization models, queueing models, robust optimization as well as distributionally robust optimization models.

We start by discussing more recent work that relates closely to this paper. [MSS14] first introduced the robust formulations RAS and $RASS$ that we consider there. For RAS, they show that when all jobs have identical underage costs ($u_i = u$ for every i), there is a simple closed form solution that gives the optimal allocation. Under the same assumption on underage costs, they also find a $2 + \epsilon$ approximation for $RASS$ and establish a connection to the theory of min-sum scheduling with non-linear objective of completion time, which we further exploit here. This connection also implies a scaling based EPTAS for $RASS$ under homogeneous underage costs, using an EPTAS for min-sum scheduling with concave objective given by [SW]. More recently, [MRZ14] considered a distributionally robust formulation for the problem. Their model incorporates ideas from both stochastic and robust optimization and relies only on marginal moments information for job durations. When the order of jobs is fixed a priori, they show that the problem can be formulated using tractable conic programs. In the setting where the first two moments (mean-variance) are known for every job duration, they formulate the problem as a second-order cone program. Under the same setting, when the order of jobs is flexible, they show that ordering the jobs in increasing order of variance is optimal under additional assumptions. Prior to [MRZ14], [KLTZ13] first considered a distributionally robust formulation but with cross-moments as opposed to marginal moments (for e.g. using and requiring co-variance information as opposed to just variance). They formulate a convex (but not necessarily tractable) program which they then solve by relaxing

to a semi-definite program.

Next, we discuss the relevant literature for all other models. For an overview of the appointment scheduling problem we refer the reader to [CV03], and to [MSS14] and [MRZ14] for other good surverys on past work.

[Wan93] models the problem using a queueing model where the processing times of the jobs are assumed to be i.i.d. exponential random variables. [Wan93] also considers the case where new jobs arrive are released over time and shows that an optimal schedule can be obtained by solving a set of non-linear equations. [Wan99] generalizes the the model to allow for different mean processing times for jobs and shows that in an optimal sequence of execution jobs are processed in increasing order of their mean processing time.

The problem was modeled as a two stage stochastic linear program in [DG03]. They solve the problem using a sequential bounding algorithm and give general upper bounds on the cost of a schedule. For this problem, [RC03] compute near optimal solutions using a Monte Carlo integration technique. They characterize the optimal schedule by showing it has a "dome shaped" structure, where as we move down the schedule the time allocated to jobs first increases and then decreases. They also give heuristics to approximate this structure. [KK07] consider a local search algorithm and prove that it converges to an optimal solution. [GSW06] considers the problem of outpatient appointment scheduling with emergency services and proposes a dynamic stochastic control problem. [BQ11] work with a discrete stochastic model (where job durations are integer random variables with finite support), and reduce the problem to a submodular minimization problem under very general conditions. [BLQ12] extends the idea and proves a similar result for a data driven discrete stochastic model.

5.2 Our Contributions

For the case of appointment scheduling with given job order (RAS), we derive several properties that an optimal allocation exhibits. Notably, we show that for every optimal allocation, the case of all jobs underaged and overaged is a always worst case

for the inner problem. Using this, we formulate a compact LP and show that every LP optimal solution is a 2-approximate allocation. This is the first constant factor approximation for this problem. We then further characterize the optimal allocation and refine our LP formulation. We show that under special conditions, the refined LP has a closed form optimal solution that is also an optimal allocation. This generalizes and sheds new light on results in [MSS14].

When the sequence of jobs is not pre-determined ($RASS$), the problem seems to become harder. We focus primarily on the case of homogeneous underage costs and find a simple ratio based heuristic which we call Customized-Smith's rule (for reasons that will be clear later). This heuristic achieves a 1.06043 approximation and we also show a nearly matching lower bound of 1.06036 for the heuristic. We also give a *locally* optimal algorithm for the problem and show that the algorithm is always at least as good as Customized-Smith's. Finally, for the general problem where underage costs can be arbitrary, we show a $\Theta(n)$ approximation.

Outline: In the rest of the paper we elaborate and formally state and prove the results summarized above. In Section 5.3 we study the traditional setting where job order is fixed a priori. In Section 5.4 we study the flexible job order setting where we first show simple near optimal heuristics for the case of homogeneous underage costs, followed by an approximation bound for the general case. Finally, we conclude in Section 5.5 with some open questions.

5.3 Optimal Appointments Given Job Order

When $u_i = u$, it was previously shown in [MSS14] that the optimal allocation is given by $x_i = \Delta_i \frac{o(i)}{o(i)+u}$. Let x_i^S denote the optimal time allocation for job i when considering only jobs in a subset S that contains i. Note that $x_i^{\{i\}} = \frac{\Delta_i o_i}{o_i+u_i}$. Now, if $u_i = 0$, we allot a very large time slot for job i and jobs on different sides of the slot become independent. So from now on, assume all jobs have strictly positive underage costs, $u_i > 0 \, \forall i$. Also, if $o_1 = 0$ we can assume w.l.o.g. that $x_1 = 0$ and in fact ignore job 1, therefore we assume $o_1 > 0$. In the following we coalesce the w.l.o.g. assumptions

we have made so far,

Assumptions. *W.l.o.g.,*

1. *For job i, time taken lies in* $[0, \Delta_i]$.

2. *For every job i,* $u_i > 0$.

3. *Job 1 has strictly positive overage cost,* $o_1 > 0$.

Now, we say job i is *underaged* if it ends on or before time B_i, and *overaged* otherwise. However, if $\Delta_i = 0$ and job i starts/ends at B_i, we consider it to be both overaged and underaged (for technical reasons). We also use the term strictly underaged (overaged) if a job is underaged (overaged) with non zero cost. Further if job i takes time Δ_i we say it runs for *maximum time*. Observe that if a job i is underaged in some worst case time profile, and it has zero cost, then i must end at B_i and take zero time. Otherwise, we have a strictly worse case by underaging i (simply reduce the time taken by i by some small $\epsilon > 0$).

A perhaps natural question at this point is whether there is always a worst case where all jobs take extreme values (zero or maximum). This is clearly not the case, as demonstrated by the following example.

Consider 3 jobs $3, 2, 1$ scheduled in that order. Let $u_1 = 5$, $u_2 = u_3 = 1$, $o_1 = o_2 = 1$, $o_3 = 3$ and $\Delta_i = 1$ for every job $i \in \{1, 2, 3\}$. Suppose $A_3 = 0, A_2 = 0.25, A_1 = 0.5, B_1 = 1.5$, then intuitively, underaging job 1 and overaging job 3 would result in a bad case. Indeed, when 3 takes time 0.5 (which is neither extreme), job 2 takes time 0 and job 1 takes time 0, we get the unique worst case. Note that in this case the values u_i were inhomogeneous and in fact, one can show that if the $u_i = u$ for every i, then there is a worst case where the jobs take extreme values (though it's certainly possible that some jobs take zero time and some take maximum time).

Given an allocation, another natural question is to efficiently compute a worst case scenario. The answer is not entirely clear but surprisingly, for an optimal allocation some worst cases are remarkably easy to characterize. In particular, we will show that all jobs together taking maximum time and all jobs taking zero time are both worst

cases for every optimal allocation (this is not true for arbitrary allocations, even with homogeneous u_i). This will effectively lead us to a 2-approximation for the general problem, and combined with additional properties shown later, to optimal solutions for several special cases.

Technically, we will need several parts to show this. And we start with some basic properties that hold for arbitrary allocations.

Lemma 25. *Given an arbitrary allocation $\{y_i\}_{i\in[n]}$ and a case Z where job i takes time t_i for every $i \in [n]$. The cost due to job i, $c(Z, i)$, is at least $o_i \sum_{j\geq i}(t_i - y_i)$.*

Proof. We have that $c(Z, i) \geq o_i(C_i - A_{i-1})$ by definition. Now $C_i \geq \sum_{j\geq i} t_j$ and $A_{i-1} = \sum_{j\geq i} y_j$ also by definition. Therefore, we have $c(Z, i) \geq o_i \sum_{j\geq i}(t_j - y_j)$. □

Lemma 26. *Given an arbitrary allocation where all jobs overaged and taking maximum time is a worst case; if the first job scheduled is always forced to start at an arbitrary time $t > 0$ instead, all jobs maximum time is still a worst case with this restriction.*

Proof. Let Y denote the case of all jobs taking maximum time. Think of Y as a representation of the time profile or time taken by jobs, which in the case of Y is Δ_i for job i. When job n (the first scheduled job) starts at time 0, denote the cost of job i by $c(0, Y, i)$. Since Y is a worst case, we have $c(0, Y) = \sum_i c(0, Y, i) = \max_X \sum_i c(0, X, i)$, where X represents possible time profiles for jobs. Consider the restricted setting where job n is always forced to start at time $t > 0$ and let X be a worst case time profile for this setting. Note that $c(t, Y) > c(0, Y)$ and assume that X differs from Y (i.e. all jobs taking maximum time). So there exists a job that is underaged in X and let j be the largest index job underaged in X. We can assume $j < n$ since if n is underaged then $c(t, X) < c(0, X)$ for $t > 0$, where the terms denote the total cost of X with job n starting at t and 0 resp. So when n starts at t, jobs $j + 1$ to n are all overaged and job j is underaged in X.

Now consider X in the normal setting where n starts at 0. Note that some jobs in $j + 1$ to n might be underaged now and job j is still underaged. We claim that $c(t, X, i) \leq c(0, X, i) + to_i$ for every i. Let x_j denote the allocated duration for job j.

Then the claim follow easily for jobs in $[j]$, since j is underaged. For jobs in $[n]-[j]$, if job i runs for time t_i then observe that $c(t, X, i) = o_i(t + \sum_{j \geq i} t_i - \sum_{j \geq i} x_j) \leq o_i t + c(0, X, i)$ due to Lemma 25. Therefore, we have for the case Y of all jobs overaged and n starting at time t, $\sum_i c(t, Y, i) = \sum_i \big(c(0, Y, i) + to_i\big) \geq \sum_i \big(c(0, X, i) + to_i\big) \geq \sum_i c(t, X, i)$ for every X. □

Lemma 27. *In every optimal allocation, for every job i, there is a worst case where the job is underaged and takes zero time as well a worst case where it is overaged.*

Proof. Let us fix arbitrary job i, we will prove the claim in parts by considering three cases; when i is strictly overaged or strictly underaged in all worst cases, when it is underaged in all cases, when it is overaged in all cases.

First, consider the case where the job is strictly underaged in all worst cases. Then reducing x_i leads to a better allocation, yielding a contradiction. Note that $x_i > 0$ since we assumed i is strictly underaged. If i is always strictly overaged then we have a similar situation but with an added subtlety. Observe that we can reduce the cost in every worst case by increasing x_i by some small $\epsilon > 0$, if $x_j = 0$ for every $j < i$. Otherwise, let k be the largest index smaller than i such that $x_k > 0$. To reduce the worst case cost we then increase x_i by ϵ and also reduce x_k by ϵ. Therefore, in an optimal allocation no job can be strictly overaged nor strictly underaged in every worst case. Recall that being charged for overage means job is completed after scheduled completion time B_i, which could occur even if job takes zero time.

Now let us examine the scenario where job i is always underaged but there is a worst case where i has zero cost. In fact, assume i is the job with smallest index that has this property, and we show this leads to a contradiction. We also have $\Delta_i > 0$, since if $\Delta_j = 0$ for some job j and there is a worst case where j is underaged with zero cost, it is also overaged (by definition). Note that being underaged, job i takes zero time in every worst case. Note further that $i > 1$ since once can always overage job 1 if it is underaged with zero cost. Now consider some worst case, call it X, where i contributes zero cost and starts/ends at B_i, and suppose that $i - 1$ takes non-zero time in X. We get another worst case with i taking time ϵ (thus becoming overaged)

and reducing the time taken by $i-1$ by ϵ, for small enough $\epsilon > 0$. Therefore, in every worst case where i has zero cost, $i-1$ starts at A_{i-1} and takes zero time and hence is underaged. However, since i ends on or before B_i in every worst case, if there was a worst case Y where $i-1$ was overaged, then by copying X for all jobs in $[n]-[i-1]$ and copying Y for all jobs in $[i-1]$, we have a worst case where i has zero cost and $i-1$ is overaged. Therefore, we more generally have that $i-1$ must be underaged in all worst cases. Further, $i-1$ must be strictly underaged in all worst cases due to our assumption on i as the minimum index job which is not strictly underaged, contradiction.

Finally, consider the remaining case where job i is always overaged but there is a worst case where i is overaged with zero cost. Indeed this only occurs if $o_i = 0$ and thus i is overaged with zero cost in all worst cases. Further assume that i is the highest index job with this property. Therefore job $i+1$ (if $i \leq n-1$) is underaged in some worst case since it falls into one of the cases resolved earlier in the proof. Also note $i > 1$ since $o_1 > 0$ by assumption. Now we could still increase x_i by a small $\epsilon > 0$ and decrease x_{i-1} by the same if possible, but this will not result in a strictly better allocation. Recall however that there is a worst case, say Z, where $i+1$ is underaged and thus i starts at A_i (trivially true if $i = n$). Job $i-1$ must be overaged in Z since otherwise we also have a worst case by underaging i. More generally, let j be the largest index smaller than i, which is underaged in some worst case (not necessarily Z). Now, for every $k, j < k \leq i$, job k is overaged in all worst cases and thus o_k must be zero. So just like job $i-1$, job j must also be overaged in Z otherwise we would also have a worst case by underaging all jobs in $\{j+1, \ldots, i\}$ and keeping other jobs as in Z. So we have that all jobs in $\{j, \ldots, i\}$ are overaged in Z. We more strongly have that there can be no case that underages j and yet has total cost contributed by jobs in $[j]$ matching (or exceeding) the total cost due to subset $[j]$ in Z. This follows by observing that i starts at A_i and the cost due to jobs $\{j+1, \ldots, i\}$ is zero, therefore we can reduce the starting time of j down to A_j if we desired and emulate any time profile for jobs in $[j]$ while mimicking Z for jobs in $[n]-[i]$.

Next, let Y be a worst case where j is underaged. Y exists by definition of j.

Since j is underaged we can assume that all jobs in $\{j+1, \ldots, i\}$ take zero time in Y, otherwise we can construct a strictly worse case. Now, we claim that one can construct a strictly worse case which mimics Y for jobs in $[n] - [i]$ and Z for jobs in $[j]$. This follows by observing that we can manipulate the times of jobs in $\{j, \ldots, i\}$ to end j at the same time as its completion time in Z, thus overaging j and allowing us to mimic the total cost contribution from jobs in $[j]$ from Z. Since this cost from jobs in $[j]$ is strictly larger than the counterpart in every case where j is underaged (which it is in Y), we have a case with strictly worse cost than Y and thus a contradiction.

□

Lemma 28. *In every optimal allocation over n jobs, the case of all jobs taking maximum time (Δ_i for job i) is a worst case.*

Proof. We will show this via induction over the subset of jobs $[i]$ for $i \leq n$ and fixed n. First consider $i = 1$. Then by Lemma 27 there is a worst case where job 1 is overaged, and hence takes time Δ_1 (since there is no job scheduled afterwards). Now assume that there is a worst case where all jobs from 1 to $k-1$ take maximum time (and are overaged) and call it case Y. We will show that there exists a worst case where jobs 1 to k all take maximum time.

