Submodular Optimization Under Uncertainty

Roie Levin

CMU-CS-22-140
August 16, 2022

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Anupam Gupta, Chair
R. Ravi
David Woodruff
Chandra Chekuri
Joseph (Seffi) Naor

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation award under grant numbers CCF-1536002, CCF-1540541, CCF-1617790, CCF-1907820, CCF1955785, and CCF-2006953, as well as by US-Israel BSF grant 2018352 and by ISF grant 2233/19 (2027511).

Keywords: Online Algorithms, Optimization Under Uncertainty, Submodular Optimization, Approximation Algorithms.

לאמא, אבא, איתי, ואיל.

Abstract

Submodular functions, which are a natural discrete analog of convex/concave functions, strike a sweet spot between generality and structure: they model an immense variety of applications in computer science and beyond, but, at the same time, are sufficiently well behaved that they can be optimized very effectively in theory and in practice. Optimization tasks involving submodular functions have classically been studied in settings of complete information, i.e. where the entire input is known upfront. However, such perfect, full-information access to the input is unrealistic in many applications, and indeed designing algorithms under uncertainty has become an important trend in modern computer science.

In this thesis, we study algorithms for submodular optimization in scenarios with incomplete information. We study three different kinds of uncertain environments:

(a) *Online* settings where problem constraints are revealed piecemeal and the algorithm must commit to irrevocable decisions as it maintains feasibility. The challenge is to make decisions that are robust to the final realization of the problem, even with limited information.

(b) *Dynamic* settings where constraints can not only be added but also removed over time. The goal is to design algorithms that maintain feasible, near optimal solutions at every stage that are stable, in the sense that they change as little as possible between time steps.

(c) *Streaming* settings where the input is too large to hold in memory all at once. The algorithm must adequately solve problems with only limited memory after one (or few) sequential passes over the data.

Prior to our work, no algorithms were known for several important instances of submodular optimization in uncertain environments; we improve known algorithms for others. In a few cases we apply insights from these submodular optimization algorithms to problems that are not on their surface related to submodularity. We hope that the techniques we develop find uses elsewhere.

Acknowledgments

The world has been a frightening place of late. I am beyond grateful I had the world of math to take refuge in, and I am well aware that I would not have been able to do so without a vast and unwavering support network.

I owe my PhD to the incredible mentorship of my advisor Anupam. Thank you for taking a chance on me and taking me on as a student when you had no evidence I could hack it as a researcher. I have grown skeptical of plenty of things in graduate school but never of your clarity and creativity in research. I feel blessed to have experienced even a few exhilarating moments of mathematical discovery with you and hope we can continue collaborating in the future.

Thank you to my other brilliant collaborators: Christian Coester, Parikshit Gopalan, Greg Kehne, Seffi Naor, Anish Sevekari, Ohad Talmon, David Wajc, Udi Wieder, and David Woodruff. Special thanks to Parikshit, Udi and Seffi for hosting me during wonderful summer visits. Thank you to my mentors from my past who helped me on the road to the PhD: Oren Etzioni, Mark Hopkins, Anna Lysyanskaya, Mr. (Tarly) Manak. Special thanks to Deb, who is hands down the best administrator in the world and a nontrivial fraction of the reason CMU is a great institution.

Thank you to all my homies from all the chapters of my life, whom I love so dearly: Corbin, Lawrence, Veronica, Hannah, Leo, Brandon, Adison, Adam, Sydney, Evan, Michael, Sonia, Hunter, Ana, Jeff, George, Shashwat, Andrew, Julia, Justin, Alvaro, Been, Nick, Vidur, Brendan, Jerome, Katy. Thanks to my grad school buddies: Ainesh, Raj, Pedro, Kevin, Alex, Ellis, Anson, David, Naama, Marina, Angela, Nic, Fait, Bailey, Colin, Ellen, Kerrek, Nathan, Goran, Laxman, Vijay, Guru, Nika, Sahil. I owe my mental health in grad school to Ascend Pittsburgh and the local bouldering community. Extra thanks to my surgeon Dr. Benjamin Ma for fixing my two shoulders three times. Greg, I would be remiss if I didn't give you your own sentence, end of sentence. Aria, I would not have made it through the PhD without you. Finally, Julia – here's to the future. When things are good, they will be good.

Thank you to my extended family, especially my savtas and sabas, Lena and Yehuda, Yehudit and Arie, who held on to life during times darker than any fiction so that I could experience this world. Finally, Ima, Aba, Itai and Eyal, I cannot think of anything to write that would do justice to what I owe you, so I will not try.

Contents

Chapter 1

Introduction

Consider the following combinatorial optimization problem. We are given a graph with edge capacities representing a local area network, along with a set of client vertices each with some positive flow demand. We wish to purchase the smallest number of servers to place in the network such that each client can feasibly route its demand to some server, all while respecting the edge capacities of the graph. What is more, this is not a static scenario. Instead the clients come and go over time. How should one approach solving such an involved problem?

As a first step, it is helpful to distill the core features of the problem. We have a ground set $\mathcal{N}$ of decisions we can make (here $\mathcal{N}$ is the set of candidate server locations), and a notion of coverage f for every subset of these locations ($f(S)$ is the total client demand that can be feasibly satisfied when S is chosen as the set of server locations). The goal is to pick a set S minimizing a linear cost function c such that coverage offered by S is best possible (i.e. all demand is satisfied). Note that maximum coverage is always achieved when $S = \mathcal{N}$, in other words when a server is opened at every vertex in the network. Hence our goal is:

$$\boxed{\begin{array}{rl} & \min\limits_{S \subseteq \mathcal{N}} c(S) \\ \text{s.t.} & f(S) = f(\mathcal{N}) \end{array}}\,.$$

This abstract min-cost subset-selection problem is intractable in general. Thankfully, in the case of the server problem, this function f satisfies the additional properties of nonnegativity, monotonicity and *submodularity* (proof due to [AGG+09], who refer to this problem as SIMULTANEOUSSOURCELOCATION). We will give a formal definition of submodularity in Chapter 2, but for now we may think of a function f as submodular if it exhibits *decreasing marginal returns* type behavior; in other words, adding an element to a set A gives larger marginal benefit than adding the same element to a superset of A. It should be intuitive that the function f in question has such a property, since a server should only become less helpful *marginally* the more servers one has already purchased.

With these assumptions, the optimization problem is known as SUBMODULARCOVER, which itself lies within a larger field of submodular optimization. Conveniently, once we have written our problem in this form, we can use known algorithms for this general version [Wol82]. As it turns out, reducing the server problem to SUBMODULARCOVER is the right approach, in the sense that the approximation guarantees for SUBMODULARCOVER are best possible in polynomial time unless $\mathsf{P} = \mathsf{NP}$, even for this special case [AGG+09].

This example highlights the expressive power of submodular functions, and indeed, these are ubiquitous in computer science and beyond. These functions, which are the natural discrete analog of convex/concave functions[1], strike a sweet spot between structure and generality. They are sufficiently well behaved that there are good algorithms for optimizing these functions in many settings. On the other hand, since many natural quantities have decreasing marginal returns, they still model an immense variety of applications. Beyond the function from the server example, functions commonly modeled as submodular are human utility functions for discrete goods in auctions [Von08, LLN06, CCPV11], the coverage of a collection of sensors [WCZW15, RP15, ZPW$^+$17, MW16] or network devices [AGG$^+$09, LPS13, KN17, LRS18, CWJ18], the influence of a set of nodes in social networks [GBLV13, LG16, TWPD17, IIOW10], feature selection/document summarization objectives in machine learning [MKSK13, LB11], diversity maximization criteria [AGHI09, ADF17], and more. An algorithm for submodular optimization is automatically an algorithm for any application that can be framed as submodular optimization; as a result, the field is a plentiful source of high impact algorithmic questions.

Yet this is not the end of our story, since the client set and their demands change with time. One way to model this is as an *online* problem where we only learn the full network traffic patterns gradually. Every time we learn a new client's demand, we must immediately and irrevocably decide which server(s) to open to satisfy the demand, and we want to minimize the total amount spent on servers over time. In the language of SUBMODULARCOVER, suppose the constraint f is only revealed piecemeal (we detail what we mean by "reveal piecemeal" formally in Chapter 2); the goal is to maintain feasibility at every step while, even with limited information, still making decisions that are robust to the final realization of the problem. Here we can study *worst-case* scenarios where we imagine an omniscient adversary chooses the request sequence, or we may make some assumptions about the input sequence. For example, in the *stochastic* paradigm, one assumes the problem input is drawn from a known or unknown distribution.

Another modeling option is to frame this as a *dynamic* problem. Whereas in the online setting we assume we discover an unknown but fixed constraint f over time, in the dynamic version we assume f actually changes over time. In the server example this means clients may not only arrive but also depart. We no longer require that decisions be irrevocable (one can show that no algorithm can be competitive if this were the case); instead our goal is to maintain a feasible and near optimal solution for the current function f, while also minimizing the amount by which our solution changes between steps. Avoiding changing the solution can be a huge boon in practice: in the server problem for example, setting up and tearing down servers is expensive and labor intensive in practice [LRS18].

In the big-data regime where the input to the problem is too large to store in memory all at once, algorithms must deal with another type of uncertainty, namely uncertainty about what is outside of main memory. Here a natural goal is to design *streaming algorithms* that can adequately solve problems with only limited space, after only one sequential pass (or few passes) over the data. Many submodular optimization problems are natural big-data questions, for example those inspired by ranking search results tasks.

In this thesis, we study problems relating to these themes and design algorithms for submodular

[1]The decreasing marginal returns description of submodular functions is most akin to the nonpositive second derivative characterization of concave functions. However, submodular functions have interesting similarities to **both** convex and concave functions. See Section 2.2 for more detail.

optimization in scenarios of incomplete information. Algorithms under uncertainty has become an important trend in modern computer science, since for many applications in practice it is unrealistic to expect complete and perfect information about the problem at hand. Moreover, since we give algorithms for general submodular optimization under uncertainty, we automatically port any downstream application of these submodular optimization tasks to the uncertain settings.

Prior to our work, no algorithms were known for several important instances of submodular optimization in uncertain environments; we improve known algorithms for others. In some cases we apply insights from these submodular optimization algorithms to solve problems that are not on their surface related to submodularity. We develop a variety of techniques along the way, and we hope these find uses elsewhere.

1.1 Overview

1.1.1 Online Submodular Cover

We begin in Chapter 3 by showing how to solve SUBMODULARCOVER problems, in the setting where submodular constraints are initially unknown and only revealed online over time. At every stage the algorithm must add to its solution in order to maintain feasibility; the difficulty arises because we require that the decisions of the algorithm be irrevocable, even though the algorithm only has incomplete information about the future.

SUBMODULARCOVER has been used in many applications to resource allocation and placement problems, by showing that the coverage function is monotone and submodular, and then applying algorithms for SUBMODULARCOVER as a black box. We port these applications to the online setting where coverage requirements change with time. E.g., in selecting influential nodes to disseminate information in social networks [GBLV13, LG16, TWPD17, IIOW10], exploration for robotics planning problems [KMGG07, JCMP17, BMKB13], placing sensors [WCZW15, RP15, ZPW$^+$17, MW16], and other physical resource allocation objectives [YCDW15, LCL$^+$16, TRPJ16]. There has been much recent interest in SUBMODULARCOVER from the networking community, as SUBMODULARCOVER models network function placement tasks [AGG$^+$09, LPS13, KN17, LRS18, CWJ18]. E.g., [LRS18] want to place middleboxes in a network incrementally, and point out that avoiding closing extant boxes is a huge boon in practice. We extend this application to the case where the coverage function f changes.

We follow the natural strategy of maintaining a solution to a linear programming (LP) relaxation of the problem, and then performing randomized rounding online. Unfortunately, the only known LP relaxation of SUBMODULARCOVER is exponential-sized and it is not known how to solve it efficiently, nor how to analyze randomized rounding given a fractional solution. We show that we can overcome these concerns and make the technique work.

The rounding is the natural one, where we sample from the fractional solution multiple times (in an online fashion); in contrast to the SETCOVER-style analysis which argues item-by-item, here we have to argue about the coverage as a whole. In particular, we exploit the relationship between the multilinear extension and the linearizations of submodular functions to reinterpret the constraints in the LP as statements about expected coverage under randomized rounding. We will demonstrate the power of this rounding analysis by reusing it in the context of a seemingly different problem in Chapter 5.

To make the above algorithms run in polynomial time, we must overcome the annoying fact that even even the separation problem for the known SUBMODULARCOVER LP is APX-hard; hence, it is not known how to solve the LP even in the offline setting. To remedy this, we use a "round-or-separate" approach: we use the rounding algorithm itself as an approximate separation oracle. Indeed, we give an approximate separation oracle that given an arbitrary solution promises to either round it to a feasible integer solution with a similar objective value, or else to output a constraint violated by the solution. This approach to blur the line between rounding and separation may prove useful for other problems in online algorithms, where we are faced with an exponential number of new constraints at each step. Furthermore, we give a new exponential-clock based sampling procedure that finds constraints violated by a large margin with high probability. We use a potential function argument to conclude that we only need a polynomial number of calls to the separation oracle in the worst case.

We end this chapter with an algorithm that performs better for the case where the submodular functions input to the algorithm are "smooth" in the sense that elements of the ground set have bounded marginal values. The intuition behind this refined analysis is that we charge each element selected by the algorithm to the elements of the optimal solution that "cover the same part of the space". To capture the overlap between two elements, we study the *Mutual Coverage*, which is a natural generalization of mutual information from information theory, and inherits properties from that literature, such as the chain-rule. It turns out that mutual coverage is the right abstract quantity to focus on for our analyses. We also have to strengthen the SUBMODULARCOVER LP slightly, and use a modified rounding-with-alterations scheme to get the tighter result.

This chapter is based on the paper [GL20b] with Anupam Gupta.

1.1.2 Random Order Set Cover

In Chapter 4, we study an important special case of ONLINESUBMODULARCOVER in more depth: the ONLINESETCOVER problem. We ask the question: can one circumvent lower bounds known for this problem when one assumes that constraints arrive in random order? This question fits into the active current of modern algorithms research known as *beyond worst-case analysis*. Traditionally, the study of algorithms has focused on fully *adversarial* models in which we assume as little as possible about the input. These often fail to capture useful structure about instances seen in practice, and thus often give overly pessimistic bounds. Beyond worst-case analysis asks what reasonable assumptions one needs to make about the input in order to improve upon the worst case bounds.

We show that the answer to the question above is yes: we give a polynomial time algorithm for ONLINESETCOVER when constraints are revealed in random order that matches the best possible approximation bound achievable offline in polynomial time (unless $\mathsf{P} = \mathsf{NP}$). We then show extensions to pure covering IPs, to NONMETRICFACILITYLOCATION, and to a variant of ONLINESETCOVER where the algorithm is given advance access to samples from the input. We also give lower bounds for related problems, including ONLINESUBMODULARCOVER.

The core contribution of this chapter is demonstrating that one can exploit randomness in the arrival order to *learn* about the underlying set system. What is more, this learning can be done fast enough (in terms of both sample and computational complexity) to build a competitive solution, even while committing to covering incoming elements immediately upon arrival. This seems like an idea with applications to other sequential decision-making problems, particularly in the RO setting.

Our algorithm is a new multiplicative-weights-based round-and-solve approach we call LearnOrCover. We maintain a coarse fractional solution that is neither feasible nor monotone increasing, but can nevertheless be rounded online to achieve the claimed guarantee. For each arriving uncovered element, we first sample from this coarse solution, then update it via a multiplicative weights rule. For the analysis, we introduce a potential function which simultaneously measures the convergence of this distribution to the optimal fractional solution, and progress towards covering the universe. Crucially, this progress is measured in expectation over the random order, thereby circumventing lower bounds for the adversarial-order setting. This gives a new offline algorithm for SETCOVER that performs a single pass through the elements. The algorithm has interesting connections to other methods in online algorithms, namely projection based algorithms and stochastic gradient descent.

This chapter is based on the paper [GKL21] with Anupam Gupta and Gregory Kehne. Sections 4.5 and 4.6 are new in this thesis.

1.1.3 Block-Aware Caching

In Chapter 5 we give an application of ONLINESUBMODULARCOVER to a variant of CACHING, which is one of the quintessential problems in online algorithms. Motivated by the design of real system storage hierarchies, we study the BLOCKAWARECACHING problem, a generalization of classic CACHING in which pages are partitioned into blocks of a fixed size, and fetching (or evicting) pages from the same block incurs the same cost as fetching (or evicting) just one page from the block. The goal is to serve a sequence of requests to pages while minimizing the total cost of fetching to (or evicting from) cache. This problem is already NP-hard offline.

We show a suite of results. For the cost model in which the cost of evictions dominates the cost of fetches, we give non-trivial algorithms, including a polylogarithmic competitive randomized algorithm which nearly matches the known lower bound from standard CACHING. We also show strong lower bounds in the reverse cost model in which the fetch cost dominates the write cost. Hence our results establish a strong separation between the tractability of the fetching and eviction cost models, which is interesting since fetching/evictions costs are the same up to an additive term for the classic caching problem.

Though this problem seems to have little to do with submodularity on the surface, our insight is to relax the block-aware caching problem to a SUBMODULARCOVER linear program: we express feasibility as the constraint that a particular sequence of monotone, submodular functions is maximized. Our formulation may be viewed as a generalization of the strengthened LP relaxation due to [BBN12b] for GENERALIZEDCACHING (a special case of BLOCKAWARECACHING), which used the so-called knapsack cover inequalities.

This establishes that BLOCKAWARECACHING is an instance of ONLINESUBMODULARCOVER. However, we require additional analysis on top of the one in Chapter 3 to get finer competitiveness bounds for this special case. Whereas in Chapter 3 we could make do with a off-the-shelf use of online covering LP solvers, here we need to open the black box. We adapt the continuous online primal-dual framework, i.e. we simultaneously maintain a feasible primal and dual solution that we carefully evolve according to continuous dynamics, and we show that the costs of the these are related.

This chapter is based on the paper [CLNT22] with Christian Coester, Seffi Naor and Ohad Talmon.

1.1.4 Fully-Dynamic Submodular Cover

In Chapter 6, we show how to solve the same class of SUBMODULARCOVER problems in a more general online setting than in Chapter 3 where constraints are not only added over time, but they can also be removed. Here we relax the restriction that decisions be completely irrevocable (one can show that no competitive algorithm can exist without this relaxation), but instead require that the solution change as little as possible between time steps. We show that only this small relaxation suffices to solve this harder problem, and furthermore to match the best possible *offline* bounds for SUBMODULARCOVER. It can be shown that it is not possible to match the offline bounds in polynomial time if no recourse is allowed [Kor04]. Our work simultaneously simplifies and unifies previous results for the special case of SETCOVER, as well as generalizes to a significantly larger class of covering problems. As before, it ports any of the many problems which can be framed as SUBMODULARCOVER to the fully-dynamic setting.

Our algorithm is local search. We maintain a candidate solution which is subset of the ground set, and update this solution dynamically via local operations. To bound the competitive ratio, we show that upon termination of the local search, this solution is identical to the output of an approximate greedy algorithm that is known to produce good solutions when run offline. To bound the number of updates to the solution, we use a potential function inspired by a generalized family of entropy functions known as *Tsallis Entropy*. We show that with every local search update we perform, this entropy must decrease by at least a minimum amount, and since the entropy is bounded above and below, this process must terminate in a predictable amount of time. To get finer results for special cases, we also extensively reuse the idea of Mutual Coverage from Chapter 3.

Our algorithm can be thought of as a general template for adapting greedy-like offline algorithms to the dynamic setting. To demonstrate its further potential for applications, we use it to derive dynamic algorithms for the metric MINIMUMSTEINERTREE problem.

This chapter is based on the paper [GL20a] with Anupam Gupta.

1.1.5 Streaming Submodular Matching

Finally, in Chapter 7 we change gears slightly from submodular covering, and instead turn to the related task of submodular maximization subject to combinatorial constraints. In particular, we study maximum submodular matching/b-matching (MSM): we imagine the the ground set elements are the edges of a graph, and a solution is feasible if and only if it is a legal matching/b-matching. We wish to find the feasible solution maximizing a given submodular function, and we consider the big-data regime where the input to the problem is too large to store in main memory. Instead, the edges of the input graph are revealed in a stream, and the algorithm must output a near optimal solution while only using a limited amount of space. We present improved upper and lower bounds for this problem.

Our algorithm uses the (randomized) primal-dual method, which originated in the study of maximum weight matching. We maintain a feasible dual solution over the course of the stream, which we used to remember a small subset of the total edge set. At the end of the stream, we further refine this set of edges to obtain a true legal matching. We argue that the quality of the optimal solution is approximately preserved, even during this aggressive two-stage pruning.

Intuitively, we may think of our dual solution as a set of vertex potentials which are a coarse proxy for the quality of the best edge incident to every vertex. These in turn govern whether or not we should remember future incoming edges (since we need not remember edges of worse quality). Crucially, this set of vertex potentials is small enough to be maintained in a stream with low space. Our LP relaxation is guided by the theory of continuous extensions of submodular functions which we will introduce shortly in Chapter 2. To our knowledge, this is the first use of primal-dual based analysis for streaming submodular optimization. We also show how to reinterpret previous algorithms for MSM in our framework.

This chapter is based on the paper [LW21] with David Wajc.

Chapter 2

Background and Preliminaries

2.1 Basic Notation and Facts

In this thesis, we deal with NP-*hard* optimization problems, that is optimization problems whose optimum solution cannot be computed in time that is polynomial in the input length, unless the famous $\mathsf{P} \neq \mathsf{NP}$ conjecture is false. To cope with intractability, we study polynomial time *approximation algorithms*, that is algorithms guaranteed to output a solution whose cost is within a bounded multiplicative factor of the optimum solution's cost. Formally, an algorithm $\mathcal{A}$ for a minimization problem Π is an α-*asymptotic approximation algorithm* if there is a constant C such that for any instance $\mathcal{I} \in \Pi$, the inequality

$$\mathcal{A}(\mathcal{I}) \leq \alpha \cdot \textsc{Opt}(\mathcal{I}) + C \tag{2.1.1}$$

holds, where $\mathcal{A}(\mathcal{I})$ is the cost of the solution algorithm $\mathcal{A}$ produces with input $\mathcal{I}$, and $\textsc{Opt}(\mathcal{I})$ is the cost of the optimal solution to $\mathcal{I}$. If (2.1.1) holds with $C = 0$, we say that $\mathcal{A}$ is an α-*approximation* algorithm for Π. Similarly, for maximization problems, we say that $\mathcal{A}$ is an α-asymptotic approximation algorithm if

$$\mathcal{A}(\mathcal{I}) \geq \frac{1}{\alpha} \cdot \textsc{Opt}(\mathcal{I}) - C, \tag{2.1.2}$$

and an α-approximation algorithm if $C = 0$.

In the context of online algorithms, we say that an algorithm $\mathcal{A}$ for a minimization problem is α-competitive if it respects inequality (2.1.1) (or (2.1.2) for maximization problems). We say an α-competitive online algorithm with $C = 0$ is α-*strictly competitive*. We sometimes say an α-competitive (or strictly competitive) algorithm $\mathcal{A}$ attains the *competitive ratio* α. In contrast to approximation algorithms which are usually required to run in polynomial time, we make no requirements about the computational efficiency of competitive online algorithms, since it is often interesting to design such algorithms even if exponential computation time is allowed. Nevertheless, all online algorithms in this thesis can me implemented in polynomial time.

For problems with arbitrarily long input sequences and unbounded costs such as CACHING (Chapter 5), we will study asymptotic approximation algorithms or non-strictly competitive online algorithms; this makes sense since the constant C becomes insignificant in the limit. However, for problems with bounded costs such as SETCOVER or SUBMODULARCOVER (Chapters 3, 4, 6 and 7), we restrict our attention to true approximation algorithms and strictly competitive algorithms. For convenience, we will henceforth abuse notation slightly and stop explicitly distinguishing between asymptotic/non-asymptotic and strict/non-strict competitive algorithms.