If job k is overaged in Y, then it clearly takes maximum time. Therefore let us assume that job k is underaged, and so takes zero time in Y. Now if we restrict attention to the subset of jobs $[k-1]$ (and ignore other jobs), the case of all jobs taking maximum time is a worst case. By Lemma 27, there exists a worst case where job k is overaged, call it case X. Then the case which matches X from jobs $k+1$ to n and runs jobs 1 to k for maximum time is also a worst case, since it matches the cost of jobs $k+1$ to n from X and total costs from jobs in $[k]$ can only be larger than the same total in X due to Lemma 26. □

Corollary 29. *For every k, $\sum_{j=k}^{n} x_j \leq \sum_{j=k}^{n} \Delta_j$.*

Proof. Suppose this is untrue, and let k_0 be the smallest index such that $\sum_{j=k_0}^{n} x_j > \sum_{j=k_0}^{n} \Delta_j$. This violates Lemma 28 since job k_0 can never be overaged and therefore the case of all jobs taking maximum time cannot be a worst case. □

Lemma 30. *In every optimal allocation over n jobs, the case of all jobs taking zero time is a worst case.*

Proof. We will prove this via induction on the subset of jobs $[n] - [i]$, starting with $i = n - 1$ and keeping n fixed. For the base case of job n, Lemma 27 implies there is a worst case where job n takes zero time. Assume there is a worst case, call it Y, where jobs $k + 1$ to n all take zero time. We wish to show that there is a worst case where job k to n are all underaged, so assume job k is overaged in Y (otherwise we are done). Again, Lemma 27 implies there is a worst case, call it X, where job k is underaged and takes zero time.

Now job k is overaged in Y (and starts at A_k) and underaged in X (ends before A_{k-1}), and suppose it terminates at times t_Y and t_X in Y and X respectively (t_X is also the start time for job k in X). Now consider a new case Z, formed by a combination of X and Y. In Z, jobs $k + 1$ to n are as given by X and jobs 1 to $k - 1$ are as in Y. We fill the gap by running job k from t_X to t_Y. This is feasible since $t_Y - t_X \leq t_Y - A_k \leq \Delta_k$. Let $c(., i) : \{X, Y, Z\} \to \mathbb{R}$ denote the cost of job i in various cases. Observe that, $\sum_i c(Z, i) = \sum_{i=k+1}^{n} c(X, i) + \sum_{i=1}^{k} c(Y, i)$. Now since Y is a worst case we also have,

$$\begin{aligned} \sum_{i=k+1}^{n} c(X,i) + \sum_{i=1}^{k} c(Y,i) &\leq \sum_{i=1}^{n} c(Y,i) \\ \sum_{i=k+1}^{n} c(X,i) &\leq \sum_{i=k+1}^{n} c(Y,i) \end{aligned}$$

Now, consider another hybrid case Q, where jobs $k + 1$ to n are all underaged and take zero time as in Y and jobs 1 to $k - 1$ are as in X. Job k starts/ends at A_k in Q and hence $c(X, k) \leq c(Q, k)$. Then we have,

$$\sum_i c(X,i) \leq \sum_{i=k+1}^{n} c(Y,i) + \sum_{1}^{k} c(X,i) \leq \sum_{i=k+1}^{n} c(Y,i) + c(Q,k) + \sum_{1}^{k-1} c(X,i) = \sum_i c(Q,i).$$

Hence, we have a worst case with jobs k to n all underaged. □

Remarks: As mentioned before, Lemmas 28 and 30 are particularly interesting

since they are not true for arbitrary allotments. In fact, it's not clear right away if the worst case can be easily found for arbitrary allotments, but the structure of an optimal allotment renders the adversary's problem easy.

Corollary 31. *The worst case cost for an optimal allocation (x_i for job i) is given by $\sum_{i=1}^{n} o(i)(\Delta_i - x_i) = \sum_{i=1}^{n} u_i x_i$.*

Proof. Using Lemma 28 we have that the worst case cost is $\sum_{i=1}^{n} o_i(\sum_{j=i}^{n} \Delta_j - \sum_{j=i}^{n} x_j) = \sum_{i=1}^{n} o_i \sum_{j=i}^{n}(\Delta_j - x_j)$. Rearranging summation we get $\sum_{i=1}^{n} o_i \sum_{j=i}^{n}(\Delta_j - x_j) = \sum_{j=1}^{n}(\Delta_j - x_j)o(j)$. This cost is equal to $\sum_{i=1}^{n} u_i x_i$ due to Lemmas 28 and 30. □

With all the above properties, we are now ready to show a simple 2-approximation for finding the optimal allocation. Consider the following LP,

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} u_j y_j \\
s.t. \quad & \sum_{j=1}^{n}(u_j + o(j))y_j = \sum_{j=1}^{n} o(j)\Delta_j && (5.2) \\
& \sum_{j=k}^{n}(y_j - \Delta_j) \leq 0 \quad \forall k && (5.3) \\
& y_j \geq 0 \quad \forall j \in [n]
\end{aligned}$$

Due to constraints (5.2) and (5.3), an optimal solution to the above LP is an allocation that satisfies Corollaries 31 and 29 respectively. An optimal allocation is therefore a feasible solution to the LP. However, an optimal solution to the LP need not have all jobs underaged with zero time as a worst case, and hence need not be an optimal allocation.

Theorem 32. *The worst case cost of the LP optimum is at most twice the optimal allocation cost and therefore, the LP is a 2-approximation.*

Proof. Let x_i^* denote the LP optimal and x_i an optimal allocation. Since x_i is a feasible solution to the LP, we have that the worst case cost of optimal allocation,

given by $\sum_i u_i x_i$, is at least $\sum_i u_i x_i^*$. Further, for any an arbitrary allocation, the worst case cost is at most the sum of costs of all jobs overaged and all jobs underaged. Therefore, for the allocation given by x_i^*, we have that the worst case cost is at most $\sum_i u_i x_i^* + \sum_i o(i)(\Delta_i - x_i^*) = 2\sum_i u_i x_i^* \leq 2\sum_i u_i x_i$. □

Lemma 33. *There is an instance where the LP optimal results in a 2-approximate allocation.*

Proof. Consider two jobs, with job 2 scheduled first, followed by job 1. Job 1 takes $\Delta_1 = 1$ unit time and job 2 takes time $\Delta_2 = \epsilon \to 0$. Further, let $u_1 = o_1 = 1$ and $u_2 = 1/\epsilon, o_2 = 1/\epsilon - (1+\epsilon)$ and therefore, $\frac{u_1}{u_1+o_1} = 0.5 \lessapprox \frac{u_2}{u_2+o(2)}$. Consider the allocation $x_1^* = 0.5$ and $x_2^* = \epsilon\frac{o(2)}{u_2+o(2)} \approx \epsilon/2$. It is easy to see that the worst case of the allocation is when both jobs are underaged (overaged) and hence the cost of this allocation is $u_1 x_1^* + u_2 x_2^* \approx 1$.

Now, consider the solution $x_1 = 0.5 + \frac{\Delta_2 o(2)}{u_1+o_1} \approx 1$ and $x_2 = 0$. This is an optimal solution to the LP for $\epsilon \to 0$ and the worst case occurs when job 1 is underaged and job 2 is overaged. The cost of the induced allocation is therefore, $\approx o_2\Delta_2 + u_1(x_1 - \Delta_2) = 2$. □

Next, we show additional properties that guide us to an optimal allocation for various special cases. However, these properties will be insufficient to get an optimal allocation for the general case, which we leave as an open problem.

5.3.1 Optimal Allocation for Special Cases

We will show a new set of constraints in Lemma 35 that will help refine the LP and find an optimal allocation under certain conditions. First, consider the following helper lemma.

Lemma 34. *Recall x_i^S denotes the time allocation for job i in an optimal allocation over jobs in set S. We have,*

$$\sum_{i\in[n-1]} u_i x_i^{[n-1]} \leq \sum_{i\in[n-1]} u_i x^{[n]}.$$

Proof. Note that the LHS denotes the worst case cost of an optimal allocation over $[n-1]$. Consider the possibly sub-optimal allocation $x_i^{[n]}$ for jobs $i \in [n-1]$. The RHS denotes the cost of all jobs underaged under this allocation, however a priori this need not be a worst case cost for the allocation. So to show the inequality we prove that all jobs underaged is a worst case for the allocation in RHS.

Assume this is not the case i.e., there is a case where some job in $[n-1]$ takes non-zero time and this case, call it Y, is strictly worse than all jobs taking zero time. Let the cost of Y be $c(Y)$, and we have $c(Y) > \sum_{i\in[n-1]} u_i x_i^{[n]}$. Now consider the set of jobs $[n]$ and the now optimal allocation $x_i^{[n]}$ over this set. We have that $\sum_{i\in[n]} u_i x_i^{[n]} < u_n x_n^{[n]} + c(Y)$, which is a contradiction due to Lemma 30 since the RHS induces a case (job n takes zero time, other jobs are as in Y) that is strictly worse than all jobs being underaged. □

Lemma 35. *In every optimal allocation over n jobs, $\sum_{j=k}^{n} u_j x_j \leq \sum_{j=k}^{n} o(j)(\Delta_j - x_j)\, \forall k \leq n$.*

Proof. Lemmas 28 and 30 showed that for $n = 1$ we in fact have an equality, since in that case the two sides represent cost when the job takes time 0 and cost when it takes maximum time. The general case is more subtle and we give a proof by induction on k, with $n > 1$ fixed.

For the base case of $k = n$, suppose the statement is false i.e., $u_n x_n > o(n)(\Delta_n - x_n)$. We derive a contradiction by constructing a new allocation that is better than the optimal. We use x'_i to denote the new allocation for job i. Then,

$$x'_n = \frac{o(n)\Delta_n}{u_n + o(n)} < x_n, \tag{5.4}$$

and the allocation for jobs 1 to $n-1$ is an optimal allocation for the subset $[n-1]$ i.e., $x'_i = x_i^{[n-1]}$ for every $i < n$. Now we show that in the new allocation, the total

cost when all jobs take maximum time equals total cost when all jobs take zero time.

$$\begin{aligned}\sum_{i=1}^{n} o(i)(\Delta_i - x'_i) &= o(n)(\Delta_n - x'_n) + \sum_{i=1}^{n-1} o(i)(\Delta_i - x'_i) \\ &= u_n x'_n + \sum_{i=1}^{n-1} u_i x'_i.\end{aligned}$$

Where the second equality follows from (5.4) and Corollary 31. Moreover, this cost is strictly smaller than the worst case cost in the original allocation i.e., $\sum_{i=1}^{n} u_i x'_i < u_n x_n + \sum_{i=1}^{n-1} u_i x'_i \leq \sum_{i=1}^{n} u_i x_i$, where the last inequality follows from Lemma 34. To finish the proof for the base case we need to show that all jobs taking maximum time or alternatively all taking zero time is a worst case in the new allocation. We argue this by considering two cases based on job n, while using the fact that the new allocation is optimal when restricted to the set $[n-1]$. If there is a worst case where n is overaged, then all jobs taking maximum time is a worst case due to Lemma 26. Similarly, if job n takes zero time in some worst case then all jobs taking zero time is a worst case due to Lemma 30. This completes the base case.

Next, assume we are given $\sum_{j=i}^{n} u_j x_j \leq \sum_{j=i}^{n} o(j)(\Delta_j - x_j)$ for every $i \geq k+1$ and we need to show the inequality for $i = k$ i.e., $\sum_{j=k}^{n} u_j x_j \leq \sum_{j=k}^{n} o(j)(\Delta_j - x_j)$. We prove by contradiction, so suppose the condition is false i.e., $\sum_{j=k}^{n} u_j x_j > \sum_{j=k}^{n} o(j)(\Delta_j - x_j)$. Then similar to the base case, we construct a new improved allocation that will yield a contradiction. In the new allocation we let,

$$x'_k = \frac{1}{u_k + o(k)}[o(k)\Delta_k + \sum_{j=k+1}^{n} o(j)(\Delta_j - x_j) - \sum_{j=k+1}^{n} u_j x_j] < x_k.$$

Further, $x'_i = x_i$ for every $i \geq k+1$ and $x'_i = x_i^{[k-1]}$ for every $i \leq k-1$. Same as the base case, observe that in the new allocation the cost when all jobs take zero time is the same as the cost when all take maximum time. So we just need to show that at least one of them is a worst case.

For the new allocation, if we have a worst case where k is underaged, then there is in fact a worst case where where all jobs from 1 to k are underaged since Lemma

30 gives us that underaging all jobs from 1 to $k-1$ is a worst case for every optimal allocation on subset $[k-1]$. If there is no worst case where all jobs are underaged, then let Y be a worst case where jobs 1 to k are all underaged. Let $c(Y,i)$ denote the cost contributed by job i in worst case Y. Then we have that sum of costs from jobs in $[n]-[k-1]$ is strictly larger than the cost of underaging those jobs, $\sum_{i=k}^{n} c(Y,i) > \sum_{i=k}^{n} u_i x'_i = \sum_{i=k+1}^{n} u_i x_i + u_k x'_k$. Therefore, $\sum_{i=k}^{n} c(Y,i) + u_k(x_k - x'_k) > \sum_{i=k}^{n} u_i x_i$ and now going back to the original allocation given by x_i for job i, we find a contradiction to Lemma 30 since the cost due to all jobs underaged is strictly smaller than the cost when jobs in $[n]-[k-1]$ are as in Y and all jobs in $[k-1]$ are underaged.

Finally, suppose there is no worst case for the new allocation (given by x'_i) where k is underaged, or in other words k is overaged in all worst cases. Then due to Lemma 26 we know there is a worst case where all jobs in $[k]$ are overaged and take maximum time. Now if there also exists a worst case where all jobs in $[n]-[k]$ are overaged, then it is not hard to see that all jobs overaged with maximum time is a worst case. So assume that in every worst case some job in $[n]-[k]$ is underaged. Then let us consider a worst case, call it Z, where all jobs in $[k]$ are overaged (with maximum time) and j is the lowest index job in $[n]-[k]$ that is underaged in Z. Since Z is strictly worse than the case of all jobs overaged we have that,

$$\begin{aligned} & \sum_{i=1}^{n} o(i)(\Delta_i - x'_i) && < \sum_{i=1}^{n} c(Z,i) \\ \implies & \sum_{i=j}^{n} o(i)(\Delta_i - x_i) && < \sum_{i=j}^{n} c(Z,i). \end{aligned} \tag{5.5}$$

Where the second inequality follows from $\sum_{i=1}^{j-1} c(Z,i) = \sum_{i=1}^{j-1} o(i)(\Delta_i - x'_i)$ and $x'_i = x_i$ for $i \geq j$. Now going back to the original allocation given by x_i for job i, consider a new case where jobs in $[n]-[j-1]$ are as in Z and jobs in $[j-1]$ all take maximum time. Note that jobs in $[j-1]$ need not all be overaged in this case, but by Lemma 25 the net cost due to jobs in $[j-1]$ is at least $\sum_{i=1}^{j-1} o_i(\sum_{j \geq i} \Delta_j - x_j) = \sum_{i=1}^{j-1}(\Delta_i - x_i)o(i)$. Combining this with (5.5) we have that for the original allocation, the case that mimics Z for jobs in $[n]-[j-1]$ and sets all jobs in $[j-1]$ to maximum

time is strictly worse than overaging all jobs, contradicting Lemma 28.