All logarithms in this thesis are taken to be base e. In the following definitions, let $x, y \in \mathsf{R}_+^n$ be vectors. The standard dot product between x and y is denoted $\langle x, y \rangle = \sum_{i=1}^n x_i y_i$. We use a weighted generalization of KL divergence. Given a weight function c, define

$$\mathrm{KL}_c\left(x \mid\mid y\right) := \sum_{i=1}^n c_i \left[x_i \log\left(\frac{x_i}{y_i}\right) - x_i + y_i \right]. \tag{2.1.3}$$

We will sometimes use the convention that $1 : k$ denotes that range of indices from 1 to k. We will require the simple and well known inequalities:

Fact 2.1.1. *Given positive numbers $a_1, \ldots, a_k$ and $b_1, \ldots, b_k$:*

$$\min_i \frac{a_i}{b_i} \le \frac{\sum_i a_i}{\sum_i b_i} \le \max_i \frac{a_i}{b_i}. \tag{2.1.4}$$

2.2 Submodular Set Functions

A set function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ is *submodular* if for every $A \subseteq B \subseteq \mathcal{N}$ and $x \in \mathcal{N} \backslash B$ it respects the inequality

$$f(A \cup \{x\}) - f(A) \ge f(B \cup \{x\}) - f(B).$$

It is *monotone* if $f(A) \le f(B)$ for all $A \subseteq B \subseteq \mathcal{N}$.

In this thesis, we assume access to a *value oracle* for f that computes $f(T)$ given $T \subseteq \mathcal{N}$. The *contraction* of $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ onto $\mathcal{N} \setminus T$ is defined as

$$f_T(S) = f(S \mid T) := f(S \cup T) - f(T).$$

If f is submodular then f_T is also submodular for any $T \subseteq \mathcal{N}$.

Example 2.2.1. *Let $\mathcal{N}$ be a set of random variables, and let $H : 2^{\mathcal{N}} \to \mathbb{R}_+$ be the joint entropy function. Then H is nonnegative, monotone and submodular.*

Example 2.2.2. *Let $G = (V, E)$ be a graph, and let $C_G : 2^V \to \mathbb{R}_+$ be the cut function of G, i.e. for any $U \subseteq V$, define $C_G(U)$ to be the number of edges in E with exactly one endpoint in U. Then C_G is nonnegative and submodular, but not monotone.*

We now make explicit one connection between submodular functions and concavity. Following the notation of [FH05], we define the *derivative of a set function f* as:

$$\frac{df}{dx}(S) = f(S \cup \{x\}) - f(S \backslash \{x\}).$$

We notate the m-th order derivative of f with respect to the subset $A = \{i_1, \ldots i_m\}$ as df/dA, and this quantity has the following concise expression:

$$\frac{df}{dA}(S) = \sum_{B \subseteq A} (-1)^{|B|} f((S \cup A) \backslash B).$$

Definition 2.2.3 (m-increasing [FH05]). *We say that a set function is m-increasing if all its m-th order derivatives are nonnegative, and m-decreasing if they are nonpositive. We denote by $\mathcal{D}_m^+$ and $\mathcal{D}_m^-$ the classes of m-increasing and m-decreasing functions respectively.*

Note that $\mathcal{D}_1^+$ is the class of monotone set functions, and $\mathcal{D}_2^-$ is the class of submodular set functions. When f is the joint entropy set function, the m-th derivative above is also known as the *interaction information*, which generalizes the usual mutual information for two sets of variables, to m sets of variables.

We use the following notation for the maximum and minimum marginals of f, which are commonly used parameters quantifying the "smoothness" of the submodular function:

$$f_{\max} := \max \{ f(j \mid S) \mid j \in \mathcal{N},\ S \subseteq \mathcal{N} \}, \tag{2.2.1}$$

$$f_{\min} := \min \{ f(j \mid S) \mid j \in \mathcal{N},\ S \subseteq \mathcal{N},\ f(j \mid S) \neq 0 \}. \tag{2.2.2}$$

Submodular functions are only defined over vertices of the hypercube, $\vec{x} \in \{0,1\}^{\mathcal{N}}$. For many applications it is often convenient to extend these to all fractional vectors $\vec{x} \in [0,1]^{\mathcal{N}}$. Formally, a *continuous extension* of a submodular function f is any function $\widehat{f} : [0,1]^{\mathcal{N}} \to \mathbb{R}^+$ such that $\widehat{f}(x) = f(x)$ for any $x \in \{0,1\}^{\mathcal{N}}$. There are several canonical continuous extensions of submodular functions which we make use of in this thesis. See e.g. Section 3.5 of [Von07] for a comprehensive survey and history of such extensions.

The *multilinear extension* of f is defined as:

$$F(x) := \mathop{\mathbb{E}}_{S \sim x}[f(S)] = \sum_{S \subseteq N} \prod_{i \in S} x_i \prod_{j \notin S} (1 - x_j) f(S) \tag{2.2.3}$$

where $S \sim x$ means we add each element $j \in N$ independently to S with probability x_j.

The *concave closure extension* of f is:

$$f^+(\vec{x}) := \max \left\{ \sum_{T \subseteq E} \alpha_T \cdot f(T) \;\middle|\; \sum_{T \subseteq E} \alpha_T = 1,\ \alpha_T \geq 0\ \forall T \subseteq E,\ \sum_{T \ni e} \alpha_T = x_e\ \forall e \in E \right\}.$$

In words, the concave closure is the maximum expected f-value of a random subset $T \subseteq \mathcal{N}$, where the maximum is taken over all distributions matching the marginal probabilities given by $\vec{x}$.

Next, for $S \subseteq N$, define $f^{\#S} : [0,1]^{|N|} \to \mathbb{R}^+$ as:

$$f^{\#S}(x) := f(S) + \textstyle\sum_{j \in N} f_S(j) x_j. \tag{2.2.4}$$

The *covering extension* (or the *Wolsey extension*) of f (defined but not named in [Von07]) is:

$$f^*(x) := \min_{S \subseteq N} f^{\#S}(x). \tag{2.2.5}$$

We will give more intuition for the covering extension in Chapter 3.

We mention one final continuous extension for completeness, although it does not make an appearance in this thesis. The *Lovász extension* is:

$$f^L(\vec{x}) := \mathop{\mathbb{E}}_{\lambda \sim U[0,1]}[f(\{i : x_i \geq \lambda\})].$$

The Lovász extension can be shown to be equivalent to the so-called *convex closure extension* (note that this equivalence does not hold if f is not submodular):

$$f^-(\vec{x}) := \min \left\{ \sum_{T \subseteq E} \alpha_T \cdot f(T) \;\middle|\; \sum_{T \subseteq E} \alpha_T = 1,\ \alpha_T \geq 0\ \forall T \subseteq E,\ \sum_{T \ni e} \alpha_T = x_e\ \forall e \in E \right\}.$$

The function f^L is convex if and only if f is submodular. A seminal result of Grötchel, Lovász and Schrijver [GLS88] is that the unconstrained submodular minimization problem $\min_{S \subseteq \mathcal{N}} f(S)$ can be solved in polynomial time given value oracle access to f; this algorithm is based on the ellipsoid method applied to the Lovász extension of f. On the other hand, by Example 2.2.2 maximizing a submodular function generalizes the APX-hard MAXCUT problem. In this sense, submodular functions are more akin to convex functions than concave functions.

We can naturally extend the definition of any of these extensions to contractions of functions.

2.3 Mutual Coverage.

For some of our results, we define the notion of *mutual coverage*. Independently, [IKBA21] defined and studied the same quantity under the slightly different name *submodular mutual information*.

Definition 2.3.1 (Mutual Coverage). *The* mutual coverage *between two subsets $A, B \subseteq \mathcal{N}$ (and their* conditional mutual coverage *conditioned on $C \subseteq \mathcal{N}$) with respect to a set function $f : 2^{\mathcal{N}} \to \mathbb{R}^+$ are defined as:*

$$\mathcal{I}(A; B) := f(A) + f(B) - f(A \cup B) \tag{2.3.1}$$

$$\mathcal{I}(A; B \mid C) := f_C(A) + f_C(B) - f_C(A \cup B) \tag{2.3.2}$$

This generalizes mutual information from information theory: if $\mathcal{N}$ is a set of random variables, and for $S \subseteq \mathcal{N}$, the quantity $f(S)$ is the joint entropy of the random variables in the set S, then $\mathcal{I}$ is the mutual information.

We sometimes write $\mathcal{I}^{(A)}(B) := \mathcal{I}(A; B)$ and $\mathcal{I}^{(A|C)}(B) := \mathcal{I}(A; B \mid C)$ if we want to view mutual coverage as a function of B for fixed A and C. We may think of $\mathcal{I}^{(A|C)}(B)$ intuitively as the amount of coverage B takes away from the coverage of A given that C was already chosen (or vice-versa, since the definition is symmetric in A and B). The following identity is easy to check:

Fact 2.3.2 (Chain Rule). *Mutual coverage respects the identity:*

$$\mathcal{I}(A; B_1 \cup B_2 \mid C) = \mathcal{I}(A; B_1 \mid C) + \mathcal{I}(A; B_2 \mid C \cup B_1).$$

This neatly generalizes the chain rule for mutual information. The expression $\mathcal{I}^{(A|C)}(B)$ is monotone in B by the submodularity of f, but not submodular in general (see Section 3.6.2).

2.4 SUBMODULARCOVER

A major focus of our work is the SUBMODULARCOVER problem described intuitively in the introduction (Chapter 1). Formally, given a monotone, submodular function f over ground set $\mathcal{N}$, and a linear cost function $c : \mathcal{N} \to \mathbb{R}^+$ on the ground set, the goal is to output a minimum cost subset $S \subseteq \mathcal{N}$ such that $f(S) \geq f(\mathcal{N})$. We let $m = |\mathcal{N}|$ denote the size of the ground set, and $c_{\max}$ and $c_{\min}$ denote the largest and smallest costs of elements respectively.

Wolsey [Wol82] (see also [NWF78, FNW78]) gave the following LP relaxation for SUBMODULARCOVER:

$$\boxed{\begin{array}{rl} \min & \displaystyle\sum_{v \in \mathcal{N}} c(v) \cdot x_v \\ \text{subject to} & \\ \forall S \subseteq \mathcal{N}: & \displaystyle\sum_{v \notin S} f(v \mid S) \cdot x_v \geq f(\mathcal{N}) - f(S) \\ \forall v \in \mathcal{N}: & x_v \geq 0 \end{array}} \tag{2.4.1}$$

The constraints of this LP may be viewed as knapsack cover inequalities for a linearized version of the function f. Wolsey proved that the integer solutions of this LP are precisely the solutions to SUBMODULARCOVER:

Claim 2.4.1 (Proposition 2 of [Wol82])**.** *A set S has $f(S) = f(\mathcal{N})$ if and only if χ_S, the characteristic vector of S, is a feasible integer solution to* (2.4.1).

Furthermore, Wolsey [Wol82] showed that the natural greedy algorithm gives a $1 + \ln(f_{\max}/f_{\min})$ approximation for SUBMODULARCOVER; this guarantee is tight unless $\mathsf{P} = \mathsf{NP}$ even for the special case of SETCOVER [DS14] (see Section 2.5 below). One can also solve SUBMODULARCOVER via repeated sequential use of submodular maximization subject to a cardinality/knapsack constraint (e.g. [Svi04] or the survey [BF18]). Fujito [Fuj99] gave a dual greedy algorithm that generalizes the $F-$approximation for SETCOVER [Hoc82] (the parameter F is known as the *frequency*, or the maximum number of sets any one element appears in).

2.5 SETCOVER and ONLINESETCOVER

An important special case of SUBMODULARCOVER that study in depth in this work is the famous SETCOVER problem. In SETCOVER, we are given a set system $(U, \mathcal{S})$ (where U is a ground set of size n and $\mathcal{S}$ is a collection of subsets with $|\mathcal{S}| = m$), along with a cost function $c : \mathcal{S} \to \mathsf{R}^+$. The goal is to select a minimum cost subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that the union of the sets in $\mathcal{S}'$ is U. Many algorithms have been discovered for this problem that achieve an approximation ratio of $\ln n$ (see e.g. [Chv79, Joh74, Lov75, WS11]), and this is best possible unless $\mathsf{P} = \mathsf{NP}$ [Fei98, DS14].

In the ONLINESETCOVER variant, we impose the additional restriction that the algorithm does not know U initially, nor the contents of each $S \in \mathcal{S}$ (at the outset, the algorithm only sees m empty sets). Instead, an adversary reveals the elements of U one-by-one in an arbitrary order. On the arrival of every element, it is revealed which sets of $S \in \mathcal{S}$ contain the element, and the algorithm must immediately pick one such set $S \in \mathcal{S}$ to cover it. The goal is to minimize the total cost of the sets chosen by the algorithm. In their seminal work, [AAA+09] show that despite the lack of foresight, it is possible to achieve a competitive ratio of $O(\log m \log n)$ for this version[1]. The result is based on solving the LP relaxation via the online primal-dual framework, followed by randomized rounding. This bound has since been shown to be tight unless $\mathsf{NP} \subseteq \mathsf{BPP}$ [Kor04], and has also been generalized significantly (e.g. [AAA+06, BN09b, GN14]).

2.6 Online and Dynamic Models for SUBMODULARCOVER

In several chapters we study online and dynamic variants of SUBMODULARCOVER. In these problems, at every point in time t, the algorithm is given a different nonnegative, monotone, submodular function $f^{(t)}$, and must maintain a feasible and near-minimum cost solution to SUBMODULARCOVER for the current function $f^{(t)}$.

We introduce the *bag-of-functions* model for this setting. Here, we assume that the function $f^{(t)}$ takes the form

$$f^{(t)}(S) = \sum_{g \in G^{(t)}} g(S)$$

for some set $G^{(t)}$ of nonnegative, monotone, submodular functions. We refer to $G^{(t)}$ as the *active set* functions at time t. For ease of notation we assume that for all $g \in \bigcup_t G^{(t)}$, the parameter $g(\mathcal{N})$ is the same.

[1]Throughout this thesis, we consider the *unknown-instance model* for ONLINESETCOVER (see Chapter 1 of [Kor04]). The result of [AAA+09] was presented for the *known-instance model*, but extends to the unknown setting as well. This was made explicit in subsequent work [BN09b].

Part I

Online Algorithms

Chapter 3

Online Submodular Cover

3.1 Introduction

We consider the problem of maintaining a minimum cost submodular cover online. As detailed in Chapter 1, for many applications it does not suffice to solve a single static instance of SUBMODULARCOVER. Instead, the function f—the notion of coverage—changes over time, so we need to update our cover S. Recall our running example of the SIMULTANEOUSSOURCELOCATION problem [AGG+09]. Given a network equipped with flow demand at every vertex, we want to designate the minimum number of vertices as servers, such that every vertex can simultaneously route its demand from some server without violating edge capacities. [AGG+09] show that if $f(S)$ is the maximum demand satisfied by selecting S as the set of servers, then f is submodular and the problem can cleanly be solved using the SUBMODULARCOVER framework. In the likely scenario that communication demand evolves over time, revoking prior decisions may be expensive or prohibitive, hence we want that our solution be monotone—i.e., we only add servers to S and not remove any.

We ask the natural question: is it possible to maintain a competitive solution to an online SUBMODULARCOVER instance as f changes over time? We show that under minimal monotonicity assumptions on the changes in f, the answer is yes. Formally, the ONLINESUBMODULARCOVER problem is the following:

> We are given a "time-monotone"[1] sequence of monotone submodular functions $f^{(1)}, f^{(2)}, \cdots$, over a common universe N that arrive online, as well as a fixed cost function c. Upon seeing $f^{(t)}$ at time t, our algorithm must output a set S_t such that $f^{(t)}(S_t) = f^{(t)}(\mathcal{N})$. Moreover, past decisions cannot be revoked, so we require $S_{t-1} \subseteq S_t$.

One example of this time-monotone setting is the bag-of-functions model (recall the definition of this model in Section 2.6), with the restriction that at every time t, a single function $g^{(t)}$ is *inserted* into the active set. It may be convenient for the reader to keep this example in mind since it captures most features of the general problem. In particular, ONLINESUBMODULARCOVER in

[1]This "time-monotonicity" means that any submodular cover for $f^{(t)}$ is also a cover for $f^{(t-1)}$; formally, for all t and S, we have $f^{(t)}(S) = f^{(t)}(\mathcal{N}) \implies f^{(t-1)}(S) = f^{(t-1)}(\mathcal{N})$. In other words, the functions cannot be completely unrelated to each other. This is a minimal requirement; we show in Section 3.6.1 that dropping it makes the problem intractable. On the other hand, we do not explicitly require that $f^{(t)}(S) \geq f^{(t-1)}(S)$.

the bag-of-functions model generalizes OnlineSetCover; if we identify $\mathcal{N} = \mathcal{S}$ with the sets in the set system, then $g^{(t)}$ is the indicator of whether the t^{th} item has been covered by the chosen sets, and and $f^{(t)}$ is the coverage function induced by the set system restricted to the first t items.

Our main theorem is:

Theorem 3.1.1. *There exists an efficient randomized algorithm for the* OnlineSubmodularCover *problem which guarantees that for each T, the expected cost of solution S_T is within $O(\log m \cdot \log(T \cdot f_{\max}/f_{\min}))$ of the optimal* SubmodularCover *solution for $f^{(T)}$.*

Here we use the notation $f_{\max} := \max_t f^{(t)}_{\max}$ and $f_{\min} := \min_t f^{(t)}_{\min}$ (recall that $f^{(t)}_{\max}$ and $f^{(t)}_{\min}$ from Section 2.2 are smoothness parameters of the function sequence $f^{(1)}, f^{(2)}, \ldots$). When restricting to the case of OnlineSetCover, $f_{\max} \leq n$ and $f_{\min} = 1$ and hence this competitiveness guarantee recovers the $O(\log m \log n)$ of [AAA+09] (see Section 2.5). As a consequence, if we restrict ourselves to efficient algorithms, our result is the best possible up to constant factors unless $\mathsf{NP} \subseteq \mathsf{BPP}$.

Our algorithm also is interesting in the *offline* setting: indeed, if we are given a single submodular function f (i.e., $T = 1$), we can avoid the $O(\log m)$ loss that comes from solving an LP relaxation online. So we get an $O(\log(f_{\max}/f_{\min}))$-approximate solution via randomized rounding a fractional LP solution. This is the first LP-rounding algorithm known for SubmodularCover, and it matches the (optimal) performance of the greedy algorithm (analyzed by dual-fitting). LP rounding approaches are powerful and versatile, since we can add side constraints to the LP and maintain them (approximately) during rounding, so we hope that our techniques will find applications even in the offline setting.

3.1.1 Results and Techniques

There are a few challenges to proving Theorem 1.1. A natural approach is to mimic the strategy of [AAA+09] for the OnlineSetCover problem. I.e., we can try to maintain a solution to an LP relaxation of the problem, and then perform randomized rounding online. Unfortunately, the only known LP relaxation of SubmodularCover from [Wol82] is exponential-sized and it is not known how to solve it efficiently, nor how to analyze randomized rounding given a fractional solution. We show that we can overcome these concerns and make the technique work.

We start in Section 3.2 with an $O(\log m \log(T \cdot f(\mathcal{N})/f_{\min}))$ competitive ratio algorithm for online SubmodularCover with no efficient runtime guarantees. Here we use the notation $f(\mathcal{N}) := \max_t f^{(t)}(\mathcal{N})$. This allows us to illustrate the overall approach of solving the SubmodularCover covering linear program online, in conjunction with a randomized-rounding algorithm for it. The rounding is the natural one, where we sample from the fractional solution multiple times (in an online fashion); as opposed to the SetCover-style analysis which argues item-by-item, here we have to argue about the coverage as a whole. In particular, we exploit the relationship between the multilinear extension and the linearizations of submodular functions (Lemma 3.2.3) to reinterpret the constraints in the LP as statements about expected coverage under randomized rounding.

In Section 3.3, we improve the $O(\log m \log(T \cdot f(\mathcal{N})/f_{\min}))$-competitiveness to a finer competitiveness bound of $O(\log m \log(T \cdot f_{\max}/f_{\min}))$, which is much tighter for the case where the functions f are "smoother" and each element has smaller marginal value compared to $f(\mathcal{N})$. The intuition behind this refined analysis is that we charge each element selected by the algorithm to the elements of OPT that "cover the same part of the space".[2] To capture the overlap between two elements, we study the *mutual coverage* $\mathcal{I}_f(A;B) := f(A) + f(B) - f(A \cup B)$, which is natural generalization of mutual information from information theory, and inherits properties from that literature, such as the chain-rule. It turns out that mutual coverage is the right abstract quantity to focus on for our analyses. We also have to strengthen the SUBMODULARCOVER LP slightly, and use a modified rounding-with-alterations scheme to get the tighter result. When run offline, our rounding algorithm recovers the $O(\log f_{\max}/f_{\min})$ approximation ratio of Wolsey's algorithm.

In Section 3.4, we make the above algorithms run in polynomial time. The difficulty is two-fold: firstly, we do not know how to solve the SUBMODULARCOVER LP even in the offline setting, since the separation problem itself is APX-hard. Secondly, even if we could separate in the offline setting, naïvely solving the exponential-sized LP online requires us to give exponentially-many constraints one at a time to the online primal-dual LP-solving framework. To remedy this, we use a "round-or-separate" approach: we use the rounding algorithm itself as an approximate separation oracle. Indeed, we give an approximate separation oracle that given an arbitrary solution promises to either round it to a feasible integer solution with a similar objective value, or else to output a constraint violated by the solution. This approach to blur the line between rounding and separation may prove useful for other problems in online algorithms, where we are faced with an exponential number of new constraints at each step. Furthermore, we give a new exponential-clock based sampling procedure that finds constraints violated by a large margin with high probability. Our analysis extends a technique of [Von07] that approximates independent random sampling by a continuous process. We use a potential function argument to conclude that we only need a polynomial number of calls to the separation oracle in the worst case.

3.1.2 Related Work

There are previous definitions of online submodular cover, but similarity to our work is mainly in the name. In one variant inspired by regret-minimization in online learning [SG08, GB11], a submodular function $f^{(t)}$ arrives at each time step t, and we must output a permutation of the universe elements that minimizes the (average or maximum) cover time. The approach compares the algorithm's objective value to the single best *fixed* cover in hindsight. Hence, there is no notion of time-monotonicity, and of element selection being irrevocable. Another distantly related line of work ([GK11, DHK16, HK18, GHKL16, AAK19, EKM21, GGN21]) explores a stochastic variant of SUBMODULARCOVER, where there is a fixed submodular function f, but each element of the universe is active only with a certain probability. The goal is to output an ordering of the elements minimizing the expected cover time.

[GTW14] maintain a matroid base (or spanning set) under changing costs. Given a fixed matroid $\mathcal{M}$, each matroid element is given holding and and acquisition costs at each time t, after which we must output a matroid base (spanning set). We want to minimize the sum of acquisition plus holding costs over time. In the spanning set case, when holding costs are zero, acquisition costs are fixed and the matroid is additionally changing over time, this is the very special case of

[2]The analogue in the case of SETCOVER is an improvement from $O(\log \text{universe-size})$ to $O(\log \text{max-set-size})$. However, there is a clear notion in SETCOVER of *items* that are covered by the same two sets (and the analysis proceeds item-by-item), whereas in SUBMODULARCOVER it is not at all obvious what it means for the coverage of two elements to overlap.

our ONLINESUBMODULARCOVER problem where the submodular functions are matroid rank functions. Recall that matroid rank functions are a very restricted class of submodular functions: for instance, we can efficiently find a min-weight matroid base, while no efficient algorithms exist for finding a minimum weight submodular cover unless $\mathsf{P} = \mathsf{NP}$. [BCN14a] study the online matroid caching problem, where one must maintain an independent set in a matroid subject to the constraints that the set must contain certain elements at specified points in time.

3.2 An $O(\log m \log(T \cdot f(\mathcal{N})/f_{\min}))$ competitive algorithm

In this section, we show how to get our algorithm for ONLINESUBMODULARCOVER: we use the approach of solving covering LPs online to maintain a competitive fractional solution x_t at every time t, with the special property that $x_1, x_2, \ldots, x_T$ are monotonically increasing, i.e., $x_t \geq x_{t-1}$. To get an integer solution online, we use randomized rounding with the method of alterations: we round the increments in variables at each step, and make corrections in the case of failures by greedily selecting elements until our solution is feasible. Our algorithm always outputs a feasible solution; the challenge is to bound its expected cost: the crucial technical component is a relationship between the SUBMODULARCOVER LP and the multilinear relaxation, which allows us to show that a logarithmic number of rounds suffice.

3.2.1 An LP for SUBMODULARCOVER

We use Wolsey's LP formulation (2.4.1), but first we rewrite it using our continuous extension notation as:

$$\boxed{\begin{array}{ll} \min & \sum_j c(j)\, x_j \\ \text{subject to} & \\ & f^*(x) \geq f(\mathcal{N}) \\ \forall j \in \mathcal{N}: & x_j \geq 0 \end{array}} \tag{P}$$

Recall that f^* is the covering extension of f defined in Section 2.2.

Let $(P^{(t)})$ refer to the program (P) for $f^{(t)}$. We would like to argue that we can maintain one large LP for ONLINESUBMODULARCOVER. At time t, we feed in the constraints of $(P^{(t)})$, and update the solution accordingly. We use the following online LP solver from [BN09b, Theorem 4.2]:

Theorem 3.2.1. *For the setting where the rows of a covering LP* $\min\{c^\intercal x \mid Ax \geq b\}$ *with* $A \in \mathsf{R}_+^{m\times n}$, $b \in \mathsf{R}_+^m$, $c \in \mathsf{R}_+^n$ *arrive online, there is an algorithm that achieves a competitive ratio of* $O(\log m)$. *Furthermore, the sequence of solutions produced* $x_1, \ldots, x_T$ *is monotonically increasing.*

Each of the linear programs $(P^{(t)})$ is indeed a covering LP with box constraints that is a feasible relaxation for the function $f^{(t)}$. However it is not *a priori* clear that the constraints for the linear program $(P^{(t')})$ for some $t' < t$ are also valid for the linear program $(P^{(t)})$. We prove a lemma which circumvents this issue. While we do not guarantee that for all $t' < t$ the constraints of $(P^{(t')})$ will be valid for the LP $(P^{(t)})$ itself, we show that they are valid for the *integer* solutions of $(P^{(t)})$. This suffices to ensure that the union of all constraints seen so far is a relaxation for ONLINESUBMODULARCOVER.

Lemma 3.2.2. *Any feasible integer solution to $(P^{(t+1)})$ is a feasible integer solution to $(P^{(t)})$.*

Proof. Suppose x is a feasible integer solution to $(P^{(t+1)})$. Then by Claim 5.3.2, x is the characteristic vector of a set S such that $f^{(t+1)}(S) = f^{(t+1)}(\mathcal{N})$. By the time-monotonicity of the sequence $f^{(0)}, f^{(1)}, \dots, f^{(T)}$, it holds that $f^{(t)}(S) = f^{(t)}(\mathcal{N})$ as well. Now again using Claim 5.3.2, x is feasible to $(P^{(t)})$. □

3.2.2 Rounding Fractional Solutions

The last remaining step in our plan is to understand the behavior of randomized rounding for (P). Since the multilinear extension G is defined as the value of g on a set obtained by randomized rounding, we can use the following relationship between G and g^* [Von07, Lemma 3.8]:

Lemma 3.2.3. *For any monotone, nonnegative set function g, it holds that $G(x) \geq (1-e^{-1})g^*(x)$.*

Observe that Lemma 3.2.3 does not require that g be submodular, only that it be monotone and nonnegative. (There exist functions for which the bound is tight. If g is also submodular, we also get the bound $G(x) \leq g^*(x)$ [Von07, Lemma 3.7].) The next lemma helps us analyze random rounding:

Lemma 3.2.4 (Rounding Lemma)**.** *Let g be a monotone, nonnegative set function such that $g(\emptyset) = 0$, and let k be a positive integer. Let $R_1, \dots R_k$ be a sequence of random sets and denote $R_{1:\ell} := \bigcup_{i\in[\ell]} R_i$. Suppose the following condition holds for all $\ell \in [k]$ and all $R \subseteq N$:*

$$\mathop{\mathbb{E}}_{R_\ell}[g(R_\ell \mid R_{1:\ell-1}) \mid R_{1:\ell-1} = R] \geq (1-\gamma)g(\mathcal{N} \mid R). \tag{3.2.1}$$

Then:

$$\mathop{\mathbb{E}}_{R_{1:k}}[g(\mathcal{N} \mid R_{1:k})] \leq \gamma^k \cdot g(\mathcal{N}). \tag{3.2.2}$$

Proof. By definition:

$$\begin{aligned}\mathop{\mathbb{E}}_{R_{1:\ell}}[g(\mathcal{N} \mid R_{1:\ell})] &= \mathop{\mathbb{E}}_{R_{1:\ell-1}}[g(\mathcal{N}) - g(R_{1:\ell-1}) - \mathop{\mathbb{E}}_{R_\ell}[g(R_{1:\ell}) - g(R_{1:\ell-1})]] \\ &\leq \mathop{\mathbb{E}}_{R_{1:\ell-1}}[g(\mathcal{N} \mid R_{1:\ell-1}) - (1-\gamma)(g(\mathcal{N} \mid R_{1:\ell-1}))] \qquad (3.2.3) \\ &= \gamma \cdot \mathop{\mathbb{E}}_{R_{1:\ell-1}}[g(\mathcal{N}) - g(R_{1:\ell-1})].\end{aligned}$$

Note that (3.2.3) holds by assumption (3.2.1). Claim (3.2.2) holds by induction on ℓ and the fact that $g(\emptyset) = 0$. □

Lemma 3.2.5. *Let $x \in [0,1]^n$ be a feasible solution to (P). Let R be a set obtained by performing k rounds of randomized rounding according to x. Then: $\mathbb{E}_R[f_R(\mathcal{N})] \leq e^{-k} f(\mathcal{N})$.*

Proof. Since x is feasible to (P), for all $S \subseteq N$, we have $f_S^*(x) \geq f_S(\mathcal{N})$. By Lemma 3.2.3 applied to f_S, it holds that $F_S(x) \geq (1-e^{-1}) f_S^*(x)$. Together these imply:

$$F_S(x) \geq (1-e^{-1}) f_S(\mathcal{N}). \tag{3.2.4}$$

Thus x, f and R together satisfy the conditions of the Rounding Lemma with $\gamma = e^{-1}$, and the claim follows directly. □

3.2.3 The Analysis

To summarize, our first algorithm for OnlineSubmodularCover is the following: maintain a fractional solution x, and two sets $R, G \subseteq N$ initially empty. At time t, feed the batch of constraints of $(P^{(t)})$ to [BN09b] and update x accordingly. Set $k := \log(t^2 \cdot f^{(t)}(\mathcal{N})/f_{\min}^{(t)})$, and for every $j \in N$, add j to R with probability Δ_j, where Δ_j is calculated such that the total probability that j gets added to R over the course of the algorithm is $1-(1-x_j)^k$. In a subsequent greedy phase, iteratively select the cheapest element $j \in N$ with non-zero marginal value and add it to the set G until no such elements remain. Return $R \cup G$.

Given our tools from the previous section, we can now analyze the expected cost.

Theorem 3.2.6. *There exists a randomized algorithm for the* OnlineSubmodularCover *problem which guarantees that for each T, the expected cost of solution S_T is within $O(\log m \cdot \log(T \cdot f(\mathcal{N})/f_{\min}))$ of the optimal* SubmodularCover *solution for $f^{(T)}$.*

Proof. By construction, the algorithm always produces a feasible solution to the submodular cover problem. Thus it suffices to relate the cost of the output to the cost of the optimal solution, $c(\text{Opt})$.

To bound the cost of the sampling phase, remark that at time t, by construction we can view R as the result of k rounds of independent random rounding. The expected cost after one round of rounding is precisely $c(x)$. By linearity of expectation, the total expected cost of R is $\log\left(t^2 \cdot f^{(t)}(\mathcal{N})/f_{\min}^{(t)}\right) \cdot c(x)$.

Next we bound the cost of the greedy phase. At time t, the solution x is feasible to $(P^{(t)})$. By Lemma 3.2.5 with parameter $k = \log\left(t^2 \cdot f^{(t)}(\mathcal{N})/f_{\min}^{(t)}\right)$:

$$\mathop{\mathbb{E}}_R[f_R^{(t)}(\mathcal{N})] \leq t^{-2}\left(\frac{f_{\min}^{(t)}}{f^{(t)}(\mathcal{N})}\right) f^{(t)}(\mathcal{N}) = t^{-2} f_{\min}^{(t)}.$$

This gives a bound on the expected cost of alterations at time t. Let G_t denote the elements added in the greedy phase at time t:

$$\begin{aligned} \mathop{\mathbb{E}}_R[c(G_t)] &\leq c(\text{Opt}) \cdot \mathop{\mathbb{E}}_R[|G_t|] && (3.2.5)\\ &\leq t^{-2} c(\text{Opt}). && (3.2.6) \end{aligned}$$

(3.2.5) comes from the fact that each element must be cheaper than $c(\text{Opt})$, the second step (3.2.6) from the fact that for every element $h \in G$, both $f_R^{(t)}(h) \geq f_{\min}^{(t)}$ and $\mathbb{E}_R[f_R^{(t)}(\mathcal{N})] \leq t^{-2} f_{\min}^{(t)}$. By linearity of expectation, the expected cost of cumulative alterations over all time steps $t \in [T]$ is

$O(c(\textsc{Opt}))$. In conclusion, the final solution has expected cost at most $\log(T \cdot f(\mathcal{N})/f_{\min})\, c(x) + O(c(\textsc{Opt}))$. By Theorem 2.1, $c(x) = O(\log m) \cdot c(\textsc{Opt})$, hence the algorithm outputs a solution with competitive ratio $O(\log m \log(T \cdot f(\mathcal{N})/f_{\min})) \cdot c(\textsc{Opt})$. □

In the next section, we show how to replace the $f(\mathcal{N})$ term in the competitiveness by an $f_{\max}$ term, which is often much smaller. In order to do the finer analysis, we will also show how to use the ideas of mutual coverage. We then make these algorithms efficient in the subsequent section.

3.3 An $O(\log m \log(T \cdot f_{\max}/f_{\min}))$ competitive algorithm

In this section, we show how to recover the finer $O(\log m \log(T \cdot f_{\max}/f_{\min}))$ approximation. The main algorithmic change is to perform $\log\left(T \cdot f^{(t)}_{\max}/f^{(t)}_{\min}\right)$ rounds of rounding instead of $\log\left(T \cdot f^{(t)}(\mathcal{N})/f^{(t)}_{\min}\right)$, and then compensate with more greedy alterations. The intuition behind the analysis is a fractional charging scheme. Using mutual coverage as a notion of contribution, we argue that:

(i) every greedily chosen element is cheaper than the elements in $\textsc{Opt}$ to which it contributes.

(ii) every greedily chosen element contributes at least a minimum amount overall.

(iii) for every element in $\textsc{Opt}$, the total contribution it receives from greedily chosen elements is bounded above.

We show that these statements together imply a good bound on the expected cost of alterations. However, this proof strategy requires that the mutual coverage between pairs of elements decrease sufficiently after randomized rounding. In Section 3.6.2, we show that the constraints of the LP (P) do not necessarily guarantee that this is the case; hence we introduce additional constraints that enforce the desired property.

3.3.1 A Stronger Formulation

Recall our definition of Mutual Coverage, $\mathcal{I}_f(A; B) = f(A) + f(B) - f(A \cup B)$. For clarity, from now on we drop the subscript f when the submodular function in question is clear from context. We begin by strengthening LP (P) with additional constraints as follows:

$$\boxed{\begin{array}{lll} \min & \displaystyle\sum_j c(j)\, x_j & \\ \text{subject to} & & \\ \forall v \in \mathcal{N}, \forall S \subseteq N: & \displaystyle\sum_{j \in \mathcal{N}} \mathcal{I}^{(v|S)}(j) x_j \geq \mathcal{I}^{(v|S)}(\mathcal{N}) & (*) \\ \forall j \in \mathcal{N}: & 1 \geq x_j \geq 0 & \end{array}} \qquad (Q)$$

To see that the new constraints imply the constraints of (P), consider any set S. Order the elements of N arbitrarily $v_1, v_2 \ldots v_n$ and define $S_i := S \cup \{v_1, \ldots v_{i-1}\}$ with $S_1 := S$. If we sum the ($*$)

constraints corresponding to all pairs (v_i, S_i), we obtain:

$$\begin{aligned}
\sum_{i\in[n]}\sum_{j\in N}\mathcal{I}(v_i; j \mid S_i)\cdot x_j &\geq \sum_{i\in[n]}\mathcal{I}(v_i; N \mid S_i) \\
\Rightarrow \sum_{j\in N}\mathcal{I}(\mathcal{N}; j \mid S)\cdot x_j &\geq \mathcal{I}(\mathcal{N}; N \mid S) \qquad (3.3.1)\\
\Rightarrow \sum_{j\in N} f_S(j)\cdot x_j &\geq f_S(\mathcal{N}).
\end{aligned}$$

where we used the Chain Rule for mutual coverage for step (3.3.1).

In order to treat our problem as an online covering LP and use [BN09b] like we did before, we again need to make sure the new constraints do not eliminate any integer solutions.

Lemma 3.3.1. (Q) *is a relaxation of* SUBMODULARCOVER.

Proof. Consider a feasible solution T to SUBMODULARCOVER and its corresponding characteristic vector χ_T. Then for any $S \subseteq N$ and $v \in \mathcal{N}\backslash S$:

$$\sum_{j\in\mathcal{N}}\mathcal{I}(v, j \mid S)(\chi_T)_j = \sum_{j\in T} f_S(j) + f_S(v) - f_S(v, j) \geq f_S(v) = \mathcal{I}(v, N \mid S).$$

The inequality follows from the fact that $f_S(A) + f_S(B) - f_S(A\cup B) \geq f_S(A\cap B)$ and $f(\emptyset) = 0$. □

The following lemma is an analog of Lemma 3.2.5 for (Q), and will enable us to perform rounding.

Lemma 3.3.2. *Let* $x \in [0,1]^n$ *be a feasible solution to* (Q)*. Let* R *be a set obtained by performing* k *rounds of randomized rounding according to* x*. Then:*

$$\mathop{\mathbb{E}}_{R\sim x}[\mathcal{I}(v; N \mid R)] \leq e^{-k}\mathcal{I}(v; N). \qquad (3.3.2)$$

Proof. We apply Lemma 3.2.3 to the function $\mathcal{I}^{(v|S)}$. Even though $\mathcal{I}^{(v|S)}$ is not necessarily submodular, it is monotone and nonnegative (because f is submodular). Thus:

$$\begin{aligned}
&\mathop{\mathbb{E}}_{R\sim x}[\mathcal{I}(v; R \mid S)] \\
&= \mathop{\mathbb{E}}_{R\sim x}[\mathcal{I}^{(v|S)}(R)] \\
&\geq (1-e^{-1})\ \min_T\left(\mathcal{I}^{(v|S)}(T) + \sum_{j\in\mathcal{N}}\mathcal{I}^{(v|S\cup T)}(j)\cdot x_j\right) && (3.3.3)\\
&\geq (1-e^{-1})\ \min_T\left(\mathcal{I}^{(v|S)}(T) + \mathcal{I}^{(v|S\cup T)}(\mathcal{N})\right) && (3.3.4)\\
&= (1-e^{-1})\,\mathcal{I}(v; N \mid S). && (3.3.5)
\end{aligned}$$

Above, Step 3.3.3 uses Lemma 3.2.3. Step 3.3.4 follows from the ($*$) constraints since x is feasible to (Q), and step (3.3.5) comes from expanding the definition of mutual coverage. This means that x, $\mathcal{I}^{(v|S)}$, and R satisfy the conditions of the Rounding Lemma with $\gamma = e^{-1}$, so (3.3.2) follows directly. □

The proof above demonstrates the purpose of the new ($*$) constraints: they ensure that, just like $f_R(\mathcal{N})$, the expected remaining mutual coverage decreases geometrically with the number of rounds.

3.3.2 The Improved Analysis

Our second algorithm for OnlineSubmodularCover resembles the first; it is the analysis that differs. Indeed, at time t, feed the batch of constraints of $(Q^{(t)})$ (defined as (Q) for the function $f^{(t)}$) to [BN09b] and update the fractional solution x accordingly. In the rounding phase, perform $k = \log(t^2 \cdot f^{(t)}_{\max}/f^{(t)}_{\min})$ rounds of randomized rounding using the appropriate probability vector $\Delta \in [0,1]^n$ and add the new elements to the set R. In a subsequent greedy phase, iteratively add the cheapest element $j \in \mathcal{N}$ with non-zero marginal value to set G, until none remain. Return $R \cup G$.

It is clear that the algorithm is correct, so it remains to bound the expected cost. The main idea is to keep track of the amount of coverage that the t^{th} greedy element in the augmentation procedure contributes to the coverage achieved by the s^{th} element of the optimal solution; the (conditional) mutual coverage between the two is the quantity of interest here.

Theorem 3.3.3. *There exists a randomized algorithm for the* OnlineSubmodularCover *problem which guarantees that for each T, the expected cost of solution S_T is within $O(\log m \cdot \log(T \cdot f_{\max}/f_{\min}))$ of the optimal* SubmodularCover *solution for $f^{(T)}$.*

Proof. The algorithm always outputs a feasible solution, and it suffices to relate the cost of $R \cup G$ to the cost of the optimal solution. As before, at every time $t \in [T]$, the probability that j is in R is $1-(1-x_j)^k$ and we may view R as the result of k rounds of randomized rounding. Thus the expected cost of R is $\log(t^2 \cdot f^{(t)}_{\max}/f^{(t)}_{\min}) \cdot c(x)$. In the remainder, we analyze the greedy phase.

At time t, the fractional solution x is feasible to $Q^{(t)}$. By Lemma 3.3.2 with $k = \log(t^2 \cdot f^{(t)}_{\max}/f^{(t)}_{\min})$:

$$\mathop{\mathbb{E}}_{R}[\mathcal{I}(v; N \mid R)] \le t^{-2}\left(\frac{f^{(t)}_{\min}}{f^{(t)}_{\max}}\right)\mathcal{I}(v;\mathcal{N}) \tag{3.3.6}$$

$$= t^{-2}\left(\frac{f^{(t)}_{\min}}{f^{(t)}_{\max}}\right) f^{(t)}(v). \tag{3.3.7}$$

Let $G_1, \dots, G_p$ denote the elements added by the greedy phase in increasing order of cost, and let $G_{1:t} = \{G_1, \dots, G_t\}$ be the prefix of the first t elements. Also order the elements of Opt arbitrarily $O_1, \dots, O_q$ and similarly let $O_{1:s} = \{O_1, \dots, O_s\}$. For convenience, let $G_{1:0} = O_{1:0} = \emptyset$.

We say an element G_t *contributes* to O_s if conditioned on having selected $G_{1:t-1}$, and $O_{1:s-1}$, selecting G_t changes the marginal value of O_s. Let $C(t,s)$ be the contribution of G_t to O_s, or the amount by which this marginal value changes. Formally:

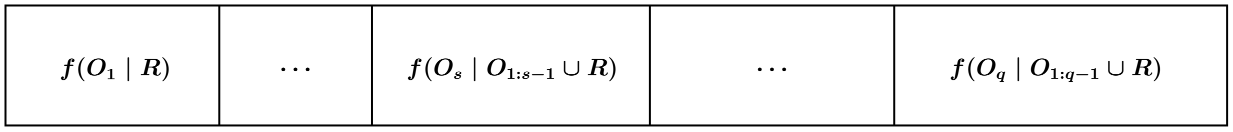
$$C(t,s) := \mathcal{I}(G_t; O_s \mid G_{1:t-1} \cup O_{1:s-1} \cup R).$$

We will need the three claims we alluded to in the opening of the section to proceed.

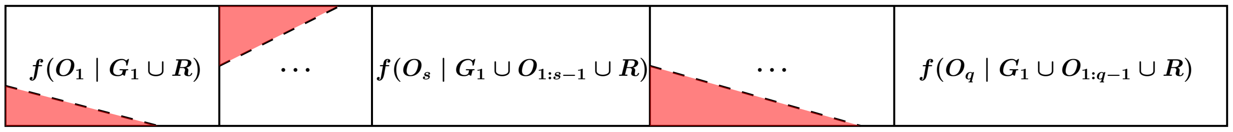
Claim 3.3.4. *If G_t contributes to O_s, i.e. $C(t,s) > 0$, then $c(G_t) \le c(O_s)$.*

Claim 3.3.5. *For any index $t \in [p]$ we have: $\sum_s C(t,s) \ge f^{(t)}_{\min}$.*

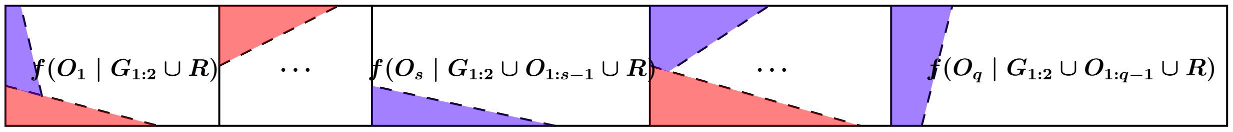
Claim 3.3.6. *For any index $s \in [q]$ we have: $\mathbb{E}_R[\sum_t C(t,s)] \le t^{-2} f^{(t)}_{\min}$.*


$f(O_1 \mid R)$
$\cdots$
$f(O_s \mid O_{1:s-1} \cup R)$
$\cdots$
$f(O_q \mid O_{1:q-1} \cup R)$

(a) Before the greedy algorithm


$f(O_1 \mid G_1 \cup R)$
$\cdots$
$f(O_s \mid G_1 \cup O_{1:s-1} \cup R)$
$\cdots$
$f(O_q \mid G_1 \cup O_{1:q-1} \cup R)$

(b) After G_1 is selected


$f(O_1 \mid G_{1:2} \cup R)$
$\cdots$
$f(O_s \mid G_{1:2} \cup O_{1:s-1} \cup R)$
$\cdots$
$f(O_q \mid G_{1:2} \cup O_{1:q-1} \cup R)$

(c) After G_2 is selected

Figure 3.1: Illustration of the charging scheme. The white areas initially represent the marginal values of each element of OPT conditioned on the elements that come before it in the ordering. The red areas represent the contribution of G_1 to the elements of OPT, or the amount by which the marginal value of each element in OPT changes after conditioning on G_1. Likewise, the purple areas represent the contribution of G_2 to the elements of OPT.

Before we prove these claims, let us show how they imply the statement of the theorem.

$$\begin{aligned}
\mathbb{E}\left[\sum_t c(G_t)\right] &= \mathbb{E}\left[\sum_t \left(\sum_s \frac{C(t,s)}{\sum_{s'} C(t,s')}\right) c(G_t)\right] \\
&\le \mathbb{E}\left[\sum_s \left(\sum_t \frac{C(t,s)}{\sum_{s'} C(t,s')}\right) c(O_s)\right] && (3.3.8) \\
&\le \sum_s \frac{\mathbb{E}[\sum_t C(t,s)]}{f_{\min}^{(t)}} \cdot c(O_s) && (3.3.9) \\
&\le t^{-2} c(\textsc{Opt}). && (3.3.10)
\end{aligned}$$

Above (3.3.8) is due to Claim 3.3.4, (3.3.9) due to Claim 3.3.5 and (3.3.10) due to Claim 3.3.6. Thus the expected cost of cumulative alterations over all time steps $t \in [T]$ is $O(c(\textsc{Opt}))$. To wrap up, Theorem 2.1 implies that $c(x) = O(\log m) \cdot c(\textsc{Opt})$, hence:

$$\begin{aligned}
\mathbb{E}[c(R \cup G)] &\le \log\left(T \cdot \frac{f_{\max}^{(t)}}{f_{\min}^{(t)}}\right) \cdot c(x) + O(c(\textsc{Opt})) \\
&\le O\left(\log m \log\left(T \cdot \frac{f_{\max}^{(t)}}{f_{\min}^{(t)}}\right)\right) \cdot c(\textsc{Opt}).
\end{aligned}$$

It remains to prove the three claims.