□

Now, based on the above properties, consider the following LP,

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} u_j y_j \\
s.t. \quad & \sum_{j=k}^{n} u_j y_j \leq \sum_{j=k}^{n} o(j)(\Delta_j - y_j) \quad \forall 2 \leq k \leq n && (5.6) \\
& \sum_{j=1}^{n} u_j y_j = \sum_{j=1}^{n} o(j)(\Delta_j - y_j) && (5.7) \\
& \sum_{j=k}^{n} (y_j - \Delta_j) \leq 0 \quad \forall k \\
& y_j \geq 0 \quad \forall j \in [n]
\end{aligned}$$

As compared to the previous LP, we have a new set of constraints (5.6) due to Lemma 35. However, every optimal allocation is still a feasible solution to the LP. Observe that, due to (5.7) we can rewrite constraints (5.6) as $\sum_{j=1}^{k} u_j y_j \geq \sum_{j=1}^{k} o(j)(\Delta_j - y_j)$ for every $k \in [n-1]$. Modifying this further, we have the following alternative formulation,

$$\begin{aligned}
\min \quad & \sum_{j=1}^{n} u_j y_j \\
s.t. \quad & \sum_{j=1}^{k} (u_j + o(j)) y_j \geq \sum_{j=1}^{k} o(j)\Delta_j \quad \forall 1 \leq k \leq n-1 && (5.8) \\
& \sum_{j=1}^{n} (u_j + o(j)) y_j = \sum_{j=1}^{n} o(j)\Delta_j && (5.9) \\
& \sum_{j=k}^{n} (y_j - \Delta_j) \leq 0 \quad \forall k && (5.10) \\
& y_j \geq 0 \quad \forall j \in [n]
\end{aligned}$$

In essence, the LP above encompasses constraints that every optimal allocations must satisfy. To show that LP optimal solutions induce optimal allocations we need to

establish that these constraints are also sufficient. One way to show this would be to prove that all jobs underaged or alternatively, all overaged is a worst case for an allocation induced by some LP optimal solution. Note that while all jobs underaged need not be a worst case for a LP optimal induced allocation, the cost is a lower bound on the worst case cost of an optimal allocation.

As it turns out, in general the constraints are not sufficient and all jobs underaged/overaged need not be worst cases for LP optimal (induced) allocations. However, for the important special case of homogeneous underage costs, $u_i = u$ for every i, we can easily find an LP optimal allocation that is also an optimal allocation. In fact, this is true more generally as long as $u_{j+1} \leq u_j \frac{o(j+1)}{o(j)} \, \forall j$. And in this case an LP optimum is explicitly given by the formula, $x_j = \frac{o(j)\Delta_j}{u+o(j)}$ for every j, which coincides with what was previously known for the case of homogeneous underage costs. We prove this and subsequently discuss other cases under which the LP allocation is optimal.

Lemma 36. *The allocation given by $x_j = \frac{o(j)\Delta_j}{u+o(j)} \, \forall j \in [n]$ is a feasible solution to the LP and has all jobs taking maximum time and all taking minimum time as worst cases.*

Proof. To verify feasibility, observe that constraints (5.8) are satisfied with equality for every k. Also, $x_i \leq \Delta_i$ for every i, so constraints (5.10) are satisfied. Finally, it is easy to see that equality (5.9) is satisfied. This establishes feasibility.

Now, note that the cost of all jobs taking zero time is the same as the cost of all jobs taking maximum time. We prove that these are worst cases by induction. Claim is trivially true for a single job and suppose true for all $n \leq k-1$, then we prove for the case of k jobs. Let x'_i be the allocation given by the formula when we focus on the subset $[k-1]$ of $k-1$ jobs. Note that $x'_i = x_i$ for every $i \in [k-1]$ and by assumption, all jobs underaged and all overaged are worst cases for allocation $\{x'_i\}_{i\in[k-1]}$. Therefore, if there is a worst case for allocation $\{x_i\}_{i\in[k]}$ where job k is underaged and takes zero time, then all jobs taking zero time is a worst case and we're done. Alternatively, if there is a worst case with job k overaged, we have from Lemma 26 that all jobs overaged is a worst case. □

Theorem 37. *If $u_{j+1} \leq u_j \frac{o(j+1)}{o(j)} \forall j$, then the allocation given by $x_j = \frac{o(j)\Delta_j}{u+o(j)} \forall j$ is an optimal allocation.*

Proof. We show that $x_j = \frac{o(j)\Delta_j}{u+o(j)} \forall j$ is an optimal solution to the LP when $u_{j+1} \leq u_j \frac{o(j+1)}{o(j)} \forall j$. The lemma above then gives us that the allocation is also an optimal one. We prove this via induction on the number of jobs n. The claim is clearly true for $n = 1$ and assume the claim holds for $n = i - 1$, we prove for the case with i jobs.

First, observe that as a direct consequence of the conditions $u_{j+1} \leq u_j \frac{o(j+1)}{o(j)} \forall j$, we have that,

$$\frac{u_{j+1}}{u_{j+1} + o(j+1)} \leq \frac{u_j}{u_j + o(j)} \quad \text{for every } j \in [i-1]. \tag{5.11}$$

Now, if constraints (5.8) and (5.10) in the LP were ignored, we have a very simple greedy solution for the LP. Find the index k that minimizes $\frac{u_k}{u_k+o(k)}$, which is implied to be i by inequalities (5.11). Then we set $y_i = \frac{\sum_j o(j)\Delta_j}{u_i/(u_i+o(i))}$ and all other variables to zero to get the optimal solution. In the presence of (5.8), the idea is still to adopt a greedy approach, albeit restricted by constraints (5.8).

Consider an optimal solution to the LP denoted $\{y_j\}_{j\in[i]}$. If $\sum_{j=1}^{k}(u_j + o(j))y_j = \sum_{j=1}^{k} o(j)\Delta_j$ for every $k \in [i]$, we are done since $y_j = x_j$ for every $j \in [i]$. So consider the smallest m, $1 \leq m < i$ such that $\sum_{j=1}^{m}(u_j + o(j))y_j > \sum_{j=1}^{m} o(j)\Delta_j$. Further, suppose y_j is an LP optimum with the maximum m. Therefore, for every $k < m$, $\sum_{j=1}^{k}(u_j + o(j))y_j = \sum_{j=1}^{k} o(j)\Delta_j$. Consider a modified allocation $\{y'_j\}_{j\in[i]}$, with $y'_j = y_j$ for every j except m and $m+1$, $y'_m = \frac{o(m)\Delta_m}{u_m+o(m)} < y_m$ and $y'_{m+1} = y_{m+1} + \frac{u_m+o(m)}{u_{m+1}+o(m+1)}(y_m - y'_m)$. It is easy to see that this is a feasible solution to the LP. More importantly, due to (5.11) we have $\sum_{j\in[i]} u_j y'_j \leq \sum_{j\in[i]} u_j y_j$ and therefore, y'_j is an LP optimum that satisfies (5.8) for indices up to and including m, contradiction. Note that an alternative proof follows from considering the dual of the LP.

□

For the general case, the constraints are still necessary however they are not sufficient. Therefore, an optimal solution to the LP need not correspond to an allocation where all jobs taking zero time or taking maximum time is a worst case. This is

demonstrated by the example we saw earlier in Lemma 33 and in fact, it is easily checked that the additional constraints given by (5.8) do not improve the approximation bound from earlier. Notice that for the example in Lemma 33, the closed form allocation $x_j = \frac{\Delta_j o(j)}{u_j + o(j)}$ is nearly optimal. We consider a new example here, for which the closed form allocation is sub-optimal too,

Example 1: Consider two jobs, jobs 2 and 1 with job 2 scheduled first. Let $\Delta_1 = u_1 = o_1 = 1$ and let $\Delta_2 = 1$, $u_2 = M$ and $o_2 = 3$, where M is a very large number such that $\frac{u_2}{u_2 + o(2)} \to 1$. The allocation induced by LP optimum is given by $x_2 = \frac{1}{\Omega(M)}$ and $x_1 = 2 - x_2$. The worst case is when 2 is overaged and 1 underaged, and the cost in this case is 4. On the other hand, it can be shown that the optimum allocation has cost 3.5 and is given by $x_2^* = \frac{1}{\Omega(M)}$ and $x_1^* = 3/2 - x_2^*$.

In the example above, and the one from Lemma 33, the LP optimum would have given an optimal allocation if $u_1 \geq o_2$. More complicated cases arise for larger number of jobs and it is not clear if one can introduce a polynomial number of linear constraints to the LP in order to ensure that the optimal LP solution is the optimal allocation. Another natural question is whether the allocation $x_j = \frac{\Delta_j o(j)}{u + o(j)} \forall j$ is a good approximation. As demonstrated below, even for two jobs this allocation can be arbitrarily worse than the optimal allocation.

Example 3: Job 1 has $u_1 = \epsilon, \Delta_1 = 1, o_1 = 1$ and $u_2 = M, \Delta_2 = 1, o_2 = \epsilon$. Therefore we have that $\frac{u_1}{u_1 + o_1} \to \epsilon < \frac{u_2}{u_2 + o(2)} \to 1$ for $M \to \infty$ and $\epsilon \to 0$. Optimal solution to the LP is $x_2 = 0, x_1 = (\Delta_1 o(1) + \Delta_2 o(2)) \frac{1}{u_1 + o_1} = \frac{2+\epsilon}{1+\epsilon}$. This is also an optimal allocation (both jobs underaged and both overaged are worst cases, also more generally true when $o_2 = u_1$) and has cost $\epsilon \frac{2+\epsilon}{1+\epsilon}$. Formula gives allocation $x_2' = \Delta_2 o(2) \frac{1}{u_2 + o(2)}$ and $x_1' = \Delta_1 o(1) \frac{1}{u_1 + o(1)} = \frac{1}{1+\epsilon}$. This has cost $M x_2' + \epsilon x_1' = 1 + O(\epsilon)$, which is arbitrarily worse than the optimal cost as $\epsilon \to 0$.

Now we show that if $u_{j+1} > u_j \frac{o(j+1)}{o(j)}$ for some j, there still exists an LP optimal solution that is also an optimal allocation, if some other conditions are satisfied.

Theorem 38. *Given a sequence of n jobs with parameters such that whenever $i \neq m_i := \max\left(\arg\min_{j \leq i} \frac{u_j}{u_j + o(j)}\right)$, we have $u_{m_i} \geq o(i) - o(m_i)$, then the following is both*

an optimal solution to the LP and also an optimal allocation,

$$x_k = \frac{\sum_{i|k=m_i} \Delta_i o(i)}{u_k + o(k)} \quad \forall k.$$

Proof. We follow the same scheme as before, first show that the allocation given by the formula is a feasible solution to the LP and that all jobs underaged/overaged are worst cases. Then prove that this allocation is an optimal solution for the LP.

To check feasibility, first note that equation (5.9) is satisfied by definition since each term $o(i)\Delta_i$ appears exactly once in the sum $\sum_j (o(j) + u_j)x_j$. Constraints (5.8) are also similarly easy to verify. However, here x_i can be larger than Δ_i for some i, so checking (5.10) is less trivial. Indeed, this is where we use the assumption $u_{m_i} \geq o(i) - o(m_i)$ for every i. Note that if for job i, $m_i \neq i$, then $x_i = 0$ and x_{m_i} has a term $\frac{\Delta_i o(i)}{u_{m_i} + o(m_i)} \leq \Delta_i$ in the summation. More generally, we have that for every k either $x_k = 0$ or $x_k \leq \sum_{i \geq k | k = m_i} \Delta_i$. It now follows that constraints (5.10) are satisfied.

To show that all jobs underaged/overaged (both have same cost) is a worst case, we proceed by induction on the number of jobs n. For $n = 1$ this is trivial. Assuming the claim holds for $k - 1$ jobs, we prove the statement for k jobs. Let the allocation given by the formula for $k - 1$ jobs be x'_i for $i \in [k - 1]$ and let x_i be the allocation for k jobs. We will break the proof down into cases. First, observe that $x_i \geq x'_i$ for every job i. Therefore, if for allocation x_i there exists a worst case where job k is underaged, then all jobs underaged is a worst case (because all jobs underaged is a worst case for allocation x'_i by assumption). Otherwise, suppose job k is overaged in all worst cases. Then if $x_k > 0$, we have $x_i = x'_i$ for all other jobs i and using Lemma 26 we have that all jobs overaged is a worst case. Finally, consider the case where k is overaged in all worst cases and $x_k = 0$. Note that $x_i = 0$ for every $i > m_k$ and w.l.o.g., there is a worst case where k takes at least $\frac{\Delta_k o(k)}{u_{m_k} + o(m_k)}$ time. Using Lemma 26 again, we have that all jobs overaged is a worst case.

Now, to show that the given allocation is an optimal solution to the LP, consider

the dual,

$$\begin{aligned} \max \quad & \sum_{i=1}^{n}(\sum_{j=1}^{i} o(j)\Delta_j)p_i + \sum_{i=1}^{n}(\sum_{j=i}^{n}\Delta_j)r_i \\ s.t. \quad & (u_j + o(j))\sum_{i\geq j} p_i + \sum_{i\leq j} r_i + s_j \leq u_j \quad \forall j \in [n] \\ & p_i \geq 0 \quad \forall i \in [n-1]; \quad -r_i, s_i \geq 0 \quad \forall i \in [n]. \end{aligned} \tag{5.12}$$

Consider the solution $p_n = \frac{u_{m_n}}{u_{m_n}+o(m_n)}$ and $p_i = \frac{u_{m_i}}{u_{m_i}+o(m_i)} - \frac{u_{m_{i+1}}}{u_{m_{i+1}}+o(m_{i+1})}$ for every $i < n$ such that $m_i \neq m_{i+1}$. Set all other variables to zero and note that when $m_i \neq m_{i+1}$ we have, $m_{i+1} = i+1$ and further, $\frac{u_{m_i}}{u_{m_i}+o(m_i)} - \frac{u_{m_{i+1}}}{u_{m_{i+1}}+o(m_{i+1})} > 0$. To check feasibility, it only remains to consider constraints (5.12). So fix arbitrary $k \in [n]$, and we have, $(u_k + o(k))\sum_{i\geq k} p_i = (u_k + o(k))\frac{u_{m_k}}{u_{m_k}+o(m_k)} \leq u_k$, by definition of m_k. Finally, since the objective value of this dual solution is designed to match the objective value of allocation $\{x_k\}_{k\in[n]}$ for the primal, we have an optimal solution to the primal.

□

5.4 Flexible Job Order

5.4.1 Homogeneous Underage Costs

So far we assumed that jobs were given to us in a fixed order and we had to allocate appointments robust to uncertainty in service times. Now we consider appointment scheduling with flexible job order i.e., we must decide the order as well as the appointment allocation for jobs. Let us index jobs from $\{1, \ldots, n\}$ as before, but now let $\pi : [n] \to [n]$ denote a permutation such that $\pi(i)$ is the $(n - \pi(i) + 1)^{th}$ appointment scheduled. So now job n is not necessarily the first scheduled job but rather the $(n - \pi(n) + 1)^{th}$ scheduled job. Our primary focus here will be on the case of homogeneous underage costs. For this case, it was previously shown in [MSS14] that the problem is an instance of single machine min-sum scheduling with a concave objective over completion times. The latter problem has an established and ongoing line of work, which we discuss more subsequently. Here we will leverage this connection

further and use additional properties of our objective to show simple near optimal heuristics. Consider now the following formulation for our optimization problem, which is similar to [MSS14] (with difference in notation),

$$\min_{\pi:[n]\leftrightarrow[n]} \sum_{j=1}^{n} \frac{\Delta_j(\sum_{\pi(k)\leq\pi(j)} o_k)u}{\sum_{\pi(k)\leq\pi(j)} o_k + u}.$$

Note that we can normalize the objective and overage costs so that w.l.o.g., $u = 1$. Now consider a job scheduling problem with n jobs indexed by $i \in [n]$, weight of job i is $w_i = \Delta_i$, processing time $p_i = o_i$ and the objective is,

$$\min_{\pi} \sum_{j=1}^{n} w_j f(C_j).$$

Where $f(x) = \frac{x}{x+1}$ is a concave function of the completion time of job j, $C_j = \sum_{\pi(k)\leq\pi(j)} p_k$. Observe that these problems are identical but with reversed ordering i.e., given an order, the first job scheduled in the concave scheduling problem is the last appointment scheduled and so on.