Proof of Claim 3.3.4. The greedy algorithm repeatedly selects the cheapest element that has nonzero marginal value. If for some element O_s it is the case that $c(O_s) < c(G_t)$, then by the time G_t is picked, O_s must already have been selected, which means $C(t,s) = 0$. □

Proof of Claim 3.3.5. This is immediate from the definitions:

$$\begin{aligned}\sum_{s\in[q]} C(t,s) &= \sum_{s\in[q]} \mathcal{I}(G_t; O_s \mid G_{1:t-1} \cup O_{1:s-1} \cup R) \\ &= \mathcal{I}(G_t; N \mid G_{1:t-1} \cup R) \qquad (3.3.11) \\ &= f^{(t)}_{G_{1:t-1}\cup R}(G_t) \\ &\geq f^{(t)}_{\min}, \qquad (3.3.12)\end{aligned}$$

where step (3.3.11) used the Chain Rule, and the final step (3.3.12) used the definition of $f^{(t)}_{\min}$. □

Proof of Claim 3.3.6. Again by definition:

$$\begin{aligned}\sum_{t} C(t,s) &= \sum_{t\in[p]} \mathcal{I}(G_t; O_s \mid G_{1:t-1} \cup O_{1:s-1} \cup R) \\ &= \mathcal{I}(\mathcal{N}; O_s \mid O_{1:s-1} \cup R) \\ &\leq \mathcal{I}(\mathcal{N}; O_s \mid R),\end{aligned}$$

where the inequality follows from the submodularity of f and $\mathcal{I}(\mathcal{N}; O_s \mid O_{1:s-1} \cup R) = f(O_s \mid O_{1:s-1} \cup R)$. Taking expectation over R:

$$\begin{aligned}\mathop{\mathbb{E}}_{R}\left[\sum_t C(t,s)\right] &\leq \mathop{\mathbb{E}}_{R}[\mathcal{I}(O_s; N \mid R)] \\ &\leq t^{-2}\left(\frac{f^{(t)}_{\min}}{f^{(t)}_{\max}}\right) f(O_s) \\ &\leq t^{-2} f^{(t)}_{\min}.\end{aligned}$$

The second inequality follows from (3.3.7), and the third from the definition of $f^{(t)}_{\max}$. □

Given the claims, the proof of Theorem 3.1 is complete. □

It remains to show that the algorithm of this section and the previous one can be made efficient; this we do in the following section.

3.4 Polynomial Runtime

The algorithms in the two previous sections naïvely run in exponential time, and here are some of the barriers to making them efficient. For one, the algorithm of [BN09b] runs in time linear in the number of constraints added, and naively both $(P^{(t)})$ and $(Q^{(t)})$ add an exponential number of constraints at every step. The standard method for solving LPs with many constraints is to construct an efficient separation oracle that, given a point as input, either confirms that it is a feasible solution or outputs a violated constraint. Given such an oracle, we could check if our current solution is feasible, if so output the solution, if not feed the violated constraint to [BN09b] and repeat.

However, a polynomial-time separation oracle is unlikely to exist since evaluating f^* is APX-hard [Von07, Section 3.7.2]. Moreover, even assuming access to a separation oracle, we would additionally need to argue that we can make do with a polynomial number of calls to the oracle. However, the algorithm of [BN09b] makes no a priori guarantees about how many constraints it will need to fix before it produces a feasible solution (in contrast to the ellipsoid algorithm).

Our solution is to avoid solving the fractional LP directly. Since we merely want to find a feasible integer solution with bounded expected cost, we combine rounding and separation. We show that every LP solution either guarantees random rounding will make progress, or violates an efficiently computable constraint by a large margin. Finally, we show that fixing the large-margin violations requires a large change to the LP solution; by a potential function argument this can only be done a polynomial number of times before the solution is necessarily feasible.

3.4.1 Finding Violated Constraints

Lemma 3.2.3 implies that for monotone, nonnegative functions g, the multilinear extension G (which is precisely the expected coverage of a set obtained by randomized rounding according to x) cannot be smaller than $(1-e^{-1})g^*$. This means that if $G(x)$ is much smaller than $g(N)$, then so is $g^*(x)$, and there must exist C such that $g^{\#C}(x) < g(N)$. Call such a set C *violated*, since we will soon use this as a mechanism for finding violated constraints of (Q). Lemma 3.2.3 does not immediately imply a constructive algorithm for finding a set C witnessing the violation. We show that the following procedure, inspired by the proof of Lemma 3.2.3 (Lemma 3.8 of [Von07]), allows us find such a C.

The input is a fractional LP solution x. For every element $i \in N$, sample an exponential random variable h_i with parameter x_i, and order all the elements $i_1, \ldots i_n$ in increasing order of their corresponding expontential h_i. Of all prefixes of this ordering, consider the prefix $Q = i_1, \ldots i_z$ that minimizes $g^{\#Q}(x)$. If $g^{\#Q}(x) < g(N)$, then say that Q is a violated set. Repeat these steps $3/\epsilon \cdot \log(1/\delta)$ times or until a violated set is found. If none is found, return "no constraint found".

Lemma 3.4.1. *There is an algorithm such that given a nonnegative, monotone function $g : 2^N \to \mathbb{R}^+$ with $g(\emptyset) = 0$, and $x \in [0,1]^n$ with the guarantee that $G(x) < (e(1+\epsilon))^{-1} g(N)$, with probability $1-\delta$ the algorithm yields a subset $C \subseteq N$ such that:*

$$g^{\#C}(x) < \left(1 - \frac{\epsilon}{2}\right) \cdot g(N).$$

Furthermore, the procedure runs in time $\mathrm{poly}\left(\frac{1}{\epsilon}\log\frac{1}{\delta}\right)$.

Proof. Consider the set $C(t) := \{i \in N \ : \ h_i < t\}$. Since $\mathbb{P}\left(j \in C(1)\right) = 1 - e^{-x_j} \le x_j$, we have by monotonicity of g that $G(x) = \mathbb{E}_{S\sim x}[g(S)] \ge \mathbb{E}[g(C(1)]$. Now for any C:

$$\begin{aligned}
&\mathbb{E}[g_{C(t)}(C(t+dt)) \mid C(t) = C] \\
&= \sum_{T \subseteq N} \mathbb{P}\left(C(t+dt) = T \mid C(t) = C\right) \cdot g(T) \\
&= \sum_{T \subseteq N \setminus C} g_C(T) \cdot \prod_{j \in T} (x_j \, dt) \cdot \prod_{j \notin T} (1 - x_j \, dt) \\
&= \sum_{j \in N} g_C(j) x_j dt + O(dt^2) \\
&= (g^{\#C}(x) - g(C))dt + O(dt^2).
\end{aligned}$$

Dividing by dt and taking an expectation over C:

$$\begin{aligned}&\frac{1}{dt}\,\mathbb{E}[g(C(t+dt))-g(C(t))]\\&\geq \mathbb{E}[g^{\#C(t)}(x)]-\mathbb{E}[g(C(t))]+O(dt).\end{aligned}$$

Now define $\phi(t)=\mathbb{E}[g(C(t))]$. Taking the limit of the last expression as $dt\to 0$ we get that $\frac{d\phi}{dt}=\mathbb{E}[g^{\#C(t)}(x)]-\phi(t)$. Using e^t as an integrating factor, for any $t\geq 0$:

$$e^t\phi(t)=\int_0^t\frac{d(e^z\phi(z))}{dz}dz=\int_0^t e^z\,\mathbb{E}\left[g^{\#C(t)}(x)\right]\,dz.$$

By the mean value theorem, for some value $t^*\in[0,t]$ it holds that

$$e^t\phi(t)=te^{t^*}\,\mathbb{E}\left[g^{\#C(t^*)}(x)\right]\geq t\,\mathbb{E}\left[g^{\#C(t^*)}(x)\right].$$

Now setting $t=1$ and using the fact that $G(x)\geq\phi(1)$, we have $\mathbb{E}[g^{\#C(t^*)}(x)]\leq e\cdot G(x)$. By construction, $Q=C(t')$ for some $t'=\operatorname{argmin}_t g^{\#C(t)}(x)$. The inequality $g^{\#Q}(x)\leq g^{\#C(t^*)}(x)$ holds regardless of the realizations of $h_1,\ldots,h_n$, hence:

$$\mathbb{E}[g^{\#Q}(x)]\leq\mathbb{E}[g^{\#C(t^*)}(x)]\leq e\cdot G(x).$$

By a Markov bound, with probability at least $1-(1+\epsilon/2)^{-1}\geq\epsilon/3$:

$$\begin{aligned}g^{\#Q}(x)&\leq\left(1+\frac{\epsilon}{2}\right)e\cdot G(x)\\&<\frac{\left(1+\frac{\epsilon}{2}\right)e}{(1+\epsilon)e}\cdot g(N)\\&\leq\left(1-\frac{\epsilon}{2}\right)\cdot g(N).\end{aligned}$$

Since we repeat this procedure $k=(3/\epsilon)\log 1/\delta$ times, the probability that after k attempts no violated constraint is found is: $(1-\epsilon/3)^k\leq\exp\left(3/\epsilon\cdot\epsilon/3\log\delta\right)=\delta$. □

3.4.2 Implementing the Algorithm Efficiently

We modify the proof of Theorem 3.1 to exploit the approximate separation oracle. This time we do not ever solve (Q) explicitly. Instead, start with a fractional solution $x=0$ and just start rounding. Before every round, for every $v\in N$, search for violated constraints using the procedure from the last section, with $g=\mathcal{I}^{(v|R)}$ and $\delta=6\cdot c_{\min}/(\pi^2knt^2\cdot c(N))$. If one is found, feed it to [BN09b], update x accordingly and restart the current round. In total, perform $k:=(\log 1/\gamma)^{-1}\log(t^2\cdot f^{(t)}_{\max}/f^{(t)}_{\min})$ rounds of rounding with respect to the appropriate $\Delta\in[0,1]^n$ (where $\gamma:=1-(e(1+\epsilon))^{-1}$) and add the result to set R. To finish, again greedily add to set G the cheapest element $j\in N$ with non-zero marginal value, until none remain. Return $R\cup G$.

Crucially, we argue that we do not need to fix too many constraints over the course of the algorithm.

Lemma 3.4.2. *The algorithm above finds a violated constraint at most $O(n/\epsilon)$ times.*

Proof. Consider a round in which the algorithm finds a violated constraint indexed by the set C. By Lemma 3.4.1:

$$\mathcal{I}^{(v|R)}(C) + \sum_{j \in N} \mathcal{I}^{(v|R \cup C)}(j) \cdot x_j < \left(1 - \frac{\epsilon}{2}\right) \cdot \mathcal{I}^{(v|R)}(N).$$

In order to fix this constraint in this round, [BN09b] must increase x by a vector Δx such that:

$$\sum_{j \in N} \frac{\mathcal{I}^{(v|R \cup C)}(j)}{\mathcal{I}^{(v|R)}(N)} \cdot (\Delta x)_j > \frac{\epsilon}{2}.$$

Since $\mathcal{I}^{(v|R \cup C)}(j)/\mathcal{I}^{(v|R)}(N) \leq 1$, it follows that:

$$\sum_{j \in N} (\Delta x)_j \geq \sum_{j \in N} \frac{\mathcal{I}^{(v|R \cup C)}(j)}{\mathcal{I}^{(v|R)}(N)} (\Delta x)_j > \frac{\epsilon}{2}.$$

Since $x \leq 1$, the total sum $\sum_j x_j$ is bounded by n, and thus we can update x at most $2n/\epsilon$ times. □

We can finally conclude with our main theorem.

Theorem 3.1.1. *There exists an efficient randomized algorithm for the* ONLINESUBMODULARCOVER *problem which guarantees that for each T, the expected cost of solution S_T is within $O(\log m \cdot \log(T \cdot f_{\max}/f_{\min}))$ of the optimal* SUBMODULARCOVER *solution for $f^{(T)}$.*

Proof. The algorithm always produces a feasible solution and we need to bound its cost. Since we perform $(\log 1/\gamma)^{-1} \log(t^2 \cdot f^{(t)}_{\max}/f^{(t)}_{\min})$ rounds of rounding, and $(\log 1/\gamma)^{-1} \leq e(1+\epsilon)$, the expected cost of R is at most $e(1+\epsilon) \cdot \log(t^2 \cdot f^{(t)}_{\max}/f^{(t)}_{\min}) \cdot c(x)$. Next we bound the cost of the greedy phase.

Let $R_{1:\ell}$ denote the state of R at the end of round $\ell \in [k]$ and let R_ℓ denote the set of elements sampled in round ℓ. By the contrapositive of Lemma 3.4.1, for every $v \in N$ and every $\ell \in [k]$, with probability $1 - 6 \cdot c_{\min}/(\pi^2 knt^2 \cdot c(N))$ it holds that:

$$\mathop{\mathbb{E}}_{R_\ell \sim x}[\mathcal{I}(v; R_\ell \mid R_{1:\ell-1})] \geq \frac{1}{e(1+\epsilon)} \mathcal{I}(v; N \mid R_{1:\ell-1}). \tag{3.4.1}$$

Let $\mathcal{E}$ be the event that (3.4.1) holds for every round of rounding over all time steps $t \in [T]$, and all elements $v \in N$. By a union bound and the fact that $\sum t^{-2} \leq \pi^2/6$, event $\mathcal{E}$ holds with probability at least $1 - c_{\min}/(c(N))$. Conditioned on this being the case, the triple x, $\mathcal{I}^{(v)}$ and $R_{1:k}$ together satisfies the conditions of the Rounding Lemma with $\gamma = 1 - (e(1-\epsilon))^{-1}$. Hence:

$$\begin{aligned}\mathop{\mathbb{E}}_{R}[\mathcal{I}(v; N \mid R)] &\leq t^{-2} \left(\frac{f^{(t)}_{\min}}{f^{(t)}_{\max}}\right) \mathcal{I}(v; N) \\ &= t^{-2} \left(\frac{f^{(t)}_{\min}}{f^{(t)}_{\max}}\right) f^{(t)}(v).\end{aligned}$$

With this, Claims 3.3.4 to 3.3.6 all hold and we again have $\mathbb{E}[\sum_t c(G_t)] \leq t^{-2} c(\textsc{Opt})$ as in (3.3.10).

We conclude that if $\mathcal{E}$ holds, then by linearity of expectation, the expected cost of cumulative alterations over all time steps $t \in [T]$ is $O(c(\textsc{Opt}))$. If $\mathcal{E}$ does not hold, we incur a cost of at most $c(N)$. Hence the total expected cost of cumulative alterations is:

$$\begin{aligned}\mathbb{E}[c(G)] &= \mathbb{E}[c(G) \mid \mathcal{E}] \cdot \mathbb{P}(\mathcal{E}) + \mathbb{E}[c(G) \mid \overline{\mathcal{E}}] \cdot \mathbb{P}\left(\overline{\mathcal{E}}\right) \\ &\leq \frac{c_{\min}}{c(N)} \cdot c(N) + O(c(\textsc{Opt})) \\ &= O(c(\textsc{Opt})).\end{aligned}$$

Finally, Theorem 2.1 implies that $c(x) = O(\log n) \cdot c(\textsc{Opt})$. Setting $\epsilon = 1$, we conclude:

$$\begin{aligned}\mathbb{E}[c(R \cup G)] &\leq e(1+\epsilon)\log\left(T^2 \cdot \frac{f_{\max}}{f_{\min}}\right) c(x) + O(c(\textsc{Opt})) \\ &\leq O\left(\log\left(T \cdot \frac{f_{\max}}{f_{\min}}\right)\right) c(\textsc{Opt}).\end{aligned}$$

This completes the proof of correctness. The polynomial runtime follows from Lemma 3.4.2. $\square$

3.5 Conclusion

In this chapter, we gave a general algorithm for ONLINESUBMODULARCOVER. Our result generalizes the work of [AAA+09] to arbitrary submodular covering problems, thereby porting any black box use of SUBMODULARCOVER to the online setting. Our bounds are best possible in polynomial time unless $NP \subseteq BPP$. In Chapter 5 we will heavily reuse ideas from this chapter to study the seemingly unrelated problem of BLOCKAWARECACHING.

3.6 Deferred Proofs

3.6.1 Necessity of Time-Monotonicity

We note briefly that if we drop the time-monotonicity condition, then no algorithm can achieve a bounded competitive ratio. Consider the sequence $f^{(0)}, f^{(1)}, f^{(2)}$ such that $f^{(0)} \equiv f^{(2)} \equiv 0$, and $f^{(1)}(S) = \mathbf{1}(j \in S)$ for an abitrary choice of $j \in N$. In this case the optimal solution at time 2 is the empty set, whereas any online algorithm must output a set containing j.

3.6.2 Mutual Coverage is not Submodular

We give one example which shows that mutual coverage is in general **not** submodular as a function of one of its arguments. Furthermore, the example demonstrates that the condition $F_S(x) \geq \alpha f_S(N)$ for all subsets $S \subseteq N$ does not guarantee that $[\mathcal{I}^{(y|S)}(R)]R \sim x \geq \alpha \cdot \mathcal{I}^{(y|S)}(N)$. This justifies our use of the stronger LP (Q) instead of (P) in section Section 3.3.

Consider the following function on $\{a, b, y\}$:

$$\begin{array}{lll} f(\emptyset) = 0, & f(b) = 1, & f(b+y) = 2, \\ f(a+b+y) = 10, & f(y) = 1, & f(a+b) = 10, \\ & f(a) = 9, & f(a+y) = 10. \end{array}$$

The function f is submodular, but $\mathcal{I}^{(y)}(a) - \mathcal{I}^{(y)}(\emptyset) < \mathcal{I}^{(y)}(b+a) - \mathcal{I}^{(y)}(b)$, since:

$$\begin{aligned} &\mathcal{I}^{(y)}(a) - \mathcal{I}^{(y)}(\emptyset) \\ &\quad = f(a) - f(y+a) - f(\emptyset) + f(y) \\ &\quad = 0 \\ &\mathcal{I}^{(y)}(b+a) - \mathcal{I}^{(y)}(b) \\ &\quad = f(b+a) - f(a+b+y) - f(b) + f(b+y) \\ &\quad = 1. \end{aligned}$$

Now consider the vector $x = [x_a, x_b, x_y] = [3/4, 1/2, 0]$. One can check that $F_S(x) \geq 1/2 \cdot f_S(N)$ for all $S \subseteq N$ but:

$$\begin{aligned} &[\mathcal{I}^{(y)}(R)]R \sim x \\ &= \frac{3}{8}\mathcal{I}^{(y)}(a+b) + \frac{3}{8}\mathcal{I}^{(y)}(a) + \frac{1}{8}\mathcal{I}^{(y)}(b) + \frac{3}{8}\mathcal{I}^{(y)}(\emptyset) \\ &= \frac{3}{8} < \frac{1}{2}\mathcal{I}^{(y)}(N). \end{aligned}$$

Chapter 4

Random Order Set Cover

4.1 Introduction

In this chapter, we answer the question:

> *What is the best competitive ratio possible for* ONLINESETCOVER *when the adversary is constrained to reveal the elements in uniformly random order (RO)?*

We call this version ROSETCOVER. Note that the element set U is still adversarially chosen and unknown, and only the arrival order is random.

4.1.1 Results

We show that with only this one additional assumption on the element arrival order, there is an efficient algorithm for ROSETCOVER with expected competitive ratio matching the best-possible offline approximation guarantee (at least in the regime where $m = \mathrm{poly}(n)$).

Theorem 4.1.1. LEARNORCOVER *is a polynomial-time randomized algorithm for* ROSETCOVER *achieving expected competitive ratio* $O(\log(mn))$.

When run offline, our approach gives a new asymptotically optimal algorithm for SETCOVER for $m = \mathrm{poly}(n)$, which may be of independent interest. Indeed, given an estimate for the optimal *value* of the set cover, our algorithm makes a single pass over the elements (considered in random order), updating a fractional solution using a multiplicative-weights framework, and sampling sets as it goes. This simplicity, and the fact that it uses only $\tilde{O}(m)$ bits of memory, may make the algorithm useful for some low-space streaming applications. (Note that previous formulations of STREAMINGSETCOVER [SG09, DIMV14, HIMV16, ER16] only consider cases where *sets* arrive in a stream.)

We show next that a suitable generalization of the same algorithm achieves the same competitive ratio for the *RO Covering Integer Program* problem (ROCIP) (see Section 4.4 for a formal description).

Theorem 4.1.2. LEARNORCOVERCIP *is a polynomial-time randomized algorithm for* ROCIP *achieving competitive ratio* $O(\log(mn))$.

We complement our main theorem with some lower bounds. For instance, we show that the algorithms of [AAA+09, BN09b] have a performance of $\Theta(\log m \log n)$ even in RO, so a new algorithm is indeed needed. Moreover, we observe an $\Omega(\log n)$ lower bound on *fractional* algorithms for ROSetCover. This means we cannot pursue a two-phase strategy of maintaining a good monotone fractional solution and then randomly rounding it (as was done in prior works) without losing $\Omega(\log^2 n)$. Interestingly, our algorithm *does* maintain a (non-monotone) fractional solution while rounding it online, but does so in a way that avoids extra losses. We hope that our approach will be useful in other works for online problems in RO settings. (We also give other lower bounds for batched versions of the problem, and for the more general SubmodularCover problem.)

4.1.2 Techniques and Overview

The core contribution of this work is demonstrating that one can exploit randomness in the arrival order to *learn* about the underlying set system. What is more, this learning can be done fast enough (in terms of both sample and computational complexity) to build an $O(\log mn)$-competitive solution, even while committing to covering incoming elements immediately upon arrival. This seems like an idea with applications to other sequential decision-making problems, particularly in the RO setting.

We start in Section 4.2 with an exponential time algorithm for the unit-cost setting. This algorithm maintains a portfolio of all exponentially-many collections of cost $c(\text{Opt})$ that are feasible for the elements observed so far. When an uncovered element arrives, the algorithm takes a random collection from the portfolio, and picks a random set from it covering the element. It then prunes the portfolio to drop collections that did not cover the incoming element. We show that either the expected marginal coverage of the chosen set is large, or the expected number of solutions removed from the portfolio is large. I.e., we either make progress *covering*, or *learning*. We show that within $c(\text{Opt}) \log(mn)$ rounds, the portfolio only contains the true optimal feasible solutions, or all unseen elements are covered. (One insight that comes from our result is that a good measure of set quality is the number of times an unseen *and uncovered* element appears in it.)

We then give a polynomial-time algorithm in Section 4.3: while it is quite different from the exponential scheme above, it is also based on this insight, and the intuition that our algorithm should make progress via learning or covering. Specifically, we maintain a distribution $\{x_S\}_{S \in \mathcal{S}}$ on sets: for each arriving uncovered element, we first sample from this distribution x, then update x via a multiplicative weights rule. If the element remains uncovered, we buy the cheapest set covering it. For the analysis, we introduce a potential function which simultaneously measures the convergence of this distribution to the optimal fractional solution, and the progress towards covering the universe. Crucially, this progress is measured in expectation over the random order, thereby circumventing lower bounds for the adversarial-order setting [Kor04].

In Section 4.4 we extend our method to the more general ROCIP problem: the intuitions and general proof outlines are similar, but we need to extend the algorithm to handle elements being partially covered. In Section 4.5 we extend it further to give an algorithm for random order online NonMetricFacilityLocation, again matching the best possible offline bounds. In Section 4.6, we show that our LearnOrCover algorithm can be adapted to solve another variant of OnlineSetCover in which an α fraction of the input is sampled before the online sequence begins.

Finally, we present lower bounds in Section 4.7. Our information-theoretic lower bounds for ROSetCover follow from elementary combinatorial arguments. We also show a lower bound for a batched version of ROSetCover, following the hardness proof of [Kor04]; we use it in turn to derive lower bounds for ROSubmodularCover.

4.1.3 Related Work

There has been much recent interest in algorithms for random order online problems. Starting from the secretary problem, the RO model has been extended to include metric facility location [Mey01], network design [MMP01], and solving packing LPs [AWY14, MR14, KRTV18, GM16, AD15, AKL21], load-balancing [Mol17] and scheduling [AJ21a, AJ21b]. See [GS20] for a recent survey.

Our work is closely related to [GGL+13], who give an $O(\log mn)$-competitive algorithm a related stochastic model, where the elements are chosen i.i.d. from a *known distribution* over the elements. [DEH+18] generalize the result of [GGL+13] to the *prophet version*, in which elements are drawn from known but distinct distributions. Our work is a substantial strengthening, since the RO model captures the *unknown* i.i.d. setting as a special case. Moreover, the learning is an important and interesting part of our technical contribution. On the flip side, the algorithm of [GGL+13] satisfies *universality* (see their work for a definition), which our algorithm does not. We point out that [GGL+13] claim (in a note, without proof) that it is not possible to circumvent the $\Omega(\log m \log n)$ lower bound of [Kor04] even in RO; our results show this claim is incorrect. We discuss this in Section 4.11.

Regret bounds for online learning are also proven via the KL divergence (see, e.g., [AHK12]). However, no reductions are known from our problem to the classic MW setting, and it is unclear how random order would play a role in the analysis: this is necessary to bypass the adversarial order lower bounds.

Finally, [Kor04] gives an algorithm with competitive ratio $k\log(m/k)$— and hence total cost $k^2\log(m/k)$—for unweighted OnlineSetCover where $k = |\textsc{Opt}|$. The algorithm is the same as our exponential time algorithm in Section 4.2 for the special case of $k = 1$; however, we outperform it for non-constant k, and generalize it to get polynomial-time algorithms as well.

4.2 Warmup: An Exponential Time Algorithm for Unit Costs

We begin with an exponential-time algorithm which we call SimpleLearnOrCover to demonstrate some core ideas of our result. In what follows we assume that we know a number $k \in [c(\textsc{Opt}), 2 \cdot c(\textsc{Opt})]$. This assumption is easily removed by a standard guess and double procedure, at the cost of an additional factor of 2.

The algorithm is as follows. Maintain a list $\mathfrak{T} \subseteq \binom{\mathcal{S}}{k}$ of candidate k-tuples of sets. When an uncovered element v arrives, choose a k-tuple $\mathcal{T} = (T_1, \ldots, T_k)$ uniformly at random from $\mathfrak{T}$, and buy a uniformly random T from $\mathcal{T}$. Also buy an arbitrary set containing v. Finally, discard from $\mathfrak{T}$ any k-tuples that do not cover v. See Section 4.10 for pseudocode.

Theorem 4.2.1. SIMPLELEARNORCOVER *is a randomized algorithm for unit-cost* ROSET-COVER *with expected cost* $O(k \cdot \log(mn))$.

Proof of Theorem 4.2.1. Consider any time step t in which a random element arrives that is *uncovered* on arrival. Let U^t be the set of elements that remain uncovered at the end of time step t. Before the algorithm takes action, there are two cases:

Case 1: At least half the tuples in $\mathfrak{T}$ cover at least $|U^{t-1}|/2$ of the $|U^{t-1}|$ as-of-yet-uncovered elements. In this case we say the algorithm performs a **Cover Step** in round t.

Case 2: At least half the tuples in $\mathfrak{T}$ cover strictly less than $|U^{t-1}|/2$ of the uncovered elements; we say the algorithm performs a **Learning Step** in round t.

Define $\mathfrak{N}(c)$ to be the number of uncovered elements remaining after c Cover Steps. Define $\mathfrak{M}(\ell)$ to be the value of $|\mathfrak{T}|$ after ℓ Learning Steps. We will show that after $10k(\log m + \log n)$ rounds, either the number of elements remaining is less than $\mathfrak{N}(10k \log n)$ or the number of tuples remaining is less than $\mathfrak{M}(10k \log m)$. In particular, we argue that both $\mathbb{E}[\mathfrak{N}(10k \log n)]$ and $\mathbb{E}[\mathfrak{M}(10k \log m)]$ are less than 1.

Claim 4.2.2. $\mathbb{E}[\mathfrak{N}(c+1) \mid \mathfrak{N}(c) = N] \leq \left(1 - \frac{1}{4k}\right) N$.

Proof. If round t is a Cover Step, then at least half of the $\mathcal{T} \in \mathfrak{T}$ cover at least half of U^{t-1}, so $\mathbb{E}[|(\bigcup \mathcal{T}) \cap U^{t-1}|] \geq \frac{|U^{t-1}|}{4}$. Since T is drawn uniformly at random from the k sets in the uniformly random $\mathcal{T}$, we have $\mathbb{E}[|T \cap U^{t-1}|] \geq \frac{|U^{t-1}|}{4k}$. □

Claim 4.2.3. $\mathbb{E}[\mathfrak{M}(\ell+1) \mid \mathfrak{M}(\ell) = M] \leq \frac{3}{4} M$.

Proof. Upon the arrival of v in a Learning Step, at least half the tuples have probability at least $1/2$ of being removed from $\mathfrak{T}$, so the expected number of tuples removed from $\mathfrak{T}$ is at least $M/4$. □

To conclude, by induction

$$\mathbb{E}[\mathfrak{N}(10k \log n)] \leq n \left(1 - \tfrac{1}{4k}\right)^{10k \log n} \leq 1 \quad \text{and} \quad \mathbb{E}[\mathfrak{M}(10k \log m)] \leq m^k \left(\tfrac{3}{4}\right)^{10k \log m} \leq 1.$$

Note that if there are N remaining uncovered elements and M remaining tuples to choose from, the algorithm will pay at most $\min(k \cdot M, N)$ before all elements are covered. Thus the total expected cost of the algorithm is bounded by

$$10k(\log m + \log n) + \mathbb{E}[\min(k \cdot \mathfrak{M}(10k \log m), \mathfrak{N}(10k \log n))] = O(k \log mn). \qquad \square$$

Apart from the obvious challenge of modifying SIMPLELEARNORCOVER to run in polynomial time, it is unclear how to generalize it to handle non-unit costs. Still, the intuition from this algorithm will be useful for the next sections.

4.3 A Polynomial-Time Algorithm for General Costs

We build on our intuition from Section 4.2 that we can either make progress in covering or in learning about the optimal solution. To get an efficient algorithm, we directly maintain a probability distribution over sets, which we update via a multiplicative weights rule. We use a potential function that simultaneously measures progress towards learning the optimal solution, and towards covering the unseen elements. Before we present the formal details and the pseudocode, here are the main pieces of the algorithm.

1. We maintain a fractional vector x which is a (*not necessarily feasible*) guess for the LP solution of cost β to the SETCOVER instance.
2. Every round t in which an uncovered element v^t arrives, we
   (a) sample every set S with probability proportional to its current LP value x_S,
   (b) increase the value x_S of all sets $S \ni v^t$ multiplicatively and renormalize,
   (c) buy a cheapest set to cover v^t if it remains uncovered.

Formally, by a guess-and-double approach, we assume we know a bound β such that $\mathrm{LP}_{\mathrm{OPT}} \leq \beta \leq 2 \cdot \mathrm{LP}_{\mathrm{OPT}}$; here $\mathrm{LP}_{\mathrm{OPT}}$ is the cost of the optimal LP solution to the final unknown instance. Define

$$\kappa_v := \min\{c_S \mid S \ni v\} \tag{4.3.1}$$

as the cost of the cheapest set covering v.

Algorithm 1 LEARNORCOVER

```
1:  Let $m' \leftarrow |\{S : c(S) \leq \beta\}|$.
2:  Initialize $x_S^0 \leftarrow \frac{\beta}{c_S \cdot m'} \cdot \mathbb{1}\{c(S) \leq \beta\}$, and $\mathcal{C}^0 \leftarrow \emptyset$.
3:  for $t = 1, 2 \ldots, n$ do
4:      $v^t \leftarrow t^{th}$ element in the random order, and let $\mathcal{R}^t \leftarrow \emptyset$.
5:      if $v^t$ not already covered then
6:          for each set $S$, with probability $\min(\kappa_{v^t} \cdot x_S^{t-1}/\beta, 1)$ add $\mathcal{R}^t \leftarrow \mathcal{R}^t \cup \{S\}$.
7:          Update $\mathcal{C}^t \leftarrow \mathcal{C}^{t-1} \cup \mathcal{R}^t$.
8:          if $\sum_{S \ni v^t} x_S^{t-1} < 1$ then
9:              For every set $S$, update $x_S^t \leftarrow x_S^{t-1} \cdot \exp\{\mathbb{1}\{S \ni v^t\} \cdot \kappa_{v^t}/c_S\}$.
10:             Let $Z^t = \langle c, x^t \rangle/\beta$ and normalize $x^t \leftarrow x^t/Z^t$.
11:         else
12:             $x^t \leftarrow x^{t-1}$.
13:         Let $S_{v^t}$ be the cheapest set containing $v^t$. Add $\mathcal{C}^t \leftarrow \mathcal{C}^t \cup \{S_{v^t}\}$.
```

The algorithm is somewhat simpler for unit costs: the x_S values are multiplied by either 1 or e, and moreover we can sample a single set for $\mathcal{R}^t$ (see Section 4.10 for pseudocode). Because of the non-uniform set costs, we have to carefully calibrate both the learning and sampling rates. Our algorithm dynamically scales the learning and sampling rates in round t depending on κ_{v^t}, the cost of the cheapest set covering v^t. Intuitively, this ensures that all three of (a) the change in potential, (b) the cost of the sampling, and (c) the cost of the backup set, are at the same scale. Before we begin, observe that Line 10 ensures the following invariant:

Invariant 1. *For all time steps t, it holds that* $\langle c, x^t \rangle = \beta$.

Theorem 4.3.1. LEARNORCOVER *is a polynomial-time randomized algorithm for* ROSETCOVER *with expected cost* $O(\beta \cdot \log(mn))$.

Let us start by defining notation. Let x^* to be the optimal LP solution to the final, unknown set cover instance. Next let $X_v^t := \sum_{S \ni v} x_S^t$, the fractional coverage provided to v by x^t. Let U^t be the elements remaining uncovered at the end of round t (where $U^0 = U$ is the entire ground set of the set system). Define the quantity

$$\rho^t := \sum_{u \in U^t} \kappa_u. \tag{4.3.2}$$

With this, we are ready to define our potential function which is the central player in our analysis. Recalling that β is our guess for the value of $\mathrm{LP}_{\mathrm{OPT}}$, and the definition of KL divergence (2.1.3), define

$$\boxed{\Phi(t) := C_1 \cdot \mathrm{KL}_c\left(x^* \mid\mid x^t\right) + C_2 \cdot \beta \cdot \log\left(\frac{\rho^t}{\beta}\right)} \tag{4.3.3}$$

where C_1 and C_2 are constants to be specified later.

Lemma 4.3.2 (Initial Potential)**.** *The initial potential is bounded as* $\Phi(0) = O(\beta \cdot \log(mn))$.

We write the proof in general language in order to reuse it for covering IPs in the following section. Recall that sets correspond to columns in the canonical formulation of SetCover as an integer program.

Proof. We require the following fact which we prove in Section 4.9.

Fact 4.3.3. *Every pure covering LP of the form* $\min_{x \geq 0}\{\langle c, x\rangle : Ax \geq 1\}$ *for* $c \geq 0$ *and* $a_{ij} \in [0,1]$ *with optimal value less than* β *has an optimal solution* x^* *which is supported only on columns* j *such that* $c_j \leq \beta$.

We assume WLOG that x^* is such a solution, and we first bound the KL term of $\Phi(0)$. Since support$(x^*) \subseteq$ support(x^0) by Fact 4.3.3, we have

$$\mathrm{KL}_c\left(x^* \mid\mid x^0\right) = \sum_j c_j \cdot x_j^* \log\left(x_j^* \frac{c_j \cdot m'}{\beta}\right) + \sum_j c_j(x_j^0 - x_j^*) \leq \beta(\log(m) + 1),$$

where we used that $\langle c, x^*\rangle \leq \beta$ and that $m' < m$ is the number of columns with cost less than β.

For the second term, $\beta \log(\rho^0/\beta) = \beta \log(\sum_{i \in U^0} \kappa_i/\beta) \leq \beta \log(|U^0|) = \beta \log n$, since for all i, the cheapest cover for i costs less than β, and therefore $\kappa_u/\beta \leq 1$. The claim follows so long as C_1 and C_2 are constants. $\square$

The rest of the proof relates the expected decrease of potential Φ to the algorithm's cost in each round. Define the event $\Upsilon^t := \{v^t \in U^{t-1}\}$ that the element v^t is *uncovered* on arrival. Note Line 5 ensures that if event Υ^t does not hold, the algorithm takes no action and the potential does not change. So we focus on the case that event Υ^t does occur. We first analyze the change in KL divergence. Recall that $X_v^t := \sum_{S \ni v} x_S^t$.

Lemma 4.3.4 (Change in KL)**.** *For rounds* t *in which* Υ^t *holds, the expected change in the weighted KL divergence is*

$$\begin{aligned}&\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \\ &\leq \mathop{\mathbb{E}}_{v \sim U^{t-1}}[(e-1) \cdot \kappa_v \min(X_v^{t-1}, 1) - \kappa_v].\end{aligned}$$

We emphasize that the expected change in relative entropy in the statement above depends only on the randomness of the arriving uncovered element v^t, not on the randomly chosen sets $\mathcal{R}^t$.

Proof. We break the proof into cases. If $X_v^{t-1} \geq 1$, in Line 12 we set the vector $x^t = x^{t-1}$, so the change in KL divergence is 0. This means that

$$\begin{aligned}
&\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t, X_{v^t}^{t-1} \geq 1\right] \\
&\leq \mathop{\mathbb{E}}_{v\sim U^t}\left[(e-1)\cdot\kappa_v \min\left(X_v^{t-1},1\right) - \kappa_v \mid X_v^{t-1} \geq 1\right] \qquad (4.3.4)
\end{aligned}$$

trivially. Henceforth we focus on the case $X_{v^t}^{t-1} < 1$.

Recall that the expected change in relative entropy depends only on the arriving uncovered element v^t. Expanding definitions,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t, X_{v^t}^{t-1} < 1\right] \\
&= \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\sum_S c_S \cdot x_S^* \cdot \log\frac{x_S^{t-1}}{x_S^t} \,\middle|\, X_v^{t-1} < 1\right] \\
&= \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\langle c, x^*\rangle \cdot \log Z^t - \sum_{S\ni v} c_S \cdot x_S^* \cdot \log e^{\kappa_v/c_S} \,\middle|\, X_v^{t-1} < 1\right] \\
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\beta\cdot\log\left(\sum_{S\ni v}\frac{c_S}{\beta}\cdot x_S^{t-1}\cdot e^{\kappa_v/c_S} + \sum_{S\not\ni v}\frac{c_S}{\beta}\cdot x_S^{t-1}\right) - \sum_{S\ni v}\kappa_v\cdot x_S^* \,\middle|\, X_v^{t-1} < 1\right], \\
&\qquad (4.3.5)
\end{aligned}$$

where in the last step (4.3.5) we expanded the definition of Z^t, and used $\langle c, x^*\rangle \leq \beta$. Since x^* is a feasible set cover, which means that $\sum_{S\ni v} x_S^* \geq 1$, we can further bound (4.3.5) by

$$\begin{aligned}
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\beta\cdot\log\left(\sum_{S\ni v}\frac{c_S}{\beta}\cdot x_S^{t-1}\cdot e^{\kappa_v/c_S} + \sum_{S\not\ni v}\frac{c_S}{\beta}\cdot x_S^{t-1}\right) - \kappa_v \,\middle|\, X_v^{t-1} < 1\right] \\
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\beta\cdot\log\left(\sum_{S}\frac{c_S}{\beta}\cdot x_S^{t-1} + (e-1)\cdot\sum_{S\ni v}\frac{\kappa_v}{\beta}\cdot x_S^{t-1}\right) - \kappa_v \,\middle|\, X_v^{t-1} < 1\right] \qquad (4.3.6)
\end{aligned}$$

where we use the approximation $e^y \leq 1 + (e-1)\cdot y$ for $y \in [0,1]$ (note that κ_v is the cheapest set covering v, so for any $S \ni v$ we have $\kappa_v/c_S \leq 1$). Finally, using Invariant 1, along with the approximation $\log(1+y) \leq y$, we bound (4.3.6) by

$$\begin{aligned}
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[(e-1)\cdot\kappa_v\cdot X_v^{t-1} - \kappa_v \,\middle|\, X_v^{t-1} < 1\right] \\
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[(e-1)\cdot\kappa_v \min(X_v^{t-1},1) - \kappa_v \mid X_v^{t-1} < 1\right]. \qquad (4.3.7)
\end{aligned}$$

The lemma statement follows by combining (4.3.4) and (4.3.7) using the law of total expectation. □

Next we bound the expected change in $\log\rho^t$ provided by the sampling $\mathcal{R}^t \sim \kappa_{v^t}x^{t-1}/\beta$ upon the arrival of uncovered v^t (where each $S \in \mathcal{R}^t$ independently with probability $\kappa_{v^t}x_S^{t-1}/\beta$). Recall that U^t denotes the elements uncovered at the end of round t; therefore element u is contained in $U^{t-1} \setminus U^t$ if and only if it is marginally covered by $\mathcal{R}^t$ in this round.

Lemma 4.3.5 (Change in $\log \rho^t$). *For rounds when v is uncovered on arrival, the expected change in $\log \rho^t$ is*

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t}\left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \leq -\frac{1-e^{-1}}{\beta} \cdot \mathop{\mathbb{E}}_{u \sim U^{t-1}}\left[\kappa_u \cdot \min(X_u^{t-1}, 1)\right].$$

Proof. Recall the definition of ρ^t from (4.3.2). Conditioned on $v^t = v$ for any fixed element v, the expected change in $\log \rho^t$ depends only on $\mathcal{R}^t$. Recall $\mathcal{R}^t$ is formed by sampling each set S with probability $\kappa_v x_S^{t-1}/\beta$.

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t, v^t = v\right] \\
&= \mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\log\left(1 - \frac{\rho^{t-1} - \rho^t}{\rho^{t-1}}\right) \,\middle|\, U^{t-1}, v^t = v\right] \\
&\leq -\frac{1}{\rho^{t-1}} \cdot \mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\rho^{t-1} - \rho^t \,\middle|\, U^{t-1}, v^t = v\right] && (4.3.8)
\end{aligned}$$

Above, (4.3.8) follows from the approximation $\log(1-y) \leq -y$. Expanding the definition of ρ^t from (4.3.2), (4.3.8) is bounded by

$$\begin{aligned}
&= -\frac{1}{\rho^{t-1}} \cdot \mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\sum_{u \in U^{t-1}} \kappa_u \cdot \mathbb{1}\{u \in U^{t-1} \setminus U^t\} \,\middle|\, U^{t-1}, v^t = v\right] \\
&= -\frac{1}{\rho^{t-1}} \sum_{u \in U^{t-1}} \kappa_u \cdot \mathop{\mathbb{P}}_{\mathcal{R}^t}\left(u \notin U^t \mid u \in U^{t-1}, v^t = v\right) \\
&\leq -\frac{1-e^{-1}}{\beta} \cdot \kappa_v \cdot \frac{1}{\rho^{t-1}} \sum_{u \in U^{t-1}} \kappa_u \cdot \min(X_u^{t-1}, 1) && (4.3.9) \\
&= -\frac{1-e^{-1}}{\beta} \cdot \kappa_v \cdot \frac{|U^{t-1}|}{\rho^{t-1}} \cdot \mathop{\mathbb{E}}_{u \sim U^{t-1}}\left[\kappa_u \cdot \min(X_u^{t-1}, 1)\right]. && (4.3.10)
\end{aligned}$$

Step (4.3.9) is due to the fact that each set $S \in \mathcal{R}^t$ is sampled independently with probability $\min(\kappa_v x_S^{t-1}/\beta, 1)$, so the probability any given element $u \in U^{t-1}$ is covered is

$$\begin{aligned}
1 - \prod_{S \ni u}\left(1 - \min\left(\frac{\kappa_v x_S^{t-1}}{\beta}, 1\right)\right) &\geq 1 - \exp\left\{-\min\left(\frac{\kappa_v}{\beta} X_u^{t-1}, 1\right)\right\} \\
&\overset{(**)}{\geq} (1 - e^{-1}) \cdot \min\left(\frac{\kappa_v}{\beta} X_u^{t-1}, 1\right)
\end{aligned}$$

Above, $(**)$ follows from convexity of the exponential. Step (4.3.10) follows since $\kappa_v/\beta \leq 1$. Taking the expectation of (4.3.10) over $v^t \sim U^{t-1}$, and using the fact that $\mathbb{E}_{v \sim U^{t-1}}[\kappa_v] = \rho^{t-1}/|U^{t-1}|$, the expected change in $\log \rho^t$ becomes

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t}\left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \leq -\frac{1-e^{-1}}{\beta} \cdot \mathop{\mathbb{E}}_{u \sim U^{t-1}}\left[\kappa_u \cdot \min(X_u^{t-1}, 1)\right],$$

as desired. □

Proof of Theorem 4.3.1. In every round t for which Υ^t holds, the expected cost of the sampled sets $\mathcal{R}^t$ is $\kappa_{v^t} \cdot \langle c, x^{t-1} \rangle / \beta = \kappa_{v^t}$ (by Invariant 1). The algorithm pays an additional κ_{v^t} in Line 13, and hence the total expected cost per round is at most $2 \cdot \kappa_{v^t}$.

Combining Lemmas 4.3.4 and 4.3.5, and setting the constants $C_1 = 2$ and $C_2 = 2e$, we have

$$\begin{aligned}
&\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Phi(t) - \Phi(t-1) \;\middle|\; v^1, \dots, v^{t-1}, \mathcal{R}^1, \dots, \mathcal{R}^{t-1}, \Upsilon^t\right] \\
&= \mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\begin{array}{rl} & C_1 \left(\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right)\right) \\ + & C_2 \cdot \beta \cdot \left(\log \rho^t - \log \rho^{t-1}\right) \end{array} \;\middle|\; v^1, \dots, v^{t-1}, \mathcal{R}^1, \dots, \mathcal{R}^{t-1}, \Upsilon^t\right] \\
&\leq - \mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[2 \cdot \kappa_{v^t} \;\middle|\; v^1, \dots, v^{t-1}, \mathcal{R}^1, \dots, \mathcal{R}^{t-1}, \Upsilon^t\right],
\end{aligned}$$

which cancels the expected change in the algorithm's cost. Since neither the potential Φ nor the cost paid by the algorithm change during rounds in which Υ^t does not hold, we have the inequality

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Phi(t) - \Phi(t-1) + c(\mathrm{ALG}(t)) - c(\mathrm{ALG}(t-1)) \;\middle|\; v^1, \dots, v^{t-1}, \mathcal{R}^1, \dots, \mathcal{R}^{t-1}\right] \leq 0.$$

Let t^* be the last time step for which $\Phi(t^*) \geq 0$. By applying Lemma 4.9.1 and the bound on the starting potential from Lemma 4.3.2, we have that $\mathbb{E}[c(\mathrm{ALG}(t^*))] \leq O(\beta \cdot \log(mn))$.

It remains to bound the expected cost paid by the algorithm after time t^*. Since KL divergence is a nonnegative quantity, Φ is negative only when $\rho^t \leq \beta$. The algorithm pays $O(\kappa_{v^t})$ in expectation during rounds t where $v^t \in U^{t-1}$ and 0 during rounds where $v^t \notin U^{t-1}$, and hence the expected cost paid by the algorithm after time t^* is at most $\sum_{u \in U^{t^*}} \kappa_u = \rho^{t^*} = O(\beta)$. □

4.4 Covering Integer Programs

We show how to generalize our algorithm from Section 4.3 to solve pure covering IPs when the constraints are revealed in random order, which significantly generalizes ROSETCOVER. Formally, the random order covering IP problem (ROCIP) is to solve

$$\begin{array}{ll} \min_z & \langle c, z \rangle \\ \text{s.t.} & Az \geq 1 \\ & z \in \mathbb{Z}_+^m, \end{array} \tag{4.4.1}$$

when the rows of A are revealed in random order. Furthermore the solution z can only be incremented monotonically and must always be feasible for the subset of constraints revealed so far. (Note that we do not consider box constraints, namely upper-bound constraints of the form $z_j \leq d_j$.) We may assume without loss of generality that the entries of A are $a_{ij} \in [0, 1]$.