The problem of scheduling with non-linear and in particular, concave objective has a well established line of work [HJ15, SW, MV13, CMSV16, ELMS$^+$10]. There is a scaling based EPTAS for the concave objective problem due to [SW], and therefore also for the appointment scheduling problem. However, the algorithm is fairly non-trivial to understand and not easy to implement in practice. Interestingly, it is still not known whether this problem is computationally hard or polynomial time solvable and this has remained an intriguing open question in the scheduling literature. Since we are interested in simple heuristics, our focus here will be on deriving customized but simple algorithms that achieve a good approximation for our problem. There has been some work on simple approximation algorithms for general concave scheduling as well [HJ15, SW]. In particular, [SW] first showed that Smith's rule[Smi56], which schedules jobs in descending order of ratio w_i/p_i, is a $(\sqrt{3}+1)/2 \gtrapprox 1.367$ approximation for concave scheduling.

For our more specific case where $f(x) = \frac{x}{x+1}$, [MSS14] gives a customized $2+\epsilon$

approximation. Further, [HJ15] gives a formula for evaluating the (tight) approximation ratio of Smith's rule for any specific concave function. Using this formula we find that Smith's rule is optimistically ≈ 1.1356 approximate when $f(x) = \frac{x}{x+1}$. While this gives us a very simple and intuitive heuristic with a good approximation for our problem, it is actually suboptimal even for the case of two jobs. This motivates us to consider a new heuristic, where we order jobs in descending order of $\frac{w_i}{p_i(p_i+1)}$. We call this Customized-Smith's or C-Smith for short. This straightforward heuristic is optimal for the case of two jobs and furthermore has an improved guarantee of $1.06036 < \beta < 1.06043$, within roughly 6% of the optimal. Perhaps another natural expectation from an ordering would be to demand that it is optimal w.r.t. exchanging the order of any two consecutive jobs. Let us call such an order *locally optimal*. The schedule output by both Smith and C-Smith is not locally optimal in general i.e., there could be two adjacent jobs in the output schedule such that swapping their order results in a strictly better schedule. This motivates us to consider a natural locally optimal algorithm. We show that this algorithm always outputs a solution that is at least as good as C-Smith, implying an upper bound of β on the guarantee of the algorithm, however we leave finding the exact guarantee of the algorithm as an open problem.

Algorithm 12 C-Smith

1: Schedule jobs in increasing order of $\frac{\Delta_i}{o_i(o_i+u)}$.

Remark: The ratio $\frac{w_i}{p_i(p_i+1)}$ translates to $\frac{\Delta_i}{o_i(o_i+u)}$ for the appointment scheduling and ordering problem. Also, while for concave scheduling, we order jobs in decreasing order of the ratio. For appointments, we want the reverse order and therefore schedule jobs in increasing order of the ratio $\frac{\Delta_i}{o_i(o_i+u)}$. For the rest of the discussion, we focus on the concave scheduling setup and therefore C-Smith stands for ordering jobs in decreasing order of $\frac{w_i}{p_i(p_i+1)} = \frac{\Delta_i}{o_i(o_i+1)}$. We use w_i, Δ_i and p_i, o_i interchangeably.

Before we delve into approximation bounds, first consider some special cases to understand how the parameters affect order. For the case with homogeneous $\Delta_i = \Delta$, the optimum schedule orders jobs in ascending order of o_i. For the case with

homogeneous $o_i = o$, we order the jobs in descending order of Δ_i values to get the optimal schedule. Therefore, Smith's rule is clearly optimal for both special cases. So under what circumstances is it suboptimal? To understand this we now consider two consecutive jobs i and j in an arbitrary schedule, and determine the conditions under which the current order is better than the order which swaps the two jobs.

Lemma 39. *Given an arbitrary schedule π, with two jobs i, j scheduled consecutively, $\pi(j) = \pi(i) + 1$. Let $o_x = \sum_{\pi(k)<\pi(i)} o_k$. Then the order is optimal w.r.t. swapping jobs i, j iff*

$$\frac{\Delta_i}{o_i(o_i + o_x + 1)} \geq \frac{\Delta_j}{o_j(o_j + o_x + 1)}.$$

Proof. Consider the schedule π' which swaps i and j. Let C_i, C'_i denote the completion time of job i in π and π' respectively. Similarly, C_j, C'_j for job j and note that the completion times of all jobs except i, j are unchanged. The difference in costs between π' and π is given by,

$$\begin{aligned}
\sum_k \Delta_k(f(C'_k) - f(C_k)) = \quad & \frac{\Delta_j(o_j + o_x)}{o_j + o_x + 1} + \frac{\Delta_i(o_i + o_j + o_x)}{o_i + o_j + o_x + 1} \\
& - \frac{\Delta_j(o_j + o_i + o_x)}{o_j + o_i + o_x + 1} + \frac{\Delta_i(o_i + o_x)}{o_i + o_x + 1} \\
= \quad & \Delta_i \frac{o_j}{(o_j + o_i + o_x + 1)(o_i + o_x + 1)} \\
& - \Delta_j \frac{o_i}{(o_j + o_i + o_x + 1)(o_j + o_x + 1)} \\
= \quad & \frac{o_i o_j}{o_i + o_j + o_x + 1}\Big(\frac{\Delta_i}{o_i(o_i + o_x + 1)} - \frac{\Delta_j}{o_j(o_j + o_x + 1)}\Big)
\end{aligned}$$

Therefore, $\sum_k \Delta_k(f(C'_k) - f(C_k)) \geq 0$ iff $\frac{\Delta_i}{o_i(o_i+o_x+1)} \geq \frac{\Delta_j}{o_j(o_j+o_x+1)}$. □

Observe that when we just have two jobs, $o_x = 0$ and the optimal order is i followed by j iff $\frac{\Delta_i}{o_i(o_i+1)} \geq \frac{\Delta_j}{o_j(o_j+1)}$. This naturally implies that Smith's rule is not optimal even for two jobs. Also note that if $o_x \to \infty$, then we have i before j iff $\frac{\Delta_i}{o_i} \geq \frac{\Delta_j}{o_j}$, indicating that Smith's rule can become optimal for ordering jobs later in the schedule (provided the overage costs are not too small). Next, we evaluate the refined approximation ratio for Smith's rule for our specific concave objective $\frac{x}{1+x}$ using results in [HJ15].

Consider q/ϵ jobs each of length and weight ϵ and thus Smith ratio 1. As ϵ tends to zero, we get a *line* job of length q, as defined in [HJ15]. Note that the both the Smith and C-Smith ratio of every infinitesimal job that makes up the line job tends to 1 as $\epsilon \to 0$.

Lemma 40 (Theorem 3.2 in [HJ15])**.** *For the concave scheduling problem, a worst case of Smith's rule occurs for the case of a line job of length and weight q and a normal job of length and weight p. Smith's rule schedules the regular job followed by the line job and the optimal order is the reverse. Therefore, the ratio is,*

$$\max_{p,q\geq 0} \frac{p\frac{p}{p+1} + q - \ln(\frac{q}{p+1}+1)}{p\frac{p+q}{p+q+1} + q - \ln(q+1)} \geq 1.1356.$$

Proof. Using a local optimization subroutine from MATLAB (fmincon), we find a local minimum with value 1.135680634960286 at $q = 1.257523511039715, p = 1.8614133406450792$. □

Using Lemma 39 it is easy to construct instances where both C-Smith and Smith's rule lead to orderings that are not locally optimal. This prompts the following locally optimal algorithm,

Algorithm 13 Locally Optimal Algorithm

1: **Initialize:** $S = \emptyset, o(S) = 0$

2: **for** $i = 1$ to n **do**

3: Breaking ties arbitrarily,

$$j = \underset{k\in[n]\setminus S}{\operatorname{argmax}} \frac{\Delta_k}{o_k(o_k + o(S) + 1)}$$

4: $\pi(j) = i, S = S \cup \{j\}, o(S) = o(S) + o_j$

5: **Output:** $\pi(.)$

Lemma 41. *Algorithm 13 is locally optimal.*

Proof. Consider arbitrary jobs j and $j+1$ in the order output by the algorithm. Let o_{S_j} represent the value of $o(S)$ at iteration j (where job j was picked out). Since we

have $\frac{\Delta_j}{o_j(o_j+o_{S_j}+1)} \geq \frac{\Delta_{j+1}}{o_{j+1}(o_{j+1}+o_{S_j}+1)}$, Lemma 39 implies that the current order is swap optimal w.r.t. pairs $j, j+1$. □

Yet, a local optimum need not be globally optimal, indeed the algorithm can be suboptimal as demonstrated by the following example.

Sub-optimality of Algorithm 13: Consider 3 jobs, with $\Delta_1 = o_1 = 1$, $\Delta_2 = 15, \Delta_3 = 14$ and $o_2 = o_3 = 5$. Then the order 1,2,3 is locally optimal since $\frac{\Delta_1}{o_1(o_1+1)} = \frac{\Delta_2}{o_2(o_2+1)} = \frac{1}{2}$ and $\frac{\Delta_3}{o_3(o_3+1)} < \frac{1}{2}$. Similarly, $\frac{\Delta_2}{o_2(o_2+o_1+1)} > \frac{\Delta_3}{o_3(o_3+o_1+1)}$. However, notice that one can swap 1 and 2 without changing the value of the solution, resulting in an equally costly order 2,1,3. Now this is not locally optimal and one gets a strictly better order by swapping 1,3 to get the order 2,3,1.

We will next show that the schedule output by the algorithm is always at least as good as the schedule given by C-Smith. We will then show an approximation guarantee of $1.06036 < \beta < 1.06043$ for C-Smith, also implying that the local algorithm is β approximate. But first, we need to understand how the optimal ordering of a pair of jobs changes due to other preceding jobs in a given schedule.

Lemma 42. *If* $\frac{\Delta_i}{o_i(o_i+1)} \geq \frac{\Delta_j}{o_j(o_j+1)}$ *but* $\frac{\Delta_j}{o_j(o_j+x+1)} \geq \frac{\Delta_i}{o_i(o_i+x+1)}$ *for some* $x > 0$*, then* $o_j \geq o_i$ *and further* $\frac{\Delta_j}{o_j(o_j+x'+1)} \geq \frac{\Delta_i}{o_i(o_i+x'+1)}$ *for every* $x' \geq x$*.*

Proof. Suppose $o_i > o_j$. We have for $x > 0$,

$$\begin{aligned}
\frac{\Delta_i}{o_i(o_i+1)} &\geq \frac{\Delta_j}{o_j(o_j+1)} \\
\frac{\Delta_i}{o_i(o_i+x+1)}\frac{o_i+x+1}{o_i+1} &\geq \frac{\Delta_j}{o_j(o_j+x+1)}\frac{o_j+x+1}{o_j+1} \\
\frac{\Delta_i}{o_i(o_i+x+1)}\Big(1+\frac{x}{o_i+1}\Big) &\geq \frac{\Delta_j}{o_j(o_j+x+1)}\Big(1+\frac{x}{o_j+1}\Big) \\
\implies \frac{\Delta_i}{o_i(o_i+x+1)} &> \frac{\Delta_j}{o_j(o_j+x+1)} \quad \text{(Contradiction).}
\end{aligned}$$

Therefore, $o_j \geq o_i$ as desired. Now starting with $\frac{\Delta_j}{o_j(o_j+x+1)} \geq \frac{\Delta_i}{o_i(o_i+x+1)}$ and repeating the steps above with the assumption $o_j \geq o_i$, we have that $\frac{\Delta_j}{o_j(o_j+x'+1)} \geq \frac{\Delta_i}{o_i(o_i+x'+1)}$ for every $x' \geq x$. □

Corollary 43. *Suppose* $\frac{\Delta_i}{o_i(o_i+1)} = \frac{\Delta_j}{o_j(o_j+1)}$. *If* $o_j \geq o_i$, *then* $\frac{\Delta_j}{o_j(o_j+x+1)} \geq \frac{\Delta_i}{o_i(o_i+x+1)}$ *for every* $x \geq 0$. *Conversely, if* $\frac{\Delta_j}{o_j(o_j+x+1)} \geq \frac{\Delta_i}{o_i(o_i+x+1)}$ *for some* $x > 0$ *then* $o_j \geq o_i$.

Theorem 44. *The cost of ordering by Algorithm 13 is always at most as much as the cost of ordering by C-Smith.*

Proof. Let π_1 be the order from C-Smith and π_2 the order from Algorithm 13. Further suppose that the orders coincide for the first $i \geq 0$ jobs and $i+1$ is the smallest index such that $\pi_1^{-1}(i+1) \neq \pi_2^{-1}(i+1)$. Let $k = \pi_2^{-1}(i+1)$ and clearly $\pi_1(k) > i+1$. We will show that the order which schedules k at $i+1$ but otherwise follows the C-Smith order π_1, is at least as good as π_1 and more generally, we demonstrate a way to transform π_1 to π_2 while never increasing the cost.

We have $\frac{\Delta_j}{o_j(o_j+1)} \geq \frac{\Delta_k}{o_k(o_k+1)}$ for every j such that $\pi_1(j) \leq \pi_1(k)$ (due to C-Cmith). Further, let $o(i) = \sum_{\pi_1(j) \leq i} o_j$ then, $\frac{\Delta_k}{o_k(o_k+o(i)+1)} \geq \frac{\Delta_j}{o_j(o_j+o(i)+1)}$ for every j such that $i < \pi_1(j) \leq \pi_1(k)$ (due to Algorithm 13). It follows that if $o(i) = 0$, then $\frac{\Delta_k}{o_k(o_k+1)} = \frac{\Delta_j}{o_j(o_j+1)}$ for all j scheduled between $i+1$ to $\pi_1(k)$ in π_1, and we are done. So assume $o(i) > 0$. Then from Lemma 42 we have, $o_k \geq o_j$ and $\frac{\Delta_k}{o_k(o_k+\sum_{\pi_1(t)<\pi_1(j)} o_t+1)} \geq \frac{\Delta_j}{o_j(o_j+\sum_{\pi_1(t)<\pi_1(j)} o_t+1)}$ for every j such that $i < \pi_1(j) \leq \pi_1(k)$. Therefore, when $j = \pi_1^{-1}(\pi_1(k)-1)$, we have that swapping j and k in π_1 does not result in a worse schedule, and subsequently for every j such that $i < \pi_1(j) \leq \pi_1(k)-1$.

Now consider the new schedule π_1', where $\pi_1'(k) = i+1$ and order is otherwise same as π_1. The cost of π_1' is at most the cost of π_1 and the first index where π_1' and π_2 differ is $i+2$. We can now repeat the same process to get a schedule at least as good as π_1' that coincides with π_2 for an even larger number of jobs. Repeating this enough times gives us the desired. □

Next, we focus on showing a nearly tight approximation bound for C-Smith. More formally, we show that,

Theorem 45. *The worst case ratio of C-Smith is given by* β *where* $1.06036 < \beta < 1.06043$.

To prove the above theorem, we will establish several intermediate results that characterize and simplify the worst case instance for C-Smith. First, we generalize

Lemma 3.5 in [HJ15] (Lemma 21 in [SW]) and show that there is a worst case instance for C-Smith where all jobs are tied and in fact have ratio 1. Then, we show that in the worst case C-Smith orders jobs in ascending order of o_i and the optimal order is the exact reverse. Interestingly, this is the opposite of Lemma 3.6 in [HJ15] (Proposition 20 in [SW]) for Smith's rule, where the optimal order is ascending in o_i and worst order is descending. Thereafter, we can cleanly write down a non-convex optimization problem in infinitely many variables. The optimal value of this problem is the approximation ratio and every optimal solution is a worst case instance.

To then get lower and upper bounds on the optimum value, we utilize properties specific to our objective $f(x) = \frac{x}{x+1}$ to approximate the problem (which has infinitely many variables) with a family of optimization problems, that while still non-convex, have a finite number of variables. As we consider more complex objectives from the family, we get tighter upper bounds on true approximation ratio but the number of variables involved increases. We find the global optimum to a problem in the family with 5 variables, and this closely matches our lower bound. We solve such a non-convex problem to global optimality by establishing upper and lower bounds on variables and using linear cuts, both of which allow us to then effectively use a nonlinear globally optimal MINLP solver, Couenne [IU06].

Now, consider heuristics that order jobs using a simple ratio. Where we use the word simple to imply that the ratio for each job is evaluated only using the job parameters (o_i, Δ_i, u_i), and is independent of other jobs. We call such a heuristic *weight proportional* if for every job i, the ratio is linearly proportional to its weight $w_i (= \Delta_i)$.

Lemma 46. *Given a weight proportional ratio based heuristic, any worst case instance I can be modified into another worst case instance I', where all jobs have the same ratio and are thus tied. Further we can also assume that jobs have ratio 1.*

Proof. Proof simply generalizes Lemma 3.5 in [HJ15] (and Lemma 21 in [SW]). Consider a ratio based heuristic and suppose there exists k classes of jobs based on the ratio. All jobs within a class have the same ratio. Order the classes in decreasing

order of the representative ratio and note that the algorithm schedules jobs in class j before jobs in class $j+1$ for every $1 \leq j \leq k-1$. Let $A(j, I)$ denote the total cost of jobs in class j, given the ratio based order for instance I. Similarly, $OPT(j, I)$ denotes the total cost over jobs in the same class j in the optimal order (in case of multiple optima, fix one for this proof). We use the shorthand $A(I), OPT(I)$ to denote the total costs over all jobs. Let $A(I) \geq \alpha OPT(I)$ and let i be the class with the smallest index such that, $A(i, I) \geq \alpha OPT(i, I)$. Note that there is such an i, since otherwise $\sum_{j=1}^{k} A(j, I) = A(I) < \alpha OPT(I)$. Now let I be a worst case instance with the smallest number, d, of distinct classes. We show that $d = 1$ by considering two cases and deriving a contradiction in each.

First, if $i = 1$, then we can simply consider a new instance I' that consists of only jobs in class 1. We have that $OPT(I') \leq OPT(1, I)$ and $A(I') = A(1, I)$. Therefore, $A(I') \geq \alpha OPT(I')$, contradiction.

Otherwise, assume $i > 1$. Since the ordering heuristic is weight proportional, we can multiply the weights of all jobs in class i by a factor $\delta > 1$ such that the ratio of class i jobs now equals the ratio of class $i-1$. Call this instance I_1. We have $A(i, I_1) = \delta A(i, I)$ and therefore $A(I_1) = A(I) + (\delta - 1)A(i, I)$. Similarly, $OPT(I_1) \leq OPT(I) + (\delta - 1)OPT(i, I)$. Further, since $A(i, I) \geq \alpha OPT(i, I)$, we get $A(I_1) \geq A(I) + (\delta - 1)\alpha OPT(i, I)$. Finally, $A(I_1) \geq \alpha(OPT(I) + (\delta - 1)OPT(i, I)) \geq \alpha OPT(I_1)$. Therefore I_1 is also a worst case instance, but with $d - 1$ distinct classes, contradiction.

Given an instance where all jobs have the same ratio, we can easily multiply all jobs weights by a scalar to normalize the ratio to one. □

Remark: One might ask at this point if tie-breaking rules make a difference to the performance guarantee. Since we can always perturb weights adversarially by tiny amounts so that there are no ties but the costs are essentially unchanged, tie-breaking doesn't make a difference in theoretical performance.

Lemma 47. *In any worst case instance for C-Smith with all jobs tied, the optimal order is to schedule jobs in descending order of p_i and the worst case is attained when*

jobs are scheduled in ascending order of p_i.

Proof. Consider two consecutive jobs, k and $k+1$ in an optimal schedule. By applying Corollary 43, we have that $p_k = o_k \geq p_{k+1} = o_{k+1}$. This more generally implies jobs must be ordered in descending order of p_k in the optimal schedule.

Now consider the worst order and observe that if there is pair of consecutive jobs $k, k+1$ such that $p_k > p_{k+1}$, then using Corollary 43 again we have that swapping k and $k+1$ would be strictly worse. Therefore, in the worst schedule jobs are ordered in ascending order of p_i. □

So far, our simplification implies that we can assume all jobs have C-Smith ratio of 1, and given this we also know the worst possible (C-Smith) order and the optimal order. So consider regular jobs 1 to n (where n can be arbitrarily large), with processing times $p_1 \geq p_2 \geq \cdots \geq p_n$ and a line job of length q. We use the notion of a line job as defined in [HJ15]; a line job of length q consists of infinitely many infinitesimally small jobs, each takes time ϵ so the total number of jobs is q/ϵ and $\epsilon \to 0$. In [HJ15], the weight of each small job was ϵ (so that the Smith's ratio was 1) and therefore, the total weight of a line job was q, same as the time taken. Similarly, we let the weight of each small job be $\epsilon(\epsilon+1)$ so that its C-Smith ratio is 1. However, the ϵ^2 term in the weight disappears upon integration and the cumulative weight of a line job of length q in our setting is still q. Now, the cost incurred due to a line job that runs from time t to $t+p$ is given by,

$$F(t, t+p) = \int_t^{t+p} f(x)dx = \int_t^{t+p} \Big(1 - \frac{1}{x+1}\Big)dx = \quad p - \ln\Big(\frac{p}{t+1} + 1\Big).$$

Observe that $F(t, t+p) + F(t+p, t+p+q) = F(t, t+p+q)$ for every $t, p, q > 0$, and this telescoping property of $F(.)$ will be useful later. Further, since $f(.)$ is monotonically increasing we have,

$$p\frac{t}{t+1} \leq F(t, t+p) \leq p\frac{t+p}{t+p+1}. \tag{5.13}$$

Let $p(i) = \sum_{k \leq i} p_k$ and $q(i) = q + \sum_{k \geq i} p_k$ with $p(0) = 0$ and $q(n+1) = q$. Note

that weight w_i of job i is given by $p_i(p_i+1)$. Now, the optimal order schedules jobs 1 through n in that order and ends with the line job. The worst case schedule is the exact reverse, beginning with the line job, followed by job n and ending with job 1. Note that $p_1 > 0$ since otherwise the optimal and C-Smith order are identical for all values of q and we have a ratio of 1. Given these definitions, the worst case ratio $W(n)$, for instances with n regular jobs (plus a line job) all with C-Smith ratio unity, can be written as,

$$N(q, p_1, \ldots, p_n) = F(0, q) + \sum_{i=1}^{n} p_i(p_i+1)\frac{q(i)}{q(i)+1} \tag{5.14}$$

$$D(q, p_1, \ldots, p_n) = F(p(n), p(n)+q) + \sum_{i=1}^{n} p_i(p_i+1)\frac{p(i)}{p(i)+1} \tag{5.15}$$

$$W(n) = \max_{q, p_1, \ldots, p_n \geq 0} \frac{N}{D}(q, p_1, \ldots, p_n).$$

Clearly $W(n) \leq W(n+1)$ for all n. However, recall from Lemma 40, that for Smith's rule there exists a worst case which includes just a regular job and a line job. So if we use $S(n)$ to denote the worst case ratio for Smith's rule with n regular jobs, [HJ15, SW] showed that $S(n) = S(1)$ for all $n \geq 1$. The worst case for C-Smith is more involved and in particular, it is not clear if there is a worst case with a small number of regular jobs. In fact, we show that $W(1) < W(2) < W(3) < W(4)$, implying that every worst case has at least 4 jobs. Nonetheless, we have $1.06036 < W(4)$ and $W(n) < 1.06043$ for every $n \geq 2$, effectively showing $W(4)$ is reasonably close to the true approximation guarantee. To establish the upper bound on $W(n)$, we introduce a family of non-linear optimization problems where we can control the number of variables.

In order to construct this family, we establish and exploit constraints on the *worst case* value of variables $q, p_1, \ldots, p_n$ i.e., values that attain objective value $W(n)$. Further, the same bounds and constraints also greatly help in using Couenne to actually solve instances of the non-convex problems in reasonable time. To give a quick overview of the rest of the analysis: First, we show a lower bound on p_1 that allows us to find $W(1)$. Without the lower bound the solver does not converge due

to numerical issues related to dividing two very small quantities as $p_1, q \to 0$. Then, we show that $W(2) > W(1)$ by finding a lower bound on $W(2)$ that is larger than $W(1)$. This implies that there are at least two regular jobs in every worst case. Given this observation we then establish bounds and constraints on other variables. Finally, we combine these conditions to establish our family of upper bounds, which we then solve numerically.

Lemma 48. *Let β be the approximation ratio for C-Smith, then for every n we have that in the worst case, $p_1 \geq \beta - 1$.*

Proof. Let us assume $p_1 < \beta - 1$ and derive a contradiction. Considering the numerator (5.14) for n jobs we have,

$$q - \ln(q+1) + \sum_{i=1}^{n} p_i(p_i+1)\frac{q(i)}{q(i)+1} < \beta\Big(q - \ln(q+1) + \sum_{i=1}^{n} p_i \frac{q(i)}{q(i)+1}\Big) = \beta N',$$

where the inequality follows from $p_i + 1 \leq p_1 + 1 < \beta$. On the other hand, the denominator (5.15),

$$q - \ln\Big(\frac{q}{p(n)+1}+1\Big) + \sum_{i=1}^{n} p_i(p_i+1)\frac{p(i)}{p(i)+1} \geq q - \ln\Big(\frac{q}{p(n)+1}+1\Big) + \sum_{i=1}^{n} p_i \frac{p(i)}{p(i)+1} = D'.$$

The key now is to observe that $\frac{N'}{D'}$ is the ratio of the optimal order divided by the worst order when all jobs have Smith's ratio 1 (thus job i takes time p_i and has weight p_i), and therefore $\frac{N'}{D'} \leq 1$. Now, we have a contradiction since $\frac{N}{D} < \beta\frac{N'}{D'} \leq \beta$. □

Now that we have a lower bound on p_1, we can evaluate $W(1)$ using Couenne to find,

$$W(1) = \max_{p \geq \beta - 1,\, q \geq 0} \frac{q - \ln(q+1) + p(p+1)\frac{p+q}{p+q+1}}{q - \ln(\frac{q}{p+1}+1) + p^2} = 1.0567 \text{ when } p = 2.8943, q = 8.0775^{1}.$$

Further, we also have $W(2) > 1.603 > W(1)$ for $q = 7.7309, p_1 = 2.3001, p_2 = 1.8336$. Therefore, every worst case has at least two regular jobs. Given this, we will now

[1] 1.056749812107016 for $q = 8.077507951542048$ and $p = 2.8943201615651377$.

establish more constraints on the variables but first, consider the following simple and useful observation.

Lemma 49. *Given $\frac{N}{D} = \beta \geq 1$, for $n, d \geq 0$, we have the following:*

1. *If $n \geq \beta d$ then $\frac{N+n}{D+d} \geq \beta$.*

2. *If $n \leq \beta d$ and $D - d > 0$ then $\frac{N-n}{D-d} \geq \beta$.*

In both cases, equality iff $n = \beta d$.

Proof. 1. follows from $\frac{N+n}{D+d} \geq \frac{N+\beta d}{D+d} = \beta$ and *2.* from $\frac{N-n}{D-d} \geq \frac{N-\beta d}{D-d} = \beta$. □

We now show a lower bound on q, which is key to proving several subsequent results.

Lemma 50. *For $n \geq 2$, in a worst case with $p_n > 0$ we have, $q > p(n-1)$.*

Proof. Consider the order given by C-Smith and the optimal order and suppose we turn job n (for $n \geq 2$) into a line job with length and weight p_n, keeping everything else as is in both orders. The change in cost (new cost - old cost) for the C-Smith schedule is,

$$\delta_N = -p_n(p_n + 1)\frac{q(n)}{q(n) + 1} + F(q(n+1), q(n)). \tag{5.16}$$

Similarly, the change in cost of the optimal order is,

$$\delta_D = -p_n(p_n + 1)\frac{p(n)}{p(n) + 1} + F(p(n-1), p(n)). \tag{5.17}$$

Using (5.13) it is easy to see that both δ_N and δ_D are negative. So if $-\delta_N \leq -\delta_D$, then we have a strictly worse instance due to Lemma 49 *(2.)*.

We claim $-\delta_N \leq -\delta_D$ as long as $q(n) \leq p(n)$ or $q \leq p(n-1)$. Therefore, we can assume that $q > p(n-1)$ in the worst case. To see the claim, treat δ_N as a function of q i.e., $\delta_N(q)$. Clearly, $\delta_N(p(n-1)) = \delta_D$. Moreover, $\frac{d\delta_N}{dq} = -p_n(p_n+1)\frac{1}{(q+p_n+1)^2} + \frac{p_n}{(q+1)(q+p_n+1)} = \frac{-qp_n^2}{(q+1)(q+p_n+1)^2} \leq 0$, therefore $\delta_N(q) \geq \delta_D$ for $0 \leq q \leq p(n-1)$ and $\delta_N(q) \leq D$ for $q > p(n-1)$. □

The following is a useful property of F, which combined with the above lower bound on q will help us establish a lemma that is key to our upper bounds on $W(n)$.

Lemma 51. *Given $t_2, t_1, p \geq 0$, we have $p\frac{t_1+p}{t_1+p+1} - F(t_1, t_1+p) > p\frac{t_2+p}{t_2+p+1} - F(t_2, t_2+p)$ iff $t_2 > t_1$.*

Proof. Observe that $p\frac{t+p}{t+p+1} - F(t, t+p) = \int_t^{t+p} (\frac{t+p}{t+p+1} - \frac{t+x}{t+x+1})dx \geq 0$. Since f is strictly concave, we have $t_2 > t_1$ iff $\frac{t_1+p}{t_1+p+1} - \frac{t_1+x}{t_1+x+1} > \frac{t_2+p}{t_2+p+1} - \frac{t_2+x}{t_2+x+1}$ for every $x \geq 0$. □

Lemma 52. *Consider the ratio $r = \frac{\lambda p_n \frac{q+p_n}{q+p_n+1} + P}{\lambda p_n \frac{p(n)}{p(n)+1} + Q}$. Given $r, \lambda \geq 1$, $P, Q, q, p_n \geq 0$, and $q + p_n > p(n) \geq p_n$ we have,*

$$\frac{\lambda F(q, q+p_n) + P}{\lambda F(p(n) - p_n, p(n)) + Q} > r.$$

Proof. Let $p(n-1) = p(n) - p_n \geq 0$. The claim is trivial when $p_n = 0$ so let us assume $p_n > 0$. Suppose we replace the term $p_n \frac{q+p_n}{q+p_n+1}$ in the numerator by $F(q, q+p_n)$ and replace $p_n \frac{p(n)}{p(n)+1}$ in the denominator by $F(p(n-1), p(n)))$. Then, the numerator decreases (old cost - new cost) by $\delta_N = \lambda(p_n \frac{q+p_n}{q+p_n+1} - F(q, q+p_n))$. The denominator decreases by $\delta_D = \lambda(\frac{p(n)}{p(n)+1} - F(p(n-1), p(n)))$. Clearly $\delta_N, \delta_D \geq 0$ and since $q > p(n-1)$, using Lemma 51 we have, $\delta_D > \delta_N$. Since the original ratio is at least 1, we have a strictly larger ratio after this operation due to Lemma 49 *(2.)*. □

Using the above lemma, we now show an upper bound on the total time taken by regular jobs, $p(n)$.