We describe an algorithm which guarantees that every row is covered to extent $1 - \gamma$, meaning it outputs a solution z with $Az \geq 1 - \gamma$ (this relaxation is convenient in the proof for technical reasons). With foresight, we set $\gamma = (e-1)^{-1}$. It is straightforward to wrap this algorithm in one that buys $\lceil (1-\gamma)^{-1} \rceil = 3$ copies of every column and truly satisfy the constraints, which only incurs an additional factor of 3 in the cost.

Once again, by a guess-and-double approach, we assume we know a bound β such that $\mathrm{LP}_{\mathrm{OPT}} \leq \beta \leq 2 \cdot \mathrm{LP}_{\mathrm{OPT}}$; here $\mathrm{LP}_{\mathrm{OPT}}$ is the cost of the optimal solution to the LP relaxation of (4.4.1). Let

z^t be the integer solution held by the algorithm at the end of round t. Define $\Delta_i^t := \max(0, 1 - \langle a_i, z^t \rangle)$ to be the extent to which i remains uncovered at the end of round t. This time we redefine $\kappa_i^t := \Delta_i^{t-1} \cdot \min_k c_k/a_{ik}$, which becomes the minimum fractional cost of covering the current deficit for i. Finally, for a vector y, denote the fractional remainder by $\widetilde{y} := y - \lfloor y \rfloor$.

The algorithm once again maintains a fractional vector x which is a guess for the (potentially infeasible) LP solution of cost β to (4.4.1). This time, when the i^{th} row arrives at time t and $\Delta_i^{t-1} > \gamma$ (meaning this row is *not* already covered to extent $1 - \gamma$), we (a) buy a random number of copies of every column j with probability proportional to its LP value x_j, (b) increase the value x_j multiplicatively and renormalize, and finally (c) buy a minimum cost cover for row i if necessary.

Algorithm 2 LEARNORCOVERCIP

```
1: $m' \leftarrow |\{j : c_j \leq \beta\}|$.
2: Initialize $x_j^0 \leftarrow \frac{\beta}{c_j \cdot m'} \cdot \mathbb{1}\{c_j \leq \beta\}$, and $z^0 \leftarrow \vec{0}$.
3: for $t = 1, 2 \ldots, n$ do
4:     $i \leftarrow t^{th}$ constraint in the random order.
5:     if $\Delta_i^{t-1} > \gamma$ then
6:         Let $y := \kappa_i^t \cdot x^{t-1}/\beta$. for each column $j$, add $z_j^t \leftarrow z_j^{t-1} + \lfloor y_j \rfloor + \text{Ber}(\tilde{y}_j)$.
7:         if $\langle a_i, x^{t-1} \rangle < \Delta_i^{t-1}$ then
8:             For every $j$, update $x_j^t \leftarrow x_j^{t-1} \cdot \exp\left\{\kappa_i^t \cdot \frac{a_{ij}}{c_j}\right\}$.
9:             Let $Z^t = \langle c, x^t \rangle/\beta$ and normalize $x^t \leftarrow x^t/Z^t$.
10:        else
11:            $x^t \leftarrow x^{t-1}$
12:        Let $k^* = \text{argmin}_k \frac{c_k}{a_{ik}}$. Add $z_{k^*}^t \leftarrow z_{k^*}^t + \left\lceil \frac{\Delta_i^{t-1}}{a_{ik^*}} \right\rceil$.
```

Note that once again, Line 9 ensures Invariant 1 holds. The main theorem of this section is:

Theorem 4.4.1. LEARNORCOVERCIP *is a polynomial-time randomized algorithm for* ROCIP *which outputs a solution z with expected cost $O(\beta \cdot \log(mn))$ such that $3z$ is feasible.*

Theorem 4.1.2 follows as a corollary, since given any intermediate solution z^t, we can buy the scaled solution $3z^t$.

We generalize the proof of Theorem 4.3.1. Redefine x^* to be the optimal LP solution to the final, unknown instance (4.4.1), and $U^t := \{i \mid \Delta_i^t > \gamma\}$ be the elements which are not covered to extent $1 - \gamma$ at the end of round t. With these new versions of U^t and κ^t, the definitions of both ρ^t and the potential Φ remain the same as in (4.3.2) and (4.3.3) (except we pick the constants C_1 and C_2 differently).

Once again, we start with a bound on the initial potential.

Lemma 4.4.2 (Initial Potential)**.** *The initial potential is bounded as $\Phi(0) = O(\beta \cdot \log(mn))$.*

The proof is identical verbatim to that of Lemma 4.3.2.

It remains to relate the expected decrease in Φ to the algorithm's cost in every round. For convenience, let $X_i^t := \langle a_i, x^t \rangle$ be the amount that x^t fractionally covers i. Define Υ^t to be the event that for constraint i^t arriving in round t we have $\Delta_i^{t-1} > \gamma$. The check at Line 5 ensures that if Υ^t does not hold, then neither the cost paid by the algorithm nor the potential will change. We focus on the case that Υ^t occurs, and once again start with the KL divergence.

Lemma 4.4.3 (Change in KL)**.** *For rounds in which Υ^t holds, the expected change in weighted KL divergence is*

$$\begin{aligned}
&\mathop{\mathbb{E}}_{i^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \\
&\le \mathop{\mathbb{E}}_{i\sim U^{t-1}}[(e-1)\cdot\kappa_i^t \min(X_i^{t-1}, \Delta_i^{t-1}) - \kappa_i^t].
\end{aligned}$$

Proof. We break the proof into cases. By the check on Line 7, if $X_{i^t}^{t-1} \ge \Delta_{i^t}^{t-1}$, then the vector x^t is not updated in round t, so the change in KL divergence is 0. This means that

$$\begin{aligned}
&\mathop{\mathbb{E}}_{i^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t, X_{i^t}^{t-1} \ge \Delta_{i^t}^{t-1}\right] \\
&\le \mathop{\mathbb{E}}_{i\sim U^{t-1}}[(e-1)\cdot\kappa_i^t \min(X_i^{t-1}, \Delta_i^{t-1}) - \kappa_i^t \mid X_i^{t-1} \ge \Delta_i^{t-1}] \qquad (4.4.2)
\end{aligned}$$

holds trivially, since in this case $\min(X_i^{t-1}, \Delta_i^{t-1}) = \Delta_i^{t-1} > \gamma = (e-1)^{-1}$ and $\kappa_i^t \ge 0$. Henceforth we focus on the case $X_i^{t-1} < \Delta_i^{t-1}$.

The change in relative entropy depends only on the arriving uncovered constraint i^t, not on the randomly chosen columns $\mathcal{R}^t$. Expanding definitions,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{i^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t, X_i^{t-1} < \Delta_i^{t-1}\right] \\
&= \mathop{\mathbb{E}}_{i\sim U^{t-1}}\left[\sum_j c_j \cdot x_j^* \cdot \log \frac{x_j^{t-1}}{x_j^t} \,\middle|\, X_i^{t-1} < \Delta_i^{t-1}\right] \\
&= \mathop{\mathbb{E}}_{i\sim U^{t-1}}\left[\sum_j c_j \cdot x_j^* \cdot \log Z^t - \sum_j c_j \cdot x_j^* \cdot \kappa_i^t \cdot \frac{a_{ij}}{c_j} \,\middle|\, X_i^{t-1} < \Delta_i^{t-1}\right] \\
&\le \mathop{\mathbb{E}}_{i\sim U^{t-1}}\left[\beta \cdot \log Z^t - \kappa_i^t \cdot \sum_j a_{ij} x_j^* \,\middle|\, X_i^{t-1} < \Delta_i^{t-1}\right] \qquad (4.4.3)
\end{aligned}$$

where we used that $\langle c, x^*\rangle \le \beta$. Expanding the definition of Z^t and applying the fact that x^* is a feasible solution i.e. $\langle a_i, x^*\rangle \ge 1$, we continue to bound (4.4.3) as

$$\begin{aligned}
&\le \mathop{\mathbb{E}}_{i\sim U^{t-1}}\left[\beta \cdot \log\left(\frac{1}{\beta}\sum_j c_j x_j^{t-1} \exp\left(\kappa_i^t \cdot \frac{a_{ij}}{c_j}\right)\right) - \kappa_i^t \,\middle|\, X_i^{t-1} < \Delta_i^{t-1}\right] \\
&\le \mathop{\mathbb{E}}_{i\sim U^{t-1}}\left[\beta \cdot \log\left(1 + \frac{e-1}{\beta} \cdot \kappa_i^t \cdot \sum_j a_{ij} x_j^{t-1}\right) - \kappa_i^t \,\middle|\, X_i^{t-1} < \Delta_i^{t-1}\right]. \qquad (4.4.4)
\end{aligned}$$

(4.4.4) is derived by applying the approximation $e^y \le 1 + (e-1)y$ for $y \in [0,1]$ and the fact that $\langle c, x^{t-1}\rangle = \beta$ by Invariant 1; the exponent lies in $[0,1]$ because by definition $\kappa_i^t \cdot a_{ij}/c_j = \Delta_i^{t-1} \cdot (a_{ij}/c_j) \cdot \min_k(c_k/a_{ik}) \le 1$ since $\Delta_i^{t-1} \in [0,1]$. Finally, using the fact that $\log(1+y) \le y$, we have that (4.4.4) is at most

$$\le \mathop{\mathbb{E}}_{i\sim U^{t-1}}\left[(e-1)\cdot\kappa_i^t \cdot \sum_j a_{ij} x_j^{t-1} - \kappa_i^t \,\middle|\, X_i^{t-1} < \Delta_i^{t-1}\right]$$

$$\leq \mathop{\mathbb{E}}_{i \sim U^{t-1}} \left[(e-1) \cdot \kappa_i^t \cdot \min\left(X_i^{t-1}, \Delta_i^{t-1}\right) - \kappa_i^t \;\middle|\; X_i^{t-1} < \Delta_i^{t-1} \right]. \tag{4.4.5}$$

The lemma statement follows by combining (4.4.2) and (4.4.5) using the law of total expectation. □

We move to bounding the expected change in $\log \rho^t$ provided by updating the solution z on Line 6 on the arrival of the random row i. Recall that $U^t = \{i \mid \Delta_i^t > \gamma\}$ are the unseen elements which are at most half covered by z.

We will make use of the following lemma, which we prove in Section 4.9:

Fact 4.4.4. *Given probabilities p_j and coefficients $b_j \in [0,1]$, let $W := \sum_j b_j \operatorname{Ber}(p_j)$ be the sum of independent weighted Bernoulli random variables. Let $\Delta \geq \gamma = (e-1)^{-1}$ be some constant. Then*

$$\mathbb{E}[\min(W, \Delta)] \geq \alpha \cdot \min(\mathbb{E}[W], \Delta),$$

for a fixed constant α independent of the p_j and b_j.

We are ready to bound the expected change in $\log \rho^t$.

Lemma 4.4.5 (Change in $\log \rho^t$). *For rounds in which Υ^t holds, the expected change in $\log \rho^t$ is*

$$\mathop{\mathbb{E}}_{i^t, \mathcal{R}^t} \left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t \right] \leq -\frac{\alpha}{\beta} \cdot \mathop{\mathbb{E}}_{i' \sim U^{t-1}} \left[\kappa_{i'}^t \cdot \min\left(X_{i'}^{t-1}, \Delta_{i'}^{t-1}\right) \right]$$

where α is a fixed constant.

Proof. Conditioned on $i^t = i$, the expected change in $\log \rho^t$ depends only on $\mathcal{R}^t$.

$$\begin{aligned}
&\mathop{\mathbb{E}}_{i^t, \mathcal{R}^t} \left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t, i^t = i \right] \\
&= \mathop{\mathbb{E}}_{\mathcal{R}^t} \left[\log\left(1 - \frac{\rho^{t-1} - \rho^t}{\rho^{t-1}}\right) \;\middle|\; U^{t-1}, i^t = i \right] \\
&\leq -\frac{1}{\rho^{t-1}} \mathop{\mathbb{E}}_{\mathcal{R}^t} \left[\rho^{t-1} - \rho^t \;\middle|\; U^{t-1}, i^t = i \right].
\end{aligned} \tag{4.4.6}$$

Above, follows from the approximation $\log(1-y) \leq -y$. Expanding definitions again, and using the fact that $\kappa_{i'}^{t-1} - \kappa_{i'}^t = \min_k (c_k / a_{i'k}) \cdot (\Delta_{i'}^{t-1} - \Delta_{i'}^t) \geq \kappa_{i'}^t (\Delta_{i'}^{t-1} - \Delta_{i'}^t)$, we further bound (4.4.6) by

$$\begin{aligned}
&\leq -\frac{1}{\rho^{t-1}} \mathop{\mathbb{E}}_{\mathcal{R}^t} \left[\sum_{i' \in U^{t-1}} \kappa_{i'}^t \cdot (\Delta_{i'}^{t-1} - \Delta_{i'}^t) \right] \\
&= -\frac{1}{\rho^{t-1}} \sum_{i' \in U^{t-1}} \kappa_{i'}^t \cdot \mathop{\mathbb{E}}_{\mathcal{R}^t} [\Delta_{i'}^{t-1} - \Delta_{i'}^t] \\
&= -\frac{1}{\rho^{t-1}} \sum_{i' \in U^{t-1}} \kappa_{i'}^t \cdot \alpha \cdot \min\left(\frac{\kappa_i^t}{\beta} \cdot X_{i'}^{t-1},\ \Delta_{i'}^{t-1} \right).
\end{aligned} \tag{4.4.7}$$

To understand (4.4.7), note that $\Delta_{i'}^{t-1} - \Delta_{i'}^{t} = \min(\sum_j a_{i'j} \lfloor y_j \rfloor + \sum_j a_{i'j} \operatorname{Ber}(\widetilde{y}),\ \Delta_{i'}^{t-1})$. By the definition of y, the first term inside the minimum has expectation $\frac{\kappa_i^t}{\beta} \cdot X_{i'}^{t-1}$, and since Υ^t holds we have $\Delta_{i'}^{t-1} > \gamma$. Therefore applying Fact 4.4.4 gives (4.4.7) (where α is the constant given by the lemma). Since $\kappa_i^t/\beta \le 1$, we bound (4.4.7) with

$$\begin{aligned} &\le -\frac{\alpha}{\beta} \cdot \kappa_i^t \cdot \frac{1}{\rho^{t-1}} \sum_{i' \in U^{t-1}} \kappa_{i'}^t \cdot \min\left(X_{i'}^{t-1}, \Delta_{i'}^{t-1}\right) \\ &= -\frac{\alpha}{\beta} \cdot \kappa_i^t \cdot \frac{|U^{t-1}|}{\rho^{t-1}} \cdot \mathop{\mathbb{E}}_{i' \sim U^{t-1}}\left[\kappa_{i'}^t \cdot \min\left(X_{i'}^{t-1}, \Delta_{i'}^{t-1}\right)\right]. \end{aligned} \tag{4.4.8}$$

Taking the expectation of (4.4.8) over $i \sim U^{t-1}$, and using the fact that $\mathbb{E}_{i \sim U^{t-1}}[\kappa_i^t] = \rho^{t-1}/U^{t-1}$, the expected change in $\log \rho^t$ becomes

$$\mathop{\mathbb{E}}_{i^t, \mathcal{R}^t}\left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \le -\frac{\alpha}{\beta} \cdot \mathop{\mathbb{E}}_{i' \sim U^{t-1}}\left[\kappa_{i'}^t \cdot \min\left(X_{i'}^{t-1}, \Delta_{i'}^{t-1}\right)\right],$$

as desired. □

We may now combine the two previous lemmas as before.

Proof of Theorem 4.4.1. In the round in which constraint i arrives, the expected cost of sampling is $\frac{\kappa_i^t}{\beta}\langle c, x^{t-1}\rangle = \kappa_i^t$ (by Invariant 1). The algorithm pays an additional

$$c_{k^*}\left\lceil \frac{\Delta_i^{t-1}}{a_{ik^*}} \right\rceil = c_{k^*}\left\lceil \frac{\kappa_i^t}{c_{k^*}} \right\rceil \le 2\kappa_i^t$$

in Line 13, where this upper bound holds because $\frac{\kappa_i^t}{c_{k^*}} = \frac{\Delta_i^{t-1}}{a_{ik^*}} \ge 1/2$, since $\Delta_i^{t-1} \ge \gamma \ge 1/2$ and $a_{ik^*} \le 1$. Hence the total expected cost per round is at most $3 \cdot \kappa_i^t$.

Combining Lemma 4.4.3 and Lemma 4.4.5 and choosing $C_1 = 3$ and $C_2 = 3(e-1)/\alpha$, we have

$$\begin{aligned} &\mathop{\mathbb{E}}_{i^t, \mathcal{R}^t}\left[\Phi(t) - \Phi(t-1) \mid i^1, \ldots, i^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}, \Upsilon^t\right] \\ &= \mathop{\mathbb{E}}_{i^t, \mathcal{R}^t}\left[\begin{array}{l} C_1 \cdot (\mathrm{KL}_c(x^* \,||\, x^t) - \mathrm{KL}_c(x^* \,||\, x^{t-1})) \\ + \;\; C_2 \cdot \beta \cdot (\log \rho^t - \log \rho^{t-1}) \end{array} \middle|\, i^1, \ldots, i^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}, \Upsilon^t\right] \\ &\le -\mathop{\mathbb{E}}_{i^t, \mathcal{R}^t}\left[3 \cdot \kappa_{i^t}^t \mid i^1, \ldots, i^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}, \Upsilon^t\right], \end{aligned}$$

which cancels the expected change in the algorithm's cost. Hence we have the inequality

$$\mathop{\mathbb{E}}_{i^t, \mathcal{R}^t}\left[\Phi(t) - \Phi(t-1) + c(\textsc{Alg}(t)) - c(\textsc{Alg}(t-1)) \mid i^1, \ldots, i^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}\right] \le 0.$$

Let t^* be the last time step for which $\Phi(t^*) \ge 0$. By applying Lemma 4.9.1 and the bound on the starting potential, $\Phi(0) = O(\beta \cdot \log(mn))$, we have that $\mathbb{E}[c(\textsc{Alg}(t^*))] \le O(\beta \cdot \log(mn))$.

To bound the expected cost of the algorithm after time t^*, note that as before that KL divergence is a nonnegative quantity, and Φ is negative only when $\rho^t \le \beta$. The algorithm pays $O(\kappa_i^t)$ in expectation during rounds t where $i \in U^{t-1}$ and 0 during rounds where $i \notin U^{t-1}$, and hence the expected cost paid by the algorithm after time t^* is at most $\sum_{i \in U^{t^*}} \kappa_i^{t^*} = \rho^{t^*} = O(\beta)$. □

4.5 Non- Metric Facility Location

We continue with an extension of LEARNORCOVER to a problem that is not on its surface an instance of SUBMODULARCOVER: NONMETRICFACILITYLOCATION. The input is a set of n clients and m facilities. Each facility f has an opening cost c_f, and each client-facility pair (v, f) has a connection cost c_{fv}. The goal is to open a number of facilities and connect each client to exactly one open facility such that the total cost is minimized. In the random order online version, which we call ROFACILITYLOCATION, the facilities are known ahead of time, and the clients arrive online in random order; on arrival each client, the algorithm must choose which (if any) facility to open, and connect the client to that facility. Decisions are irrevocable, in the sense that a client may not change which facility it has connected to after arrival.

Our algorithm is essentially LEARNORCOVER run on the following *dynamic* set system that changes with time. The sets are the facilities and the clients are the elements. Let $\mathcal{C}^t$ be the facilities purchased by the end of round t. For every client v, define

$$f^t(v) := \operatorname*{argmin}_f(\mathbb{1}\{f \notin \mathcal{C}^t\} \cdot c_f + c_{fv})$$
$$\kappa_v^t := \min_f(\mathbb{1}\{f \notin \mathcal{C}^t\} \cdot c_f + c_{fv})$$

to be the marginal cost of connecting v in the cheapest way possible. Also define $\Gamma^t(v) := \{f : c_{fv} \leq \kappa_v^t/2\}$ to be the set of set of facilities that reduce the marginal cost of connecting v by at least a factor of 2. Then a facility f *covers* a client v at time t, if $f \in \Gamma^t(v)$.

As before, by a guess-and-double approach, we assume we know a bound β such that $\text{LP}_{\text{OPT}} \leq \beta \leq 2 \cdot \text{LP}_{\text{OPT}}$; here LP_{OPT} is the cost of the optimal integral solution to the NONMETRICFACILITYLOCATION instance.

Algorithm 3 LEARNORCOVERNMFL

```
1:  Let $m' \leftarrow |\{f : c_f \leq \beta\}|$.
2:  Initialize $x_f^0 \leftarrow \frac{\beta}{c_f \cdot m'} \cdot \mathbb{1}\{c_f \leq \beta\}$, and $\mathcal{C}^0 \leftarrow \emptyset$.
3:  for $t = 1, 2 \ldots, n$ do
4:      $v^t \leftarrow t^{th}$ client in the random order, and let $\mathcal{R}^t \leftarrow \emptyset$.
5:      for each facility $f$, with probability $\min(\kappa_{v^t}^{t-1} \cdot x_f^{t-1}/\beta, 1)$ add $\mathcal{R}^t \leftarrow \mathcal{R}^t \cup \{f\}$.
6:      Update $\mathcal{C}^t \leftarrow \mathcal{C}^{t-1} \cup \mathcal{R}^t$.
7:      if $\sum_{f \in \Gamma^{t-1}(v^t)} x_f^{t-1} < 1$ then
8:          For every set $f$, update $x_S^t \leftarrow x_S^{t-1} \cdot \exp\left\{\mathbb{1}\{f \in \Gamma^{t-1}(v^t)\} \cdot \kappa_{v^t}^{t-1}/c_f\right\}$.
9:          Let $Z^t = \langle c, x^t \rangle/\beta$ and normalize $x^t \leftarrow x^t/Z^t$.
10:     else
11:         $x^t \leftarrow x^{t-1}$.
12:     $\mathcal{C}^t \leftarrow \mathcal{C}^t \cup \{f^{t-1}(v^t)\}$ and connect $v^t$ to $f^{t-1}(v^t)$.
```

We maintain the invariant:

Invariant 2. *For all time steps t, it holds that $\langle c, x^t \rangle = \beta$.*

Our main theorem is:

Theorem 4.5.1. *Algorithm 3 is $O(\log mn)$-competitive for* ROFACILITYLOCATION

Let us start by defining notation. Let x^* to be the indicator vector of the facilities in the optimal solution. Let U^t be the clients remaining uncovered at the end of round t (where $U^0 = U$ is the entire client set). Recall again the definition of ρ^t and Φ from (4.3.2) and (4.3.3).

Lemma 4.5.2 (Initial Potential)**.** *The initial potential is bounded as $\Phi(0) = O(\beta \cdot \log(mn))$.*

We now show the potential decreases sufficiently in every round. Define Υ^t to be the event that $\textsc{Opt} \cap \Gamma^{t-1}(v^t) \neq \emptyset$. If event Υ^t does not hold, then $\textsc{Opt}$ pays at least $\kappa_v^{t-1}/2$ to connect client v; the algorithm also pays $O(\kappa_v^{t-1})$ in expectation in every round, so in this case we can charge the cost of this round to $\textsc{Opt}$. Henceforth we focus on the case that event Υ^t does occur.

We bound the decrease of each term in the potential separately.

Lemma 4.5.3 (Change in KL)**.** *The expected change in the weighted KL divergence is*

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \tag{4.5.1}$$

$$\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\frac{e^2-1}{2}\cdot\kappa_v^{t-1}\cdot\min\left(\sum_{f\in\Gamma^{t-1}(v)} x_f,\, 1\right) - \kappa_v^{t-1}\right]. \tag{4.5.2}$$

We emphasize that the expected change in relative entropy in the statement above depends only on the randomness of the arriving uncovered client v^t, not on the randomly chosen facilities $\mathcal{R}^t$.

Proof. We break the proof into cases. Let Λ^t be the event that $\sum_{f\in\Gamma^{t-1}(v^t)} x_f^{t-1} < 1$. If Λ^t does not hold, in Line 11 we set the vector $x^t = x^{t-1}$, so the change in KL divergence is 0. This means that inequality (4.5.2) holds trivially. Henceforth we focus on the case when Λ^t holds.