Lemma 53. *Given a worst case solution with $n \geq 2$ regular jobs ($p_n > 0$) and a line job q, we have $\frac{(p_n+q)(p(n)+1)}{(p_n+q+1)p(n)} \geq \beta$.*

Proof. Consider the numerator N and denominator D from (5.14),(5.15). We assume that $\frac{\frac{p_n+q}{p_n+q+1}}{\frac{p(n)}{p(n)+1}} < \beta$ and derive a contradiction. Using Lemma 49 *(2.)* we have that replacing $p_n(p_n+1)\frac{q(n)}{q(n)+1}$ in N and $p_n(p_n+1)\frac{p(n)}{p(n)+1}$ in D by $p_n \frac{q(n)}{q(n)+1}$ and $p_n \frac{p(n)}{p(n)+1}$ respectively, increases the objective. And finally, using Lemma 52 (with $\lambda = 1$) we have that turning job n into a line job of length p_n increases the ratio, giving us

a contradiction since the instance with line job of length $q + p_n$ and regular jobs $1, \ldots, n-1$ as before, cannot be strictly worse. Note that we satisfy all conditions necessary to use Lemma 52 since the ratio is at least 1, $q > p(n-1)$ using Lemma 50 and other terms in the numerator and denominator are unaffected when converting n into a line job. □

Corollary 54. *Given a worst case solution with $n \geq 2$ regular jobs ($p_n > 0$), $p(n) \leq \frac{1}{\beta-1}$.*

Proof. $\frac{(p_n+q)(p(n)+1)}{(p_n+q+1)p(n)} \leq \frac{p(n)+1}{p(n)}$ and then a direct application of Lemma 53 to the LHS. □

Suppose now that for the case of n regular jobs there exists a worst case with $p_n > 0$, then combining several of the results above we now have,

$$
\begin{aligned}
& W(n) = \max_{q,p_1,\ldots,p_n} \frac{N}{D}(q, p_1, \ldots, p_n) && (5.18)\\
\text{s.t.} \quad & q \geq p(n-1) \\
& p_i \geq p_{i+1} \quad \forall 1 \leq i \leq n-1 \\
& q, p_1 \geq \beta - 1 \\
& p(n) \leq \frac{1}{\beta - 1} \\
& q \geq 0; \quad p_i \geq 0 \forall i
\end{aligned}
$$

The true approximation factor is given by $W = \lim_{n\to\infty} W(n)$. The limit exists since the sequence $W(n)$ is non-decreasing. However, solving (5.18) is non-trivial since the objective is provably non-convex. In fact, it can be shown that optimal solutions to $W(n-1)$ are saddle points for $W(n)$. Furthermore, we find that $W(4) > W(3) > W(2) > W(1)$ and the program above is already intractable to solve for $W(5)$ and beyond using Couenne (solver does not converge within two weeks on a computing cluster). So while we have lower bounds on the true guarantee W (using $W(4)$), in order to find upper bounds we need to account for the possibly large number of regular jobs in a worst case instance.

Lemma 55. *For $n \geq 2$, suppose that there is a worst case instance for $W(n)$ with $p_n > 0$. Then for every value of parameter m, such that $1 \leq m \leq n-1$, W is upper bounded by the optimal value of the following non-linear program,*

$$\begin{aligned}
\max \quad & u(q, r, p_1, \ldots, p_m) \\
& = \frac{F(0,q) + (p_m+1)F(q, q+r) + \sum_{i=1}^{m} p_i(p_i+1)\frac{q'(i)}{q'(i)+1}}{F(p(m)+r, p(m)+r+q) + (p_m+1)F(p(m), p(m)+r) + \sum_{i=1}^{m} p_i(p_i+1)\frac{p(i)}{p(i)+1}} \\
s.t. \quad & q \geq p(m) \qquad (5.19) \\
& p_i \geq p_{i+1} \quad \forall 1 \leq i \leq m-1 \\
& q, p_1 \geq \beta - 1 \\
& p(m) + r \leq \frac{1}{\beta - 1} \\
& q'(i) = q + r + \sum_{i \leq j \leq m} p_j \quad \forall i \leq m \\
& q, r \geq 0; \quad p_i \geq 0 \forall i
\end{aligned}$$

Proof. Let $U(m)$ denote the optimal value of the above program and note that the program is independent of n. So, consider any $n \geq 2$ such that there is an optimal solution $q, p_1, \ldots, p_n$ to program (5.18) with $p_n > 0$. We show that for every $m \leq n-1$, $u(q, \sum_{i=m+1}^{n} p_i, p_1, \ldots, p_m)$ upper bounds $W(n)$. Since the choice of n is arbitrary (as long as $p_n > 0$ and $n \geq 2$), this implies, $U(m) \geq W$ for every $m \leq p-1$. The constraints in the program (5.19) follow almost directly from their counterparts in (5.18).

Fix arbitrary $m \geq 1$. To show $u(q, \sum_{i=m+1}^{n} p_i, p_1, \ldots, p_m) \geq W(n)$, let $p(i) = \sum_{j=1}^{i} p_j$ and $q(i) = q + \sum_{j=i}^{n} p_j$ as before. We have from Lemma 53, $\frac{\frac{q(i)}{q(i)+1}}{\frac{p(i)}{p(i)+1}} \geq \frac{\frac{q(n)}{q(n)+1}}{\frac{p(n)}{p(n)+1}} \geq \beta$ for every i. So for every $i \geq m+1$, replacing $p_i(p_i+1)\frac{q(i)}{q(i)+1}$ in the numerator (5.14) and $p_i(p_i+1)\frac{p(i)}{p(i)+1}$ in the denominator (5.15) by $(p_m+1)p_i\frac{q(i)}{q(i)+1}$ and $(p_m+1)p_i\frac{p(i)}{p(i)+1}$ respectively, we get,

$$V(q, p_1, \ldots, p_n) = \frac{F(0,q) + (1+p_m)\sum_{i=m+1}^{n} p_i \frac{q(i)}{q(i)+1} + \sum_{i=1}^{m} p_i(p_i+1)\frac{q(i)}{q(i)+1}}{F(p(n), p(n)+q) + (1+p_m)\sum_{i=m+1}^{n} p_i \frac{p(i)}{p(i)+1} + \sum_{i=1}^{m} p_i(p_i+1)\frac{p(i)}{p(i)+1}}.$$

Due to Lemma 49 *(1.)*, we have $V(q, p_1, \ldots, p_n) \geq W(n)$. We now apply Lemma 52

on V with $\lambda = 1 + p_m$ to obtain,

$$\frac{F(0,q) + (1+p_m)\sum_{i=m+1}^{n} F(q(i+1), q(i)) + \sum_{i=1}^{m} p_i(p_i+1)\frac{q(i)}{q(i)+1}}{F(p(n), p(n)+q) + (1+p_m)\sum_{i=m+1}^{n} F(p(i-1), p(i)) + \sum_{i=1}^{m} p_i(p_i+1)\frac{p(i)}{p(i)+1}} > V(q, p_1, \ldots, p_n).$$

Where we applied the lemma once for every $j \geq m+1$. Note that a crucial requirement for applying Lemma 52 requires $q + p_n > p(n)$, and this follows from Lemma 50. Now, since the terms $F(q(i+1), q(i)))$ and $F(p(i-1), p(i))$ telescope, the LHS above is exactly $u(q, r, p_1, \ldots, p_m)$ for $r = \sum_{i \geq m+1} p_i$. To finish the proof, note that $q, p_1, \ldots, p_n$ is optimal for (5.18) and therefore the constraints in (5.19) follow directly from their counterparts in (5.18) with the substitution $r = \sum_{i \geq m+1} p_i$. □

We are now ready to combine everything together and show the approximation bounds from Theorem 45.

Proof. (of Theorem 45) We argued earlier that $W(2) > W(1)$ by finding a local optimum that lower bounded $W(2)$. Using program (5.18) we can now compute $W(2)$ exactly. Similarly, we find $W(4) = 1.060369 > W(3) > W(2)^2$. This implies that every worst case has at least 4 regular jobs. We deal with the possibility of more than 4 jobs by using the family of upper bounds for $m = 3$. Solving the 5 variable problem using Couenne, we get $U(3) = 1.060428 > W^3$. Therefore $1.060369 < W = \beta < 1.060428$. □

5.4.2 General Case

Using notation from earlier sections, the appointment scheduling problem in its most general form can be stated as,

[2] $W(4) = 1.060368904346609$ for $q = 7.386816352844115, p_4 = 0.0505739093477502, p_3 = 0.4491255264768826, p_2 = 1.8092985887683977, p_1 = 2.282536937503935$, $W(3) = 1.060368797614823$, $W(2) = 1.0603032474797747$,$W(1) = 1.056749812107016$

[3] More precisely, $U(3) = 1.060428822765705$ for $q = 7.2120992284622245, r = 0.37782737463982036, p_3 = 0.3825692129329594, p_2 = 1.792716190567159, p_1 = 2.2707094349145875$

$$\min_{\pi:[n]\leftrightarrow[n]} \min_{x_1,\ldots,x_n} \quad \max_{t_i\in[0,\Delta_i]\,\forall i\in[n]} \quad \sum_{i\in[n]} \max\{o_{\pi(i)}(C_{\pi(i)} - A_{\pi(i)-1}), u_{\pi(i)}(A_{\pi(i)-1} - C_{\pi(i)})\}. \tag{5.20}$$

So far we worked in the regime of homogeneous underage costs, where we have a closed form solution for the optimal allocation given some fixed order. Using this closed form solution we could explicitly write the minimum allocation cost of any given order, and the problem of finding an optimal order (one that minimizes cost) was a type of min-sum scheduling problem with concave objective over completion times. In the general case, where we make no assumptions on underage costs u_i, we follow a similar recipe and give a $4\alpha n$ approximation, where $\alpha \leq 4 + \epsilon$. The main idea is still to use a closed form allocation that allows us to explicitly write the optimization problem and leverage connections with scheduling theory.

Given an an arbitrary order, consider the allocation $x_j = \Delta_j \frac{o(j)}{o(j)+u_j}$ for job j, $j \in [n]$. The problem of finding a schedule π, that minimizes the cost given this allocation rule can be written as,

$$\min_{\pi:[n]\leftrightarrow[n]} \sum_{j=1}^{n} \frac{\Delta_j(\sum_{\pi(k)\leq\pi(j)} o_k)u_j}{\sum_{\pi(k)\leq\pi(j)} o_k + u_j}. \tag{5.21}$$

Observe that the above is equivalent to a min-sum scheduling problem of the form,

$$\min_{\pi} \sum_{j} f_j(C_j).$$

Where C_j is the completion time of job j which has processing time $p_j = o_j$. Functions $f_j(x) = \Delta_j u_j \frac{x}{x+u_j}$ are concave, but now we have a different function for each job and therefore, this is a more general framework than $1|\sum_j w_j f(C_j)$ (which we used in case of homogeneous underage costs). We hereafter stick to the order in the min-sum scheduling problem, which reverses the schedule in the appointment scheduling problem (5.21), same as before.

Our heuristic is to simply use an approximation algorithm for the problem $1|\sum_j f_j(C_j)$

with $f_j(x) = \Delta_j u_j \frac{x}{x+u_j}$. We show that an α approximation for this problem is a $4\alpha n$ approximation for the appointment scheduling problem (5.20). The additional factor of $4n$ arises from the fact that (5.21) does not equal (5.20). More specifically, we are using the closed form allocation formula $x_j = \Delta_j \frac{o(j)}{o(j)+u_j}$, which does not give the optimal allocation in general. In fact, as we saw in Example 3 earlier, this allocation can be arbitrarily worse than the optimal allocation for certain orders. So it is perhaps a little surprising that using this allocation we can still achieve some approximation bound. Note that for the min-sum scheduling problem, $1|\sum f_j(C_j)$, [CMSV16] gives a $4+\epsilon$-approximation for arbitrary f_j. Conceivably, this could be improved for concave functions, and more specifically the function we have here, but we leave that open for future work. Using the algorithm in [CMSV16] we have a $(16+\epsilon)n$ approximation, and we show that this is tight up to constant factors by giving an instance where the optimal value of (5.21) is $\Omega(n)$ times the optimal for (5.20).

Lemma 56. *Using an α approximation for $1|\sum f_j(C_j)$, with $f_j(x) = \Delta_j u_j \frac{x}{x+u_j}$ for every j, we have a $4\alpha n$ approximation for (5.20).*

Proof. We claim that the optimal value of (5.21) is at most a factor $4n$ times the optimal for (5.20). We show this by proving there always exists an order such that given this order the cost of the formula allocation $x_j = \Delta_j \frac{o(j)}{o(j)+u_j}$, is within a factor $4n$ of the optimal value of (5.20). Then the guarantee follows due to an α approximation for (5.21).

For jobs $j \in [n]$, let y_j be a fixed optimal allocation for an optimal order. Recall that all jobs underaged and overaged are worst cases for y_j therefore, $\sum_j u_j y_j$ equals the optimal value of (5.20). Further, let C_j denote the completion time when all jobs are overaged and B_j the anointed end time, for every job $j \in [n]$. Observe that for every job j, either the cost of underaging, $u_j y_j$, or overaging, $o_j(C_j - B_j)$, is at least $\Delta_j \frac{o_j u_j}{o_j+u_j} \geq \Delta_j \frac{\min\{o_j,u_j\}}{2}$. With $c_j = \max\{u_j y_j, o_j(C_j - B_j)\}$ we therefore have, $c_j \geq \Delta_j \frac{\min\{o_j,u_j\}}{2}$ and $\sum_j c_j \leq 2\sum_j u_j y_j$. Now, let S denote the set of jobs such that $c_j < \Delta_j u_j/2$. Then, for every job $j \in S$ we have, $c_j \geq \Delta_j o_j/2$.

Consider the order that schedules jobs in S in ascending order of o_j and subse-

quently schedules the remaining jobs in arbitrary order. Re-index the jobs so that job j is the j^{th} job scheduled in this order and change S accordingly. Also, let $o'(j) = \sum_{k \le j} o_k$. We claim that the cost of the formula allocation $x_j = \Delta_j \frac{o'(j)}{u_j + o'(j)}$ for this order is at most $4n$ times the optimal cost. Recall that all jobs underaged is a worst case for the formula allocation and observe that, $\sum_{j \in S} u_j x_j \le \sum_{j \in S} \Delta_j \min\{u_j, o'(j)\} \le \sum_{j \in S} \Delta_j \min\{u_j, no_j\} \le 2n \sum_{j \in S} c_j \le 4n \sum_{j \in S} u_j y_j$. Finally, note that for a job $i \notin S$, we have $u_i x_i \le \Delta_i u_i \le 2c_i$, regardless of the order. $\square$

The following example now shows that there is a gap of $\Omega(n)$ between the optimal values of (5.20) and (5.21) and thus, the analysis above is tight up to constant factors.