Recall that the expected change in relative entropy depends only on the arriving uncovered element v^t. Expanding definitions,

$$\begin{aligned}
&\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\mathrm{KL}_c\left(x^* \mid\mid x^t\right) - \mathrm{KL}_c\left(x^* \mid\mid x^{t-1}\right) \mid x^{t-1}, U^{t-1}, \Upsilon^t, \Lambda^t\right]\\
&= \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\sum_S c_f\cdot x_f^*\cdot\log\frac{x_S^{t-1}}{x_f^t}\,\middle|\,\Upsilon^t,\Lambda^t\right]\\
&= \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\langle c, x^*\rangle\cdot\log Z^t - \sum_{f\in\Gamma^{t-1}(v)} c_f\cdot x_f^*\cdot\log e^{\kappa_v^{t-1}/c_f}\,\middle|\,\Upsilon^t,\Lambda^t\right]\\
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\begin{array}{l}\beta\cdot\log\left(\displaystyle\sum_{f\in\Gamma^{t-1}(v)}\frac{c_f}{\beta}\cdot x_f^{t-1}\cdot e^{\kappa_v^{t-1}/c_f} + \sum_{f\notin\Gamma^{t-1}(v)}\frac{c_f}{\beta}\cdot x_f^{t-1}\right)\\ -\displaystyle\sum_{f\in\Gamma^{t-1}(v)}\kappa_v^{t-1}\cdot x_f^*\end{array}\,\middle|\,\Upsilon^t,\Lambda^t\right],
\end{aligned} \tag{4.5.3}$$

where in the last step (4.5.3) we expanded the definition of Z^t, and used $\langle c, x^*\rangle \leq \beta$. Since we condition on Υ^t holding, which means that $\sum_{f\in\Gamma^{t-1}(v)}\kappa_v^{t-1}\cdot x_f^* \geq \kappa_v^{t-1}$, we can further bound (4.5.3) by

$$\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\beta\cdot\log\left(\sum_{f\in\Gamma^{t-1}(v)}\frac{c_f}{\beta}\cdot x_f^{t-1}\cdot e^{\kappa_v^{t-1}/c_f} + \sum_{f\notin\Gamma^{t-1}(v)}\frac{c_f}{\beta}\cdot x_f^{t-1}\right) - \kappa_v^{t-1}\,\middle|\,\Lambda^t\right]$$

$$\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\beta \cdot \log\left(\sum_f \frac{c_f}{\beta}\cdot x_f^{t-1} + \frac{e^2-1}{2}\cdot \sum_{f\in\Gamma^{t-1}(v)} \frac{\kappa_v^{t-1}}{\beta}\cdot x_f^{t-1}\right) - \kappa_v^{t-1} \;\middle|\; \Lambda^t\right], \tag{4.5.4}$$

where we use the approximation $e^y \leq 1 + (e^2-1)\cdot y/2$ for $y \in [0,1]$ (note that κ_v^{t-1} is the cheapest marginal connection cost for v, so for any $f \in \Gamma^{t-1}(v)$ for which $c_{fv} \leq \kappa_v^{t-1}/2$ we have that $c_f \geq \kappa_v^{t-1}/2$ and thus $\kappa_v^{t-1}/c_f \leq 2$). Finally, using Invariant 2, along with the approximation $\log(1+y) \leq y$, we bound (4.5.4) by

$$\begin{aligned}
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\frac{e^2-1}{2}\cdot \sum_{f\in\Gamma^{t-1}(v)} \kappa_v^{t-1}\cdot x_f - \kappa_v^{t-1} \;\middle|\; \Lambda^t\right] \\
&\leq \mathop{\mathbb{E}}_{v\sim U^{t-1}}\left[\frac{e^2-1}{2}\cdot \kappa_v^{t-1}\cdot \min\left(\sum_{f\in\Gamma^{t-1}(v)} x_f,\, 1\right) - \kappa_v^{t-1}\right]. \qquad (4.5.5)
\end{aligned}$$

We have shown the lemma statement when Λ^t holds and when it does not, which completes the proof. □

We move to bounding the change in $\log\rho$.

Lemma 4.5.4 (Change in $\log\rho^t$). *For rounds when v is uncovered on arrival, the expected change in $\log\rho^t$ is*

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\log\rho^t - \log\rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t\right] \leq -\frac{1-e^{-1}}{2\beta}\cdot \mathop{\mathbb{E}}_{u\sim U^{t-1}}\left[\kappa_u^{t-1}\cdot \min\left(\sum_{f\in\Gamma(u)} x_f,\, 1\right)\right].$$

Proof.

$$\begin{aligned}
&\mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\log\rho^t - \log\rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t, v^t = v\right] \\
&= \mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\log\left(1 - \frac{\rho^{t-1}-\rho^t}{\rho^{t-1}}\right) \;\middle|\; U^{t-1}, v^t = v\right] \\
&\leq -\frac{1}{\rho^{t-1}}\cdot \mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\rho^{t-1} - \rho^t \mid U^{t-1}, v^t = v\right]. \qquad (4.5.6)
\end{aligned}$$

Above, (4.5.6) follows from the approximation $\log(1-y) \leq -y$. Expanding the definition of ρ^t from (4.3.2), (4.5.6) is bounded by

$$\begin{aligned}
&\leq -\frac{1}{\rho^{t-1}}\cdot \mathop{\mathbb{E}}_{\mathcal{R}^t}\left[\sum_{u\in U^{t-1}} \frac{\kappa_u^{t-1}}{2}\cdot \mathbb{1}\{\Gamma^{t-1}(u)\cap\mathcal{R}^t \neq \emptyset\} \;\middle|\; U^{t-1}, v^t = v\right] \\
&= -\frac{1}{\rho^{t-1}}\sum_{u\in U^{t-1}} \frac{\kappa_u^{t-1}}{2}\cdot \mathop{\mathbb{P}}_{\mathcal{R}^t}\left(\Gamma^{t-1}(u)\cap\mathcal{R}^t \neq \emptyset \mid U^{t-1}, v^t = v\right) \\
&\leq -\frac{1-e^{-1}}{2\beta}\cdot \kappa_v^{t-1}\cdot \frac{1}{\rho^{t-1}}\sum_{u\in U^{t-1}} \kappa_u^{t-1}\cdot \min\left(\sum_{f\in\Gamma^{t-1}(v)} x_f^{t-1},\, 1\right) \qquad (4.5.7)
\end{aligned}$$

$$= -\frac{1-e^{-1}}{2\beta} \cdot \kappa_v^{t-1} \cdot \frac{|U^{t-1}|}{\rho^{t-1}} \cdot \mathop{\mathbb{E}}_{u \sim U^{t-1}} \left[\kappa_u^{t-1} \cdot \min\left(\sum_{f \in \Gamma^{t-1}(v)} x_f^{t-1},\ 1 \right) \right]. \tag{4.5.8}$$

Step (4.5.7) is due to the fact that each facility f is sampled independently with probability $\min(\kappa_v^{t-1} x_f^{t-1}/\beta,\ 1)$, so the probability any given client $u \in U^{t-1}$ gets at least one facility from $\Gamma^{t-1}(u)$ is

$$1 - \prod_{f \in \Gamma^{t-1}(u)} \left(1 - \min\left(\frac{\kappa_v^{t-1} x_f^{t-1}}{\beta},\ 1 \right) \right) \geq 1 - \exp\left\{ -\min\left(\frac{\kappa_v^{t-1}}{\beta} \sum_{f \in \Gamma^{t-1}(u)} x_f^{t-1},\ 1 \right) \right\}$$
$$\overset{(**)}{\geq} (1-e^{-1}) \cdot \min\left(\frac{\kappa_v^{t-1}}{\beta} \sum_{f \in \Gamma^{t-1}(u)} x_f^{t-1},\ 1 \right).$$

Above, $(**)$ follows from convexity of the exponential. Step (4.5.8) follows since $\kappa_v^{t-1}/\beta \leq 1$. Taking the expectation of (4.5.8) over $v^t \sim U^{t-1}$, and using the fact that $\mathbb{E}_{v \sim U^{t-1}}[\kappa_v^{t-1}] = \rho^{t-1}/|U^{t-1}|$, the expected change in $\log \rho^t$ becomes

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t} \left[\log \rho^t - \log \rho^{t-1} \mid x^{t-1}, U^{t-1}, \Upsilon^t \right]$$
$$\leq -\frac{1-e^{-1}}{2\beta} \cdot \mathop{\mathbb{E}}_{u \sim U^{t-1}} \left[\kappa_u^{t-1} \cdot \min\left(\sum_{f \in \Gamma^{t-1}(u)} x_f^{t-1},\ 1 \right) \right],$$

as desired. □

Proof of Theorem 4.5.1. In every round t, the expected cost of the sampled facilities $\mathcal{R}^t$ is $\kappa_{v^t}^{t-1} \cdot \langle c, x^{t-1} \rangle / \beta = \kappa_{v^t}^{t-1}$ (by Invariant 2). The algorithm pays an additional $\kappa_{v^t}^{t-1}$ in Line 12, and hence the total expected cost per round is at most $2 \cdot \kappa_{v^t}^{t-1}$.

In rounds t for which Υ^t holds, by combining Lemmas 4.5.3 and 4.5.4, and setting the constants $C_1 = 2$ and $C_2 = 2e(e+1)$, we have

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t} \left[\Phi(t) - \Phi(t-1) \mid v^1, \ldots, v^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}, \Upsilon^t \right]$$
$$= \mathop{\mathbb{E}}_{v^t, \mathcal{R}^t} \left[\begin{array}{rl} & C_1 \left(\mathrm{KL}_c(x^* \,\|\, x^t) - \mathrm{KL}_c(x^* \,\|\, x^{t-1}) \right) \\ + & C_2 \cdot \beta \cdot (\log \rho^t - \log \rho^{t-1}) \end{array} \middle| v^1, \ldots, v^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}, \Upsilon^t \right]$$
$$\leq -\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t} \left[2 \cdot \kappa_{v^t}^{t-1} \mid v^1, \ldots, v^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1}, \Upsilon^t \right],$$

which cancels the expected change in the algorithm's cost. Hence we have the inequality

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t} \left[\Phi(t) - \Phi(t-1) + c(\textsc{Alg}(t)) - c(\textsc{Alg}(t-1)) \mid \Upsilon^t, v^1, \ldots, v^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1} \right] \leq 0.$$

Let t^* be the last time step for which $\Phi(t^*) \geq 0$. By applying Lemma 4.9.1 and the bound on the starting potential from Lemma 4.5.2, we have that the total cost paid by the algorithm over all rounds in which Υ^t holds is $O(\beta \cdot \log(mn))$.

In rounds for which Υ^t does not hold, we know that $\textsc{Opt} \cap \Gamma^{t-1}(v^t) = \emptyset$, so $\textsc{Opt}$ connects v^t to a facility $f^*(v^t)$ and pays at least $c_{f^*(v^t)v^t} \geq \kappa_v^{t-1}/2$ to connect v^t. Hence

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t} \left[c(\textsc{Alg}(t)) - c(\textsc{Alg}(t-1)) \mid \overline{\Upsilon^t}, v^1, \ldots, v^{t-1}, \mathcal{R}^1, \ldots, \mathcal{R}^{t-1} \right] \leq 2 \cdot \kappa_v^{t-1} \leq 4 \cdot c_{f^*(v^t)v^t}.$$

Summing over all such rounds, we get that the total cost paid by the algorithm over all rounds in which Υ^t does not hold is $O(\beta)$.

It remains to bound the expected cost paid by the algorithm after time t^*. Since KL divergence is a nonnegative quantity, Φ is negative only when $\rho^t \leq \beta$. The algorithm pays $O(\kappa_{v^t}^{t-1})$ in expectation during rounds t where $v^t \in U^{t-1}$ and 0 during rounds where $v^t \notin U^{t-1}$. Since for each v the backup cost κ_v^t is nonincreasing in t, the expected cost paid by the algorithm after time t^* is therefore bounded above by $\sum_{t>t^*} 2 \cdot \kappa_{v^t}^{t-1} \leq 2 \sum_{u \in U^{t^*}} \kappa_u^{t^*} = 2 \cdot \rho^{t^*} = O(\beta)$. □

4.6 SetCover with a Sample

The idea of *algorithms with predictions* has emerged as a recent trend in modern computer science: can algorithms be made to perform better if they are granted access to some advance insight or advice about the input they are given? In the context of online algorithms, a natural assumption is that the algorithm is allowed to sample from its input before the online sequence begins. We ask what benefit such an assumption gives for SetCover. Specifically:

> *What is competitive ratio is possible for* OnlineSetCover *when the algorithm is allowed to sample an α fraction of the elements before the sequence begins?*

We call this version α-SampleSetCover[1]. We highlight that the adversary is allowed to choose the order of the elements *after* the realization of the randomness from the sampling.

A priori it is not clear that this problem has much to do with ROSetCover. Nevertheless, we give an efficient algorithm for α-SampleSetCover with expected competitive ratio $O(\log(mn)/\alpha)$ via a reduction to the random order setting. This is best possible offline in polynomial time when $m = \text{poly}(n)$ and $\alpha = \Omega(1)$.

Theorem 4.6.1. *Algorithm 4 is a polytime randomized algorithm for* α-SampleSetCover *achieving competitive ratio* $O(\log(mn)/\alpha)$.

It is convenient to first reduce α-SampleSetCover to a slightly different problem in which instead of assuming that the algorithm samples exactly αn of the elements, every element is assigned to the sampled set independently with probability α. By subsampling elements with probability $\alpha/2$ and using standard concentration bounds, it is straightforward to argue that the number of elements sampled this way is less than αn with high probability. Hence, at the expense of a constant multiplicative decrease in α, it suffices to give an algorithm for this modified version.

Definition 4.6.2. *A randomized* OnlineSetCover *algorithm $\mathcal{A}$ is* RO-compliant *with a potential function $\Phi \geq 0$, if for every time step t, if the element v^t is sampled uniformly at random from the uncovered elements U^t, then*

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t}[\Delta^t \Phi + \Delta^t c(\mathcal{A})] \leq 0$$

where $\Delta^t\Phi$, $\Delta^t c(\mathcal{A})$ and $\mathcal{R}^t$ are the change in Φ, the cost paid by algorithm $\mathcal{A}$, and the random coin flips of $\mathcal{A}$ at time t respectively.

From the proof of Theorem 4.3.1, the following is immediate:

[1]A similar question was asked in the context of the secretary problem [KNR20] and online matching [KNR22].

Fact 4.6.3. LEARNORCOVER *is RO-compliant with a potential function* Φ *such that* $\Phi(0) = c(\text{OPT})\log(mn)$.

Algorithm 4 REDUCTION TO RO-COMPLIANT ALGORITHM $\mathcal{A}$

```
1: SAMPLES ← sampled elements of $U$.
2: REST ← $U \backslash$SAMPLES.
3: $\mathcal{C} \leftarrow \emptyset$.
4: for $v \in$ SAMPLES in uniformly random order do
5:     Feed $v$ to $\mathcal{A}$ and add any sets bought to $\mathcal{C}$.
6: Buy all sets of $\mathcal{C}$.
7: for $v \in$ REST in arbitrary order do
8:     if $v$ uncovered by $\mathcal{C}$ then
9:         Buy cheapest set covering $v$.
```

Our main theorem is:

Theorem 4.6.4. *If* ONLINESETCOVER *algorithm* $\mathcal{A}$ *is RO-compliant with a potential* Φ*, then Algorithm 4 with* $\mathcal{A}$ *is* $O(\Phi(0)/\alpha)$*-competitive for* α-SAMPLESETCOVER.

Together with Fact 4.6.3, we immediately get:

Corollary 4.6.5. *There is an* $O(\log(mn)/\alpha)$*-competitive algorithm for* α-SAMPLESETCOVER.

Proof of Theorem 4.6.4. Consider the following imaginary process. Define U^t to be the elements uncovered by the algorithm $\mathcal{A}$ at time t, where initially $U^0 := U$. In step t, sample an element v^t uniformly at random from U^t, then color it red with probability α and blue with probability $1-\alpha$. If element was colored red, feed v^t to algorithm $\mathcal{A}$ and buy any sets $\mathcal{A}$ buys; else if the element v^t was blue, buy the cheapest set covering it.

Observe that the cost of Algorithm 4 is only less than the cost paid during this imaginary process. To see this note that the distribution on the ordered sequences of red elements is the same as the distribution on ordered sequences of the SAMPLES elements, and hence $\mathcal{A}$ pays the same total cost in both. On the other hand in the imaginary process, at time t we pay κ_{v^t} for every blue element v^t not covered by $\mathcal{A}$ by time t; in Algorithm 4 we pay κ_{v^t} for every element not covered by $\mathcal{A}$ *ever*. It remains to bound the cost paid during the imaginary process.

To this end, let $\mathcal{D}^t$ be the sets bought in round t of the imaginary process. Since v^t was uncovered on arrival and is covered by the end of round t, the cost $c(\mathcal{D}^t)$ paid in round t is at least κ_{v^t}. Since this cost is exactly κ_{v^t} when the element v^t is colored blue, we have

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Delta^t c(\mathcal{A}) \mid v^t \text{ red}\right] = \mathop{\mathbb{E}}_{v^t}\left[c(\mathcal{D}^t) \mid v^t \text{ red}\right] \geq \mathbb{E}\left[c(\mathcal{D}^t) \mid v^t \text{ blue}\right].$$

Since $\mathcal{A}$ is RO-compliant with Φ, this implies that

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Delta^t \Phi \mid v^t \text{ red}\right] \leq -\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Delta^t c(\mathcal{A}) \mid v^t \text{ red}\right] \leq -\mathbb{E}\left[c(\mathcal{D}^t)\right].$$

Since elements are colored red with probability α, and Φ does not change in rounds when v^t is colored blue, we get

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Delta^t \Phi\right] = \alpha \mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Delta^t \Phi \mid v^t \text{ red}\right] \leq -\alpha\,\mathbb{E}\left[c(\mathcal{D}^t)\right].$$

Defining $\Phi' = \Phi/\alpha$, we have that $\Phi' \geq 0$, $\Phi'(0) = \Phi(0)/\alpha$, and

$$\mathop{\mathbb{E}}_{v^t, \mathcal{R}^t}[\Delta^t \Phi' + \Delta^t c(\mathcal{A})] \leq 0,$$

which together imply that the total expected cost paid during the imaginary process is $\Phi(0)/\alpha$. $\square$

4.7 Lower Bounds

We turn to showing lower bounds for SETCOVER and related problems in the random order model. The lower bounds for ROSETCOVER are proven via basic probabilistic and combinatorial arguments. We also show hardness for a batched version of ROSETCOVER, which has implications for related problems.

4.7.1 Lower Bounds for ROSETCOVER

We start with information theoretic lower bounds for the RO setting.

Theorem 4.7.1. *The competitive ratio of any randomized fractional or integral algorithm for* ROSETCOVER *is* $\Omega(\log n)$.

Proof. Consider the following instance of ROSETCOVER in which $m = 2^\ell$ and $n = 2^\ell - 1$. Construct the instance recursively in ℓ rounds. Define the sub collection of sets $\mathcal{S}_0 = \mathcal{S}$, i.e. initially all the sets. For each round i from 1 to ℓ, do: (a) create $2^{\ell-i}$ new elements and add each to every set in $\mathcal{S}_{i-1}$, and (b) choose $\mathcal{S}_i$ to be a uniformly random subcollection of $\mathcal{S}_{i-1}$ of size $|\mathcal{S}_{i-1}|/2$.

By Yao's principle, it suffices to lower bound the cost of any fixed deterministic algorithm A that maintains a monotone LP solution in random order. Let x^t be the fractional solution of A at the end of round t. Assume A is lazy in the sense that in every round and for every coordinate x_S that is incremented in that round, setting that coordinate to $x_S - \epsilon$ is infeasible for all $\epsilon > 0$ with respect to the elements observed up until and including this round.

We refer to the elements added in round i as the type i elements. Let $\mathfrak{g}(i)$ be the event that at least one element of that type arrives in random order before any element of type j for $j > i$. Note that $\mathbb{P}(\mathfrak{g}(i)) = 2^{l-i}/(2^{l-i+1} - 1) > 1/2$ for all i.

Also define $c(A, i)$ be the cost paid by the algorithm to cover the first element of type i that arrives in random order (potentially 0). Conditioned on $\mathfrak{g}(i)$ holding, we claim the expected cost $c(A, i)$ is at least $1/2$. To see this, first suppose that i is the first type for which $\mathfrak{g}(i)$ holds; then this is the first element to arrive, and so $\sum_{S \in \mathcal{S}_0} x_S^t = 0$ beforehand and $\sum_{S \in \mathcal{S}_i} x_S^t = 1$ afterward, and so $c(A, i) = 1 \geq 1/2$.

Otherwise, let $k < i$ be the last type for which $\mathfrak{g}(k)$ held. Since A is lazy, at the time t before the first element of type i arrived, we had $\sum_{S \in \mathcal{S}_k} x_S^t = 1$. By the construction of the instance and the fact that $i > k$, the collection $\mathcal{S}_i$ consists of a uniformly random subset of $\mathcal{S}_k$ of size at most $|\mathcal{S}_k|/2$. Deferring the random choice of this collection $\mathcal{S}_i$ until this point, we have that $\mathbb{E}[\sum_{S \in \mathcal{S}_i} x_S^t] \leq 1/2$. Hence $\mathbb{E}[c(A, i) \mid \mathfrak{g}(i)] \geq 1/2$.

To conclude, the optimal solution consists of the vector that has 1 on the one coordinate S containing all the elements and 0 elsewhere, whereas

$$\mathbb{E}[c(A)] \geq \sum_i \mathbb{P}\left(\mathfrak{g}(i)\right) \cdot \mathbb{E}[c(A,i) \mid \mathfrak{g}(i)] > \frac{\ell}{4} = \Omega(\log n).$$

Since the optimum is integral, the competitive ratio lower bound also holds for integral algorithms. $\square$

We emphasize that this set system has a VC dimension of 2, which rules out improved algorithms for set systems of small VC dimension in this setting.

Theorem 4.7.2. *The competitive ratio of any randomized algorithm for* ROSETCOVER *is* $\Omega\left(\frac{\log m}{\log\log m}\right)$ *even when* $m \gg n$*.*

The following proof uses the construction in Proposition 4.2 of [AAA$^+$09]; they take the product of this construction with another to show a stronger bound for the adversarial order setting. It is a simple observation that the part of the construction we use gives a bound even in random order, but we include the proof for completeness.

Proof. Given integer parameter r, let $\mathcal{S} = \binom{[10r^2]}{r}$ be the set of all subsets of $[10r^2]$ of size r. The adversary chooses U to be a random subset of $[10r^2]$ of size r and reveals it in random order.

By Yao's principle, it again suffices to bound the performance of any deterministic algorithm A, and we may again assume that A is lazy. By deferring randomness, the adversary is equivalent to one which randomly selects an element r times without replacement from $[10r^2]$. Since the algorithm selects at most r sets each of size at most r, the number of elements of $[10r^2]$ covered by A is at most $[r^2]$, and hence every element chosen by the adversary has probability at least $4/5$ of being uncovered on arrival. Hence the algorithm selects at least $4r/5$ sets in expectation, whereas OPT consists of the single set covering the r elements of the adversary, so the competitive ratio of A is $\Omega(r)$. The claim follows by noting that $r = \Omega(\log m / \log\log m)$. $\square$

4.7.2 Performance of [BN09b] in Random Order

In this section we argue that the algorithm of [BN09b] has a performance of $\Omega(\log m \log n)$ for ROSETCOVER in general. One instance demonstrating this bound is the so called upper triangular instance $\Delta = (U_\Delta, \mathcal{S}_\Delta)$ for which $n = m$ and which we now define. Let the sets $\mathcal{S}_\Delta = \{S_1, \ldots, S_n\}$ be fixed. Choose a random permutation $\pi \in S_n$. Then for every $i = 1, \ldots, m$, let $S_{\pi(i)} = \{n - i + 1, \ldots, n\}$, since it will be convenient for elements to appear in Fig. 4.1 in descending order.