Consider jobs $1, \ldots, 2n$ and an order such that job i is the $2n - i + 1^{th}$ appointment scheduled. Let even indexed jobs have large underage cost $u_{2i} = M$ such that $\frac{M}{M + o(2n)} \to 1$, and small overage cost $o_{2i} = 1 \ll M$, for every $i \in [n]$. For the odd indexed jobs, let $o_{2i-1} = u_{2i-1} = 1$ for every $i \in [n]$. Therefore, $\frac{u_{2i+1}}{u_{2i+1} + o(2i+1)} \le \frac{u_{2i-1}}{u_{2i-1} + o(2i-1)}$ for every $i \in [n]$. Finally, let $\Delta_{2i} = 1$ and $\Delta_{2i-1} = 0$ for every $i \in [n]$. Now, for any two consecutive jobs $2i, 2i-1$, using Theorem 38 we have in the optimal allocation, $x_{2i} = 0$ and $x_{2i-1} = (\Delta_{2i} o(2i) + \Delta_{2i-1} o(2i-1)) \frac{1}{u_{2i-1} + o(2i-1)}$. Then since $u_{2i-1} = o_{2i} = 1$, we have $u_{2i-1} x_{2i-1} = \Delta_{2i} \frac{o(2i) u_{2i-1}}{u_{2i-1} + o(2i-1)} = 1$. Therefore, the cost of this order is $\sum_{i \in [2n]} u_i x_i = n$. This establishes an upper bound on the optimal value of (5.20) for this instance. Next, consider an arbitrary order π, where we use the formula allocation. Let $o'(i) = \sum_{\pi(k) \le \pi(i)} o_k$. Then based on the original indexing, the cost of underaging an even indexed job is given by $\Delta_{2i} \frac{o'(2i) u_{2i}}{o'(2i) + u_{2i}} \approx \Delta_{2i} o'(2i)$. Therefore the total cost for this allocation is $\sum_{i \in [n]} \Delta_{2i} o'(2i) \ge \sum_{i \in [n]} 2i = n(n+1)$. Since this was for an arbitrary order, we have a multiplicative gap of $\Omega(n)$ between (5.20) and (5.21).

5.5 Conclusion and Open Problems

We considered two settings in this paper, with an eye towards designing very simple but theoretically sound heuristics for each setting. In the first setting, we are given a fixed schedule of jobs and we would like to allocate appointment durations (or

alternatively, appointment start times) for each job. We found a simple LP, which gives a 2-approximation for the problem. Then we further refined this LP, resulting in a closed form solution for the optimal allocations in certain special cases (including the case of homogeneous underage cost). It is not known whether an optimal solution can be found in polynomial time for the general appointment allocation problem.

In the second setting, we are allowed to change the order of jobs and so we seek an optimal order in addition to an optimal allocation given that order. We focus primarily on the case of homogeneous underage costs, where we find two simple and practical heuristics that are guaranteed to be with $\approx 6\%$ of the optimal. For the case of general underage costs, we give an $\Theta(n)$ approximation. Apart from improving on these bounds, a natural open question here is to understand the hardness of the problem, both for the general case as well as the special case of homogeneous underage costs for which polynomial time solvability is still open.

Chapter 6

Conclusion

We studied three problems in combinatorial optimization, in each of which the objective was uncertain, and this uncertainty was modeled via a robust re-formulation. For each problem, we designed and analyzed algorithms with theoretical guarantees – in some cases the best possible guarantee – and also faster and more practical algorithms. We conclude this thesis here, with a summary of our contributions.

In Chapter 3, we considered the problem of simultaneously maximizing m monotone submodular objectives subject to a cardinality constraint

$$MO_1 : \max_{A \subseteq N, |A| \leq k} \min_{i \in \{1,2,\ldots,m\}} f_i(A).$$

We focused on the setting where $m = o(k)$ and showed both a best possible $(1 - 1/e)$ approximation (by extending the algorithm in [CVZ10]) as well as a faster $O(\frac{n}{\delta^3} \log m \log \frac{n}{\delta})$ time MWU based algorithm with guarantee $(1 - 1/e)^2 - \delta$. We also showed that the classical greedy algorithm could be generalized to achieve a deterministic $(1 - 1/e) - \epsilon$ algorithm with runtime $O(kn^{m/\epsilon^4})$. Our guarantees are asymptotic in nature i.e., they tend to these constant values for $k \to \infty$. To partially address this issue, we tested a heuristic inspired by our MWU algorithm on synthetic data and found significant improvement over previous heuristics in all settings of m, k that we considered.

In Chapter 4, we considered a deletion-robust version of the single objective mono-

tone submodular function maximization problem and showed the first constant factor approximation results.

$$DRO: \max_{A \subseteq N, |A| \leq k} \min_{Z \subseteq A, |Z| \leq \tau} f(A - Z).$$

We start with a special case, which we called the case of *copies*. Here, we found exceptionally simple algorithms with the best possible asymptotic guarantee of $(1 - 1/e)$ for $\tau = o(k)$. Using insights from this special case, we then gave an algorithm with the best possible guarantee of $(1 - 1/e)$ (asymptotically), for $\tau = \frac{o(\log k)}{\log \log k}$. We further found a fast nearly linear time 0.387 algorithm for $\tau = o(\sqrt{k})$. Finally, we also gave a general black box scheme for deletion-robust maximization of monotone submodular functions subject to an Independence system.

In Chapter 5, we considered a robust formulation of the appointment scheduling problem. Here our main goal was to find simple, nearly optimal heuristics that could be used in practice, such as in appointment scheduling systems for health care. We considered two different settings. In the first setting, we assumed that the order of jobs is fixed a priori and we only seek an optimal schedule of appointment start times for the jobs. We found a simple LP, that yields a 2 approximation or the problem. We then refine this LP and show that the resulting LP lends itself to a simple closed form optimal solution, that is also an optimal allocation, under various special cases. In the second setting, the order of jobs is flexible and we seek both optimal order as well as optimal appointment schedule given this order. The problem becomes more sophisticated in this case and we focus primarily on the important special case of homogeneous under-utilization costs across jobs. We find a simple ratio based 1.0604 approximation for this case and also find a nearly matching worst case instance for the algorithm. For the general setting, we find a $\Theta(n)$ approximation.

Appendix A

Additional Proofs for Chapter 3

A.1 Negative Results

A.1.1 Analysis of Naive algorithm

Lemma 57. *Consider elements $x \in N$ in decreasing order of their values $f(x)$. The set of the first k elements (the ones with the largest value) is $\frac{1}{k-\tau}$ approximate for (k, N, τ).*

Proof. Let A be the set picked by the algorithm, with minimizer Z. From Lemma 13 we have that $g(OPT(k, N, \tau)) \leq f(OPT(k - \tau, N - Z, 0))$. Let $e \in N - Z$ be such that $f(e) = \max_{x \in N-Z} f(x)$. By definition, A contains at least one element with value $f(e)$. Submodularity then gives us, $f(OPT(k - \tau, N - Z, 0)) \leq (k - \tau)f(e) \leq (k - \tau)f(A - Z)$. □

A.1.2 Counterexample for non-super/sub modularity of g

Consider 3 elements x, x', y and let $f(x) = f(x') = f(y) = 1$, $f(\{x, y\}) = 2$ and $f(x|x') = f(x'|x) = 0$. This implies that $f(\{x', y\}) = f(\{x, x', y\}) = 2$ by submodularity. Now, note that the singleton set $g(\{x\}) = f(\emptyset) = 0$ and $g(\{x, x'\}) = \min\{f(x), f(x')\} = 1$. Hence $g(.)$ is clearly not submodular. Next, consider the sets x and $\{x, y\}$, then $g(\{x\} \cup \{x'\}) - g(x) = 1 - 0 = 1$ and $g(\{x, y\} \cup \{x'\}) - g(\{x, y\}) =$

$f(\{x, x'\}) - f(x) = 0$. Hence g is not supermodular either.

A.2 Tight analysis of Algorithm 6

Theorem 58. *The 0.387-algorithm is* $\frac{1}{2}\beta(0.5, \frac{k-2}{k-1})(> 0.387$ *asymptotically) approximate.*

Proof. Let $OPT = g(OPT(k, N, 1))$, A be the output of the 0.387-algorithm and a'_1 be the first element added to A apart from a_1. The case $z = a_1$ is straightforward since $f(A - a_1) \geq \beta(0, 1)f(OPT(k-1, N-a_1, 0)) \geq \beta(0, 1)OPT$ where the last inequality follows from Lemma 13. So assume $z \neq a_1$. Further, let $f(z|A - a_1 - z) = \eta f(A - a_1)$ which implies that $f(a'_1) \geq f(z) \geq f(z|A - a_1 - z) = \eta f(A - a_1)$ and now from Lemma 16 with N replaced by $N - a_1$, A replaced by $A - a_1$ and thus k replaced by $k - 1$, $S = a'_1$ with $s = 1$ and $l = k - 1 - |S| = k - 2$, we get,

$$f(A - a_1) \geq \beta(\eta, \frac{k-2}{k-1})f(OPT(k-1, N-a_1, 0))$$

This together with Lemma 13 implies, $f(A - a_1) \geq \beta(\eta, \frac{k-2}{k-1})OPT$. Also, we have by definition,

$$f(A - a_1 - z) = (1 - \eta)f(A - a_1) \geq (1 - \eta)\beta(\eta, \frac{k-2}{k-1})OPT$$

Further, we have,

$$\begin{aligned}
g(A) \quad &\geq \max\{f(a_1), f(A - a_1 - z)\} \\
&\geq \max\{f(z|A - a_1 - z), f(A - a_1 - z)\} \\
&\geq \max\{\eta\beta(\eta, \frac{k-2}{k-1}), (1 - \eta)\beta(\eta, \frac{k-2}{k-1})\}OPT \\
&\geq 0.5\beta(0.5, \frac{k-2}{k-1})OPT \quad [\text{for } \eta = 0.5] \\
&\xrightarrow{k\to\infty} 0.387OPT
\end{aligned}$$

□

We now give an instance where the above analysis is tight. Let the algorithm start with a maximum value element a_1, then pick a_2, and then add the set C, such that the output of the algorithm is $a_1 \cup a_2 \cup C$, with C being a set of size $k-2$. Let $f(a_1) = 1, f(a_2) = 1, f(C) = 1$ with $f(a_1 + C) = 1, f(a_1 + a_2) = 2, f(a_2 + C) = 2$ i.e. C copies a_1. Hence $f(a_1 + a_2 + C) = 2$ and $g(a_1 + a_2 + C) = f(a_1 + C) = 1$.

Let $OPT(k, N, 1)$ include a_2, a copy a_2' of a_2 (so $f(a_2') = 1, f(a_2 + a_2') = 1$) and a set D of $k-2$ elements of value $\frac{1}{(k-2)\beta(0,1)}$ each, such that $f(OPT(k, N, 1)) = 1 + (k-2)\frac{1}{(k-2)\beta(0,1)} = 1 + \frac{e}{e-1} = \frac{2}{\beta(0.5,1)}$. Observe that the small value elements are all minimizers and $g(OPT(k, N, 1)) \approx \frac{2}{\beta(0.5,1)}$ as k becomes large. Note that $f(D) = \frac{f(C)}{\beta(0,1)}$ and we can have sets C and D as above based on the worst case example for the greedy algorithm given in [NWF78].This proves that the inequality in Lemma 16 is tight.

A.3 Analysis of Algorithm 7

Theorem 59. *Algorithm 7 is* $0.5547 - \Omega(1/k)$ *approximate.*

Proof. Let A denote the output and $A_0 \subset A$ denote $\{a_1, a_2\}$. Due to submodularity, there exists at most two distinct $x \in A$ with $f(x|A - x) > \frac{f(A)}{3}$. Additionally, for every $x \notin A_0$, we have that $f(x|S) \leq f(a_1)$ and $f(x|S) \leq f(a_2|a_1)$ for arbitrary subset S of A containing A_0 and $x \notin S$. This implies that that $2f(x|S) \leq f(A_0) \leq f(S)$, which gives us that $f(x|S) \leq \frac{f(S+x)}{3}$.

Note that due to condition in Phase 1, the algorithm ignores a_1 even if it is not a minimizer, as long as its marginal is more than a third the value of the set at that iteration. At the end of Phase 1, if a_2 has marginal more than third of the set value, then it is ignored until its contribution/marginal decreases. Phase 3 adds greedily (without ignoring any element added). As argued above, no element other than a_1, a_2 can have marginal more than a third of the set value at any iteration, so during Phase 2 we have that a_2 is also a minimizer.

We will now proceed by splitting into cases. Denote $g(OPT(k, N, 1))$ as OPT and recall from Lemma 13, $OPT \leq f(OPT(k-1, N - a_i, 0))$ for $i \in \{1, 2\}$. Also, let the

set of elements added to A_0 during Phase 1 be $U = \{u_1, \ldots, u_p\}$, similarly elements added during Phases 2 and 3 be $V = \{v_1, \ldots, v_q\}$,$W = \{w_1, \ldots, w_r\}$ respectively, with indexing in order of addition to the set. Finally, let $\alpha_p = \frac{p-2}{k-1}$, $\alpha_q = \frac{q-1}{k-1}$, $\alpha_r = \frac{r}{k-1}$, $\alpha = \alpha_p + \alpha_q + \alpha_r = \frac{k-5}{k-1}$, and assume $k \geq 8$.

Case 1: Phase 2,3 do not occur i.e. $p = k-2, q = r = 0$.

Since we have,

$$\begin{aligned} f(A - a_1) &\overset{(a)}{\geq} f(a_2) + \beta\Big(0, \frac{k-2}{k-1}\Big)(f(OPT(k-1, N-a_1, 0)) - f(a_2)) \\ &\geq \beta\Big(0, \frac{k-2}{k-1}\Big) f(OPT(k-1, N-a_1, 0)) \\ &\overset{(b)}{\geq} \beta\Big(0, \frac{k-2}{k-1}\Big) OPT, \end{aligned} \tag{A.1}$$

where (a) follows from Lemma 1 and (b) from Lemma 13. This deals with the case $z = a_1$. If $z = u_p$, we have,

$$\begin{aligned} f(A - u_p) \geq f(A - u_p - a_1) &\overset{(c)}{\geq} \beta\Big(0, \frac{k-3}{k-1}\Big) f(OPT(k-1, N-a_1, 0)) \\ &\geq \beta\Big(0, \frac{k-3}{k-1}\Big) OPT, \end{aligned}$$

where (c) is like (A.1) above but with $k-2$ replaced by $k-3$ when using Lemma 1. Finally, let $z \notin \{a_1, u_p\}$, then due to the Phase 1 termination criteria, we have $f(a_1|A - a_1 - u_p) \geq f(A - u_p)/3$, which implies that,

$$2f(a_1|A - a_1 - u_p) \geq f(A - a_1 - u_p) \tag{A.2}$$

Now letting $\eta = \frac{f(z|A-a_1-u_p-z)}{f(A-a_1-u_p)}$, we have by submodularity $f(z|A-z) \leq \eta f(A-a_1-u_p)$ and by definition $f(A - a_1 - u_p - z) = (1-\eta) f(A - a_1 - u_p)$. Using the above we

get,

$$
\begin{aligned}
f(A-z) &\overset{(d)}{\geq} \max\{f(a_1), f(A-u_p-z)\} \\
&\geq \max\{f(z|A-a_1-u_p-z), f(A-a_1-u_p-z)+f(a_1|A-a_1-u_p-z)\} \\
&\geq \max\{\eta f(A-a_1-u_p), (1-\eta)f(A-a_1-u_p)+f(a_1|A-a_1-u_p)\} \\
&\geq \max\{\eta, (1-\eta)+\frac{1}{2}\}f(A-a_1-u_p) \qquad \text{(A.3)}
\end{aligned}
$$

where (d) follows from monotonicity and the fact that $a_1 \in A-z$ and $A-u_p-z \subset A-z$. Now, from Lemma 16 with $S=\{a_2,u_1\}$, $l=p-2$, k replaced by $k-1$, N by $N-a_1$ and $s=1$, we have, $f(A-a_1-u_p) \geq \beta(\eta,\alpha_p)f(OPT(k-1,N-a_1,0)) \geq \beta(\eta,\alpha_p)OPT$. Substituting this in (A.3) above we get,

$$
\begin{aligned}
f(A-z) \qquad & \max\{\eta, \frac{3}{2}-\eta\}\beta(\eta,\alpha_p)OPT \\
& \geq \frac{3}{4}\beta\Big(\frac{3}{4},\alpha_p\Big)OPT \quad [\eta=3/4] \\
& > \beta(0,\alpha_p)OPT = \beta\Big(0,\frac{k-4}{k-1}\Big)OPT \qquad \text{(A.4)}
\end{aligned}
$$

Case 2: Phase 2 occurs, 3 doesn't i.e. $p+q=k-2$ and $q>0$.

As stated earlier, during Phase 2, a_2 is the minimizer of $A_0 \cup U \cup (V-v_q)$. We have $g(A) \geq g(A-v_q) = f(A-v_q-a_2) = f(a_1+U)+f(V-v_q|a_1+U)$. Further, since the addition rule in Phase 2 ignores a_2, we have from Lemma 1, $f(V-v_q|a_1+U) \geq \beta(0,\alpha_q)(OPT-f(a_1+U))$, and $f(a_1+U) \geq \beta(0,\alpha_p)OPT$ follows from the previous case (to see this, suppose that the algorithm was terminated after Phase 1 and note that $z=a_2$ falls under the scenario $z \notin \{a_1,u_p\}$). Using this,

$$
\begin{aligned}
g(A-v_q) \qquad & \geq f(a_1+U)+\beta(0,\alpha_q)(OPT-f(a_1+U)) \\
\frac{f(A-z)}{OPT} \qquad & \geq (1-\beta(0,\alpha_q))\beta(0,\alpha_p)+\beta(0,\alpha_q) \\
& = \beta(0,\alpha) = \beta\Big(0,\frac{k-5}{k-1}\Big) \qquad \text{(A.5)}
\end{aligned}
$$

Case 3: Phase 3 occurs i.e., $r>0$.

We consider two sub-cases, $z \in A - W$ and $z \in W$. Suppose $z \in A - W$. Due to $f(z|A - W - z) \leq f(A - W)/3$ we have, $f(A - W) \leq \frac{3}{2} f(A - W - z)$. Also, $f(W|A - W - z) \geq f(W|A - W) \geq \beta(0, \alpha_r)(OPT - f(A - W))$. Then using this along with the previous cases,

$$
\begin{aligned}
f(A - z) \quad &= f(A - W - z) + f(W|A - W - z) \\
&\geq f(A - W - z) + \beta(0, \alpha_r)(OPT - f(A - W)) \\
&\geq f(A - W - z) + \beta(0, \alpha_r)(OPT - \frac{3}{2} f(A - W - z)) \\
&\geq (1 - \frac{3}{2}\beta(0, \alpha_r)) f(A - W - z) + \beta(0, \alpha_r) OPT \\
&\geq (1 - \frac{3}{2}\beta(0, \alpha_r))\beta(0, \alpha_p + \alpha_q) OPT + \beta(0, \alpha_r) OPT \quad \text{[from (A.4),(A.5)]} \\
\frac{f(A - z)}{OPT} \quad &\geq 0.5 - \frac{3}{2e^{\alpha}} + \frac{1}{2}(e^{-(\alpha_p + \alpha_q)} + e^{-\alpha_r}) \\
&\geq 0.5 - \frac{3}{2e^{\alpha}} + e^{-\alpha/2} \quad \text{[for } \alpha_r = \alpha_p + \alpha_q = \alpha/2\text{]} \\
&= 0.5 - \frac{3}{2e^{\frac{k-4}{k-1}}} + e^{-\frac{k-4}{2(k-1)}} \xrightarrow{k \to \infty} 0.5547
\end{aligned}
$$

Now, suppose $z \in W$, then note that for $p + q \geq 6$ we have either $p \geq 3$, and hence due to greedy additions $f(z|A - z) \leq f(\{u_1, u_2, u_3\})/3 \leq f(A - W)/3$, or $q \geq 3$, and again due to greedy additions $f(z|A - z) \leq f(a_1 + U \cup \{v_1, v_2\})/3 \leq f(A - W)/3$.

If $q > 0$, then note that $f(A - W) \geq f(A - W - v_q) \geq \frac{3}{2} f(A - W - v_q - a_2)$ due to the Phase 2 termination conditions. Now we reduce the analysis to look like the previous sub-case through the following,

$$
\begin{aligned}
f(A - z) \quad &= f(A - W) + f(W|A - W) - f(z|A - z) \\
&\geq f(A - W) + \beta(0, \alpha_r)(OPT - f(A - W)) - f(z|A - z) \\
&\geq (1 - \beta(0, \alpha_r)) f(A - W) + \beta(0, \alpha_r) OPT - f(A - W)/3 \\
&\geq \Big(1 - \frac{3}{2}\beta(0, \alpha_r)\Big)\frac{2}{3} f(A - W) + \beta(0, \alpha_r) OPT \\
&\geq \Big(1 - \frac{3}{2}\beta(0, \alpha_r)\Big) f(A - W - v_q - a_2) + \beta(0, \alpha_r) OPT
\end{aligned}
$$

Which since $f(A-W-v_q-a_2) \geq \beta\Big(0, \frac{p+q-3}{k-1}\Big)OPT$ from (A.5), leads to the same ratio asymptotically as when $z \in A-W$. The case $q=0$ can be dealt with similarly by using $f(A-W) \geq f(A-W-u_p) \geq \frac{3}{2}f(A-W-u_p-a_1)$

If $p+q<6$, then let $f(z|A-z)=\eta f(A)$. Now we have, $f(A-W) \geq f(A_0) = f(a_1)+f(a_2|a_1) \geq 2f(z|A-z) = 2\eta f(A)$, which further implies that $f(A-W+w_1) \geq 3\eta f(A)$ since $z \in W$. Then proceeding as in Lemma 16 with k replaced by $k-1$, $S = A-W+w_1$ and hence $s=3$ and finally $l = k-|S| = k-(p+q+2+1) \leq k-8$ gives us,

$$f(A) \geq \beta(3\eta, \frac{k-8}{k-1})f(OPT(k-1,N,0)) \geq \beta(3\eta, \frac{k-8}{k-1})OPT$$

Then using Lemma 17 we have, $f(A-z) \geq (1-\eta)\beta(3\eta, \frac{k-8}{k-1})OPT \geq \beta(0, \frac{k-8}{k-1})OPT$.

□

Bibliography

[AG12] Y. Azar and I. Gamzu. Efficient submodular function maximization under linear packing constraints. ICALP, pages 38–50, 2012.

[AHK12] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8:121–164, 2012.

[AK07] S. Arora and S. Kale. A combinatorial, primal-dual approach to semidefinite programs. In *STOC*, pages 227–236, 2007.

[AS04] A. A Ageev and M. I Sviridenko. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, 2004.

[Bie06] D. Bienstock. *Potential function methods for approximately solving linear programming problems: theory and practice*, volume 53. Springer, 2006.

[BLQ12] M. A Begen, R. Levi, and M. Queyranne. A sampling-based approach to appointment scheduling. *Operations Research*, 60(3):675–681, 2012.

[BMSC17] I. Bogunovic, S. Mitrovic, J. Scarlett, and V. Cevher. Robust submodular maximization: A non-uniform partitioning approach. In *ICML*, 2017.

[BQ11] M. A Begen and M. Queyranne. Appointment scheduling with discrete random durations. *Mathematics of Operations Research*, 36(2):240–257, 2011.

[BS03] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical programming*, 98(1-3):49–71, 2003.

[BS04a] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[BS04b] D. Bertsimas and M. Sim. The price of robustness. *Operations research*, 52(1):35–53, 2004.

[BTEGN09] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Princeton University Press, 2009.

[BV14] A. Badanidiyuru and J. Vondrák. Fast algorithms for maximizing submodular functions. In *SODA '14*, pages 1497–1514. SIAM, 2014.

[CCPV07] G. Calinescu, C. Chekuri, Martin Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *IPCO*, volume 7, pages 182–196, 2007.

[CCPV11] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 40(6):1740–1766, 2011.

[CJV15] C. Chekuri, T.S. Jayram, and J. Vondrak. On multiplicative weight updates for concave and submodular function maximization. ITCS, pages 201–210, 2015.

[CMSV16] M. Cheung, J. Mestre, D.B. Shmoys, and J. Verschae. A primal-dual approximation algorithm for min-sum single-machine scheduling problems. *CoRR*, abs/1612.03339, 2016.

[CV03] T. Cayirli and E. Veral. Outpatient scheduling in health care: a review of literature. *Production and operations management*, 12(4):519–549, 2003.

[CVZ10] C. Chekuri, J. Vondrák, and R. Zenklusen. Dependent randomized rounding via exchange properties of combinatorial structures. In *FOCS 10*, pages 575–584. IEEE, 2010.

[DG03] B. Denton and D. Gupta. A sequential bounding approach for optimal appointment scheduling. *IIE transactions*, 35(11):1003–1016, 2003.

[DV12] S. Dobzinski and J. Vondrák. From query complexity to computational complexity. In *STOC '12*, pages 1107–1116. ACM, 2012.

[EAVSG09] K. El-Arini, G. Veda, D. Shahaf, and C. Guestrin. Turning down the noise in the blogosphere. In *ACM SIGKDD*, pages 289–298, 2009.

[ELMS+10] L. Epstein, A. Levin, A. Marchetti-Spaccamela, N. Megow, J. Mestre, M. Skutella, and L. Stougie. Universal sequencing on a single machine. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 230–243, 2010.

[Fei98] U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM (JACM)*, 45(4):634–652, 1998.

[Fle00] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. *SIAM Journal on Discrete Mathematics*, 13(4):505–520, 2000.

[FNS11] M. Feldman, J.S. Naor, and R. Schwartz. A unified continuous greedy algorithm for submodular maximization. In *FOCS 11*, pages 570–579, 2011.

[FW14] Y. Filmus and J. Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM Journal on Computing*, 43:514–542, 2014.

[GK94] M. Grigoriadis and L. Khachiyan. Fast approximation schemes for convex programs with many blocks and coupling constraints. *SIAM Journal on Optimization*, 4(1):86–107, 1994.

[GK04] N. Garg and R. Khandekar. Fractional covering with upper bounds on the variables: Solving lps with negative entries. In *European Symposium on Algorithms*, pages 371–382, 2004.

[GK07] N. Garg and J. Koenemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. *SIAM Journal on Computing*, 37(2):630–652, 2007.

[GK11] D. Golovin and A. Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artificial Intelligence Research*, 2011.

[GKS05] C. Guestrin, A. Krause, and A.P. Singh. Near-optimal sensor placements in gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 265–272. ACM, 2005.

[GR06] A. Globerson and S. Roweis. Nightmare at test time: robust learning by feature deletion. In *Proceedings of the 23rd international conference on Machine learning*, pages 353–360. ACM, 2006.

[GSW06] L. V Green, S. Savin, and B. Wang. Managing patient service in a diagnostic medical facility. *Operations Research*, 54(1):11–25, 2006.

[HJ15] W. Höhn and T. Jacobs. On the performance of smiths rule in single-machine scheduling with nonlinear cost. *ACM Transactions on Algorithms (TALG)*, 11(4):25, 2015.

[HK16] X. He and D. Kempe. Robust influence maximization. In *SIGKDD*, pages 885–894, 2016.

[IU06] IBM and Carnegie Mellon University. Couenne, an exact solver for nonconvex MINLPs. *https://projects.coin-or.org/Couenne/*, 2006.

[Kar72] R. M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[KG05] A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. UAI'05, pages 324–331, 2005.

[KGGK06] A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pages 2–10. ACM, 2006.

[KK07] G. C Kaandorp and G. Koole. Optimal outpatient appointment scheduling. *Health Care Management Science*, 10(3):217–229, 2007.

[KKT03] D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *ACM SIGKDD*, pages 137–146, 2003.

[KLG+08] A. Krause, J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloutsos. Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management*, 134(6):516–526, 2008.

[KLTZ13] Q. Kong, C. Lee, C. Teo, and Z. Zheng. Scheduling arrivals to a stochastic service delivery system using copositive cones. *Operations research*, 61(3):711–726, 2013.

[KMGG08] A. Krause, H B. McMahan, C. Guestrin, and A. Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9:2761–2801, 2008.

[KY07] C. Koufogiannakis and N. Young. Beating simplex for fractional packing and covering linear programs. In *FOCS*, pages 494–504, 2007.

[KZK17] E. Kazemi, M. Zadimoghaddam, and A. Karbasi. Deletion-robust submodular maximization at scale. *arXiv preprint arXiv:1711.07112*, 2017.

[LB11] H. Lin and J. Bilmes. A class of submodular functions for document summarization. In *ACL*, pages 510–520, 2011.

[LCK+10] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *JMLR*, 11:985–1042, 2010.

[LKG+07] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2007.

[LWK+13] Y. Liu, K. Wei, K. Kirchhoff, Y. Song, and J. Bilmes. Submodular feature selection for high-dimensional acoustic score spaces. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7184–7188. IEEE, 2013.

[MBK+15] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause. Lazier than lazy greedy. In *AAAI*, 2015.

[MRZ14] H. Mak, Y. Rong, and J. Zhang. Appointment scheduling with limited distributional information. *Management Science*, 61(2):316–334, 2014.

[MSS14] S. Mittal, A.S. Schulz, and S. Stiller. Robust appointment scheduling. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 28, 2014.

[MV13] N. Megow and J. Verschae. Dual techniques for scheduling on a machine with varying speed. In *ICALP*, pages 745–756, 2013.

[NW78] G.L. Nemhauser and L.A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Mathematics of operations research*, 3(3):177–188, 1978.

[NWF78] G.L. Nemhauser, L.A. Wolsey, and M.L. Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

[OSU15] J. Orlin, A.S. Schulz, and R. Udwani. Robust monotone submodular function maximization. *CoRR*, abs/1507.06616, 2015.

[OUS+08] A. Ostfeld, J.G. Uber, E. Salomons, J.W. Berry, Phillips C.A. Hart, W.E., J.P. Watson, G. Dorini, P. Jonkergouw, Z. Kapelan, and F. di Pierro. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. *Journal of Water Resources Planning and Management*, 2008.

[PST91] S. Plotkin, D. Shmoys, and E. Tardos. Fast approximation algorithms for fractional packing and covering problems. In *FOCS*, pages 495–504, 1991.

[PY00] C. Papadimitriou and M. Yannakakis. On the approximability of tradeoffs and optimal access of web sources. In *FOCS*, pages 86–92, 2000.

[RC03] L. W Robinson and R. R Chen. Scheduling doctors' appointments: optimal and empirically-based heuristic policies. *IIE Transactions*, 35(3):295–307, 2003.

[SB14] A. Singla and I. Bogunovic. Near-optimally teaching the crowd to classify. In *ICML*, pages 154–162, 2014.

[Smi56] W. E Smith. Various optimizers for single-stage production. *Naval Research Logistics (NRL)*, 3(1-2):59–66, 1956.

[SW] Sebastian S. and Andreas W. Increasing speed scheduling and flow scheduling. In *ISAAC 2010*, pages 279–290.

[TCG+09] M. Thoma, H. Cheng, A. Gretton, J. Han, HP. Kriegel, A.J. Smola, L. Song, S.Y. Philip, X. Yan, and K.M. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *SDM*, pages 1076–1087. SIAM, 2009.

[VCZ11] J. Vondrák, C. Chekuri, and R. Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. In *STOC '11*, pages 783–792. ACM, 2011.

[Von08] J. Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In *STOC*, pages 67–74, 2008.

[Von13] J. Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM Journal on Computing*, 42(1):265–304, 2013.

[Wan93] P P. Wang. Static and dynamic scheduling of customer arrivals to a single-server system. *Naval Research Logistics (NRL)*, 40(3):345–360, 1993.

[Wan99] P P. Wang. Sequencing and scheduling n customers for a stochastic server. *European journal of operational research*, 119(3):729–738, 1999.

[You95] N. Young. Randomized rounding without solving the linear program. In *SODA*, volume 95, pages 170–178, 1995.

[You01] Neal E Young. Sequential and parallel algorithms for mixed packing and covering. In *FOCS*, pages 538–546, 2001.