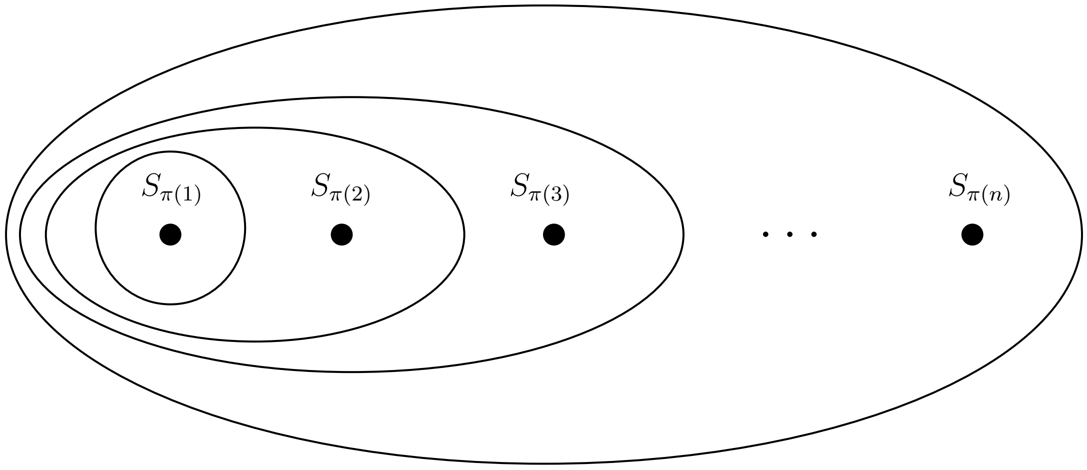


Figure 4.1: Tight instance for [BN09b] in random order.

Claim 4.7.3. *[BN09b] is $\Omega(\log m \log n)$-competitive on the instance $\Delta = (U_\Delta, \mathcal{S}_\Delta)$ in RO.*

Proof. The final solution $\mathcal{T}$ output by [BN09b] on this instance is equivalent to that of the following algorithm which waits until the end of the sequence to buy all of its sets. Maintain a (monotone) LP solution x whose coordinates are indexed by sets. Every time an uncovered element i arrives, for all $j \geq i$ increase the weights of all sets $x_{\pi(j)}$ uniformly until $\sum_{j \geq i} x_{\pi(j)} = 1$. Finally, at the end of the element sequence, sample each set with probability $\min(\log n \cdot x_S, 1)$. We claim that this produces a solution of cost $\Omega(\log^2 n)$ in expectation on the instance above when elements are presented in random order, whereas the optimal solution consists of only the one set S_1.

Let X_i be the event that i arrives before any element j with $j < i$. This corresponds to i appearing in the fewest sets of any element seen thus far; we call such an element *leading*. Let $k(i) = \min\{k \mid X_k,\ k > i\}$ be the most recent leading index before i, if one exists. Note that if X_i occurs and $k(i) = k$, then the expected increase in the size of the final solution due to X_i is exactly $i \cdot \left(\min\left(\frac{\log n}{i}, 1\right) - \min\left(\frac{\log n}{k}, 1\right)\right)$. (If X_i occurs but $k(i)$ does not exist, then i is the first leading element and so the expected cost increase in the final solution is $i \cdot \min\left(\frac{\log n}{i}, 1\right)$.)

What is $\mathbb{P}(X_i,\ k(i) = k)$ for some $i < k$? It is precisely the probability that the random arrival order induces an order on only the elements $k, k-1, \ldots, i, \ldots, 1$ which puts k first and i second, which is $1/(k(k-1))$.

Thus the total expected size of the final solution $\mathcal{T}$ can be bounded by

$$\begin{aligned}
\mathbb{E}[|\mathcal{T}|] &= \sum_{i=1}^{n-1} \sum_{k=i+1}^{n} \mathbb{P}(X_i,\ k(i) = k) \cdot i \left(\min\left(\frac{\log n}{i}, 1\right) - \min\left(\frac{\log n}{k}, 1\right)\right) \\
&\geq \sum_{i=1}^{n-1} \sum_{k=i+1}^{n} \frac{1}{k(k-1)} \cdot i \left(\min\left(\frac{\log n}{i}, 1\right) - \min\left(\frac{\log n}{k}, 1\right)\right) \\
&\geq \log n \cdot \sum_{i=\log n}^{n-1} \sum_{k=i+1}^{n} \frac{1}{k(k-1)} \cdot i \left(\frac{1}{i} - \frac{1}{k}\right) \\
&\geq \log n \cdot \Omega(\log n - \log\log n) \\
&= \Omega(\log^2 n).
\end{aligned}$$

□

4.7.3 Lower Bounds for Extensions

We study lower bounds for several extensions of ROSETCOVER. Our starting point is a lower bound for the batched version of the problem. Here the input is specified by a set system $(U, \mathcal{S})$ as before, along with a partition of U into batches $B_1, B_2, \ldots B_b$. For simplicity, we assume all batches have the same size s. The batches are revealed one-by-one in their entirety to the algorithm, in uniform random order. After the arrival of a batch, the algorithm must select sets to buy to cover all the elements of the batch.

Using this lower bound, we derive as a corollary a lower bound for the random order version of SUBMODULARCOVER from Chapter 3. It is tempting to use the method of Theorem 4.3.1 to improve the competitive ratio of $O(\log m \log(T \cdot f^{(T)}(\mathcal{N})/f_{\min}))$ in RO. We show that removing a log from the bound is not possible in general.

Theorem 4.7.4. *The competitive ratio of any polynomial-time randomized algorithm on batched* ROSETCOVER *with b batches of size s is $\Omega(\log b \log s)$ unless* $\mathsf{NP} \subseteq \mathsf{BPP}$.

We follow the proof of [Kor04, Theorem 2.3.4], which demonstrates that there is no randomized $o(\log m \log n)$-competitive polynomial-time algorithm for (adversarial order) ONLINESETCOVER unless $\mathsf{NP} \subseteq \mathsf{BPP}$. We adapt the argument to account for random order.

Consider the following product of the upper triangular instance from Section 4.7.2 together with an arbitrary instance of (offline) SETCOVER H. In particular, take $\Delta = (U_\Delta, \mathcal{S}_\Delta)$ to be the upper triangular instance on N elements, and let $H = (U_H, \mathcal{S}_H)$ denote a SETCOVER instance, where $U_H = [N']$ and $|\mathcal{S}_H| = M'$. Note that Δ is a random instance, where the randomness is over the choice of label permutation π. Define product instance $\Delta \times H = (U_{\Delta\times H}, \mathcal{S}_{\Delta\times H})$ by

$$\begin{aligned} U_{\Delta\times H} &= \{(i,j) \in U_\Delta \times U_H\} \\ \mathcal{S}_{\Delta\times H} &= \{S_{ij} = S_i \times S_j : (S_i, S_j) \in \mathcal{S}_\Delta \times \mathcal{S}_H\}. \end{aligned}$$

Observe that $|U_{\Delta\times H}| = NN'$ and $|\mathcal{S}_{\Delta\times H}| = NM'$.

Each copy of H is a batch of this instance, so that batches of elements $B_1, \ldots, B_N$ are given by $B_i = \{(i,j) : j \in U_H\}$. Thus the parameters of the batched ROSETCOVER instance are $b = N$ and $s = N'$. These batches B_i will arrive in uniformly random order according to some permutation $\sigma \in S_N$. For a randomized algorithm A, let

$$C(A(\Delta \times H)) := \mathop{\mathbb{E}}_{\substack{\sigma,\pi\in S_N \\ \mathcal{R}}} [c(A(\Delta \times H))]$$

denote the expected cost of A on the instance $\Delta \times H$, where $\mathcal{R}$ denotes the randomness of A.

We begin by establishing an information-theoretic lower bound, whose proof we defer to Section 4.9.

Lemma 4.7.5. *Let A be a randomized algorithm for batched* ROSETCOVER. *Then on the instance $\Delta \times H$,*

$$C(A(\Delta \times H)) \geq \frac{1}{2} \cdot |\textsc{Opt}(H)| \cdot \log N.$$

We also require the following theorem of [RS97]:

Lemma 4.7.6. *There exists a polynomial-time reduction from* SAT *to* SETCOVER *that, given a formula ψ, produces a* SETCOVER *instance H^ψ with N' elements and M' sets for which*

- *$M' = N'^{\alpha}$ for some constant α,*
- *if $\psi \in \mathsf{SAT}$ then $\textsc{Opt}(H^{\psi}) = K(N')$,*
- *if $\psi \notin \mathsf{SAT}$ then $\textsc{Opt}(H^{\psi}) \geq c \cdot \log N' \cdot K(N')$,*

for some polynomial-time-computable K and constant $c \in (0,1)$.

We are ready to show the lower bound.

Proof of Theorem 4.7.4. Suppose that there is some polynomial-time randomized algorithm A with competitive ratio at most $c/4 \cdot \log b \log s$ for batched ROSetCover (recall that c is the constant given in Lemma 4.7.6 above). Then, we argue, the following is a BPP algorithm deciding SAT.

Given a formula ψ, we first reduce it to an instance of batched ROSetCover: first feed the formula through the reduction in Lemma 4.7.6 to get the instance H^{ψ}, then create the batched ROSetCover instance defined by $\Delta \times H^{\psi}$. Finally, run A on $\Delta \times H^{\psi}$ a number $W = \mathrm{poly}(n)$ times, and let $\overline{C}$ be the empirical average of $c(A(\Delta \times H^{\psi}))$ over these runs. If $\overline{C} \geq 3c/8 \cdot \log b \log s \cdot K(N')$, output $\psi \in \mathsf{SAT}$, else output $\psi \notin \mathsf{SAT}$. It suffices to argue that this procedure answers correctly with high probability.

Claim 4.7.7. *If $\psi \in \mathsf{SAT}$, then $C(A(\Delta, H^{\psi})) \leq \frac{c}{4} \cdot \log b \log s \cdot K(N')$.*

Proof. By assumption, A has the guarantee

$$\begin{aligned} C(A(\Delta \times H^{\psi})) &\leq \frac{c}{4} \log b \log s \cdot |\textsc{Opt}(\Delta \times H^{\psi})| \\ &= \frac{c}{4} \cdot \log b \log s \cdot K(N'). \end{aligned}$$ □

Claim 4.7.8. *If $\psi \notin \mathsf{SAT}$, then $C(A(\Delta, H^{\psi})) \geq \frac{c}{2} \cdot \log b \log s \cdot K(N')$.*

Proof. By Lemma 4.7.5, the performance of A is lower bounded as

$$\begin{aligned} C(A(\Delta \times H^{\psi})) &\geq \frac{1}{2} H_N \cdot |\textsc{Opt}(H^{\psi})| \\ &\geq \frac{1}{2} \log N \cdot c \cdot \log(N') \cdot K(N') \\ &\geq \frac{c}{2} \cdot \log b \log s \cdot K(N'). \end{aligned}$$ □

To complete the proof, note that $c(A(\Delta \times H^{\psi})) \in [0, NM']$. Setting $W = \mathrm{poly}(n)$ sufficiently high, by a Hoeffding bound, the estimate $\overline{C}$ concentrates to within $\frac{c}{8} \cdot \log b \log s \cdot K(N')$ of $C(A(\Delta, H^{\psi}))$ with high probability, in which case the procedure above answers correctly. □

We now use Theorem 4.7.4 to derive lower bounds for ROSubmodularCover.

Corollary 4.7.9. *The competitive ratio of any polynomial-time randomized algorithm against ROSubmodularCover is $\Omega(\log m \cdot \log(f^{(T)}(\mathcal{N})/f_{\min}))$ unless $\mathsf{NP} \subseteq \mathsf{BPP}$.*

Proof. Batched ROSetCover is a special case of online SubmodularCover in which f_i is the coverage function of block i. In this case the parameter $f^{(T)}/f_{\min} = n$, so the statement follows by applying Theorem 4.7.4 with $b = s = \sqrt{n}$ (and remembering that $m = \mathrm{poly}(n)$). □

4.8 Conclusion

In this work we introduce LEARNORCOVER as a method for solving ROSETCOVER and ROCIP with competitive ratio nearly matching the best possible offline bounds. On the other hand we prove nearly tight *information theoretic* lower bounds in the RO setting. We also show lower bounds and separations for several generalizations of ROSETCOVER.

We leave as an interesting open question whether it is possible to extend the technique to covering IPs *with box constraints*, and beyond that to the even more general SUBMODULARCOVER problem. Another question is whether the result for NONMETRICFACILITYLOCATION extends to GROUPSTEINERTREE. In particular, can one get an $O(\log^3 n)$ competitive algorithm when the terminals are revealed in random order[2]? Finally, we show an $O(\log(mn)/\alpha)$ competitive ratio for α-SAMPLESETCOVER, but what is the right dependence on α? We conjecture that $O(\log(mn)\log(1/\alpha))$ is possible.

We hope our method finds uses elsewhere in online algorithms for RO settings. In Section 4.12 we discuss suggestive connections between our technique and other methods in online algorithms, namely projection based algorithms and stochastic gradient descent.

4.9 Deferred Proofs

In this chapter we use several potential function arguments, and the following simple lemma.

Lemma 4.9.1 (Expected Potential Change Lemma)**.** *Let* ALG *be a randomized algorithm for* ROSETCOVER *and let Φ be a potential which is a function of the state of the algorithm at time t. Let $c(\text{ALG}(t))$ be the cost paid by the algorithm up to and including time t. Let $\mathcal{R}^t$ and v^t respectively be the random variables that are the random decisions made by the algorithm in time t, and the random element that arrives in time t. Suppose that for all rounds t in which the algorithm has not covered the entire ground set at the beginning of the round, the inequality*

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[\Phi(t) - \Phi(t-1) + c(\text{ALG}(t)) - c(\text{ALG}(t-1)) \;\middle|\; v^1,\ldots,v^{t-1},\mathcal{R}^1,\ldots,\mathcal{R}^{t-1}\right] \le 0.$$

holds. Let t^* be the last time time step such that $\Phi(t^*) \ge 0$. Then the expected cost of the algorithm, $c(\text{ALG}(t^*))$ can be bounded by

$$\mathbb{E}[c(\text{ALG}(t^*))] \le \Phi(0).$$

Proof. Let $\mathcal{T}$ be the set of rounds for which $\Phi > 0$ at the beginning of the round.

Define the stochastic process:

$$X^t := \begin{cases} \Phi(t) + c(\text{ALG}(t)) & \text{if } t \in \mathcal{T}, \\ X^{t-1} & \text{otherwise.} \end{cases}$$

By assumption, this is a supermartingale with respect to $((v^t,\mathcal{R}^t))_t$; that is,

$$\mathop{\mathbb{E}}_{v^t,\mathcal{R}^t}\left[X^{t+1} \;\middle|\; v^1,\ldots,v^{t-1},\mathcal{R}^1,\ldots,\mathcal{R}^{t-1}\right] \le X^t$$

[2]We refer the reader to [AAA+06] for a full description of online GROUPSTEINERTREE.

for all t. By induction we have that for all $t > 0$

$$\mathop{\mathbb{E}}_{\substack{v^1,\ldots,v^t \\ \mathcal{R}^1,\ldots,\mathcal{R}^t}} [X^t] \leq X^0,$$

so in particular,

$$\mathbb{E}[\Phi(t^*)] + \mathbb{E}[c(\textsc{Alg}(t^*))] \leq \Phi(0).$$

The claim follows since the leftmost term is nonnegative by assumption. $\square$

Fact 4.3.3. *Every pure covering LP of the form* $\min_{x\geq 0}\{\langle c, x\rangle : Ax \geq 1\}$ *for* $c \geq 0$ *and* $a_{ij} \in [0,1]$ *with optimal value less than* β *has an optimal solution* x^* *which is supported only on columns* j *such that* $c_j \leq \beta$*.*

Proof. Suppose otherwise and let x^* be an optimal LP solution. Let j' be a coordinate for which $c_{j'} > \beta$ and $x^*_{j'} > 0$. Then define the vector x' by

$$x'_j = \begin{cases} 0 & \text{if } j = j' \\ \frac{x^*_j}{1-x'^*_j} & \text{otherwise,} \end{cases}$$

First of all, note that $x^*_{j'} < 1$ since $x^*_{j'}c_{j'} \leq \sum_j c_j x^*_j \leq \beta < c_{j'}$, and so $x' \geq 0$. To see that x' is feasible for each constraint $\langle a_i, x\rangle \geq 1$,

$$\langle a_i, x^*\rangle = (1 - x^*_{j'})\langle a_i, x'\rangle + a_{ij'}x^*_{j'} \geq 1 \quad \text{so} \quad \langle a_i, x'\rangle \geq \frac{1}{1 - x^*_{j'}}(1 - a_{ij'}x^*_{j'}) \geq 1,$$

since $a_{ij'} \in [0,1]$. Finally, observe that x' costs strictly less than x^*, since

$$\langle c, x'\rangle = \frac{\langle c, x^*\rangle - x^*_{j'}c_{j'}}{1 - x^*_{j'}} < \frac{\langle c, x^*\rangle - x^*_{j'}\langle c, x^*\rangle}{1 - x^*_{j'}} = \langle c, x^*\rangle.$$

This contradicts the optimality of x^*, and so the claim holds. $\square$

Fact 4.4.4. *Given probabilities* p_j *and coefficients* $b_j \in [0,1]$*, let* $W := \sum_j b_j \operatorname{Ber}(p_j)$ *be the sum of independent weighted Bernoulli random variables. Let* $\Delta \geq \gamma = (e-1)^{-1}$ *be some constant. Then*

$$\mathbb{E}[\min(W, \Delta)] \geq \alpha \cdot \min(\mathbb{E}[W], \Delta),$$

for a fixed constant α *independent of the* p_j *and* b_j*.*

Proof. We first consider the case when $\mathbb{E}[W] \geq \Delta/3$. By the Paley-Zygmund inequality, noting that $\mathbb{E}[W] = \sum_j b_j p_j$ and so $\sigma^2 = \sum_j p_j(1-p_j)b_j^2 \leq \mathbb{E}[W]$, we have

$$\mathbb{P}(W \geq \Delta/6) \geq \mathbb{P}(W \geq \mathbb{E}[W]/2) \geq \frac{1}{4}\cdot\frac{\mathbb{E}[W]^2}{\mathbb{E}[W]^2 + \sigma^2} \geq \frac{1}{4}\cdot\frac{\mathbb{E}[W]}{1 + \mathbb{E}[W]} \geq \frac{1}{4}\cdot\frac{\Delta/3}{1 + \Delta/3} \geq 1/28,$$

since $\Delta \geq \gamma \geq 1/2$ by assumption. This implies the claim because in this case

$$\mathbb{E}[\min(W, \Delta)] \geq \frac{\Delta}{6}\cdot\mathbb{P}(W \geq \Delta/6) \geq \frac{\Delta}{168} \geq \frac{1}{168}\cdot\min(\mathbb{E}[W], \Delta).$$

Otherwise $\mathbb{E}[W] < {}^{\Delta}/_{3}$. Let $\mathcal{R}$ denote the random subset of j for which $\mathrm{Ber}(p_j) = 1$ in a given realization of W, and let $\mathcal{K}$ be the random subset of the j which is output by the $({}^1/_3, {}^1/_3)$-contention resolution scheme for knapsack constraints when given $\mathcal{R}$ as input, as defined in [CVZ14, Lemma 4.15]. The set $\mathcal{K}$ has the properties that (1) (over the randomness in $\mathcal{R}$) every j appears in $\mathcal{K}$ with probability at least $p_j/3$, (2) $\sum_{j\in\mathcal{K}} a_{ij} < \Delta$, and (3) $\mathcal{K} \subseteq \mathcal{R}$. Hence in this case

$$\mathbb{E}[\min{(W, \Delta)}] \geq \mathop{\mathbb{E}}_{\mathcal{K}}\left[\sum_{j\in\mathcal{K}} a_{ij}\right] \geq \frac{\mathbb{E}[W]}{3} = \frac{1}{3}\cdot\min(\mathbb{E}[W], \Delta). \qquad \square$$

Therefore the claim holds with $\alpha = {}^1/_{168}$.

Lemma 4.7.5. *Let A be a randomized algorithm for batched* ROSETCOVER. *Then on the instance $\Delta \times H$,*

$$C(A(\Delta \times H)) \geq \frac{1}{2}\cdot|\textsc{Opt}(H)|\cdot\log N.$$

Proof. By Yao's principle, it suffices to bound the expected performance of any deterministic algorithm A over the randomness of the instance. Here the performance is in expectation over the random choice of instance as well as the random order of batch arrival.

The randomness in the input distribution $\Delta \times H$ is over the set labels, given by the random permutation $\pi \in S_N$. For convenience, we instead equivalently imagine the set labels in Δ are fixed, and that π is a random permutation over batch labels, so that the label of batch i is $l_i = \pi(i)$. This means that for a fixed realization of π, for any two sets $S_{lj}, S_{l'j'} \in \mathcal{S}_{\Delta\times H}$, even before any batches have arrived A can determine whether $l = l'$, but because $\pi \sim S_N$ uniformly at random, A cannot determine which batches i the sets S_{lj} and $S_{l'j'}$ intersect. Without loss of generality, we assume that A is lazy, in that for each batch it only buys sets which provide marginal coverage.

The (offline) instance H has some optimal cover $\{S_1^*, \dots, S_k^*\} := \textsc{Opt}(H)$. Let $\mathcal{S}^* := \{S_{lj} = S_l \times S_j : S_l \in \mathcal{S}_\Delta, S_j \in \textsc{Opt}(H)\}$ denote the sets in $\mathcal{S}_{\Delta\times H}$ which project down to sets in $\textsc{Opt}(H)$. We argue that it suffices to consider a deterministic lazy algorithm A^* which only buy sets in $\mathcal{S}^*$. Let $c(A(\pi, \sigma))$ denote the cost of A on batch label permutation π, when the batch arrival order is $\sigma \in S_N$. For every feasible A we argue that there is some A^* which buys only sets in $\mathcal{S}^*$, is feasible for every batch $\sigma(i)$ upon arrival, and buys at most as many sets as A for any $\pi \in \sigma_N$, so that

$$c(A^*(\pi, \sigma)) \leq c(A^*(\pi, \sigma)). \tag{4.9.1}$$

This will in turn imply that

$$\mathop{\mathbb{E}}_{\sigma,\pi\sim S_N}[c(A^*(\pi, \sigma))] \leq \mathop{\mathbb{E}}_{\sigma,\pi\sim S_N}[c(A(\pi, \sigma))], \tag{4.9.2}$$

and so it will suffice to lower bound the performance of any A^* in this restricted class.

Given A, we will construct an A^* which satisfies (4.9.1). But first, some notation. For each batch arrival order $\sigma(1), \dots, \sigma(N)$, suppose that in round i upon the arrival of $B_{\sigma(i)}$, A buys the sets $\mathcal{C}_i \subseteq \mathcal{S}_{\Delta\times H}$. These $\mathcal{C}_1, \dots, \mathcal{C}_i$ together cover each $B_{\sigma(i)}$ upon its arrival, and $c(A, \sigma) = \sum_i |\mathcal{C}_i|$. Let $\overline{\mathcal{C}_i} := \{S \in \bigcup_{i'\leq i} \mathcal{C}_{i'} : S \cap B_{\sigma(i)} \neq \emptyset\}$ be the collection of sets which A uses to cover $B_{\sigma(i)}$. We say a set $S_{lj} \in \mathcal{S}_{\Delta\times H}$ is *live* in round i if it intersects with all batches $B_{\sigma(1)}, \dots, B_{\sigma(i)}$ which have arrived so far. All sets are live to start, and once a set is not-live it will never be live again; note that the liveness of S_{lj} in round i is a property of its label l and the batch order σ. In each round i we will *match* each set $S^* \in \mathcal{C}_i^*$ which A^* buys with some set $S \in \bigcup_{i' \leq i} \mathcal{C}_{i'}$. We will maintain that at most one S^* is matched to each S, and that a matched pair of sets is never unmatched.

Let A^* operate by running A in the background. For each round i with incoming batch $B_{\sigma(i)}$, if A^* already covers $B_{\sigma(i)}$ upon arrival then A^* does nothing. Otherwise for each $j^* \in \textsc{Opt}(H)$ for which A^* has not already bought a copy which is live in round i, A^* identifies an unmatched set $S_{lj} \in \overline{\mathcal{C}}_i$ which A is using to cover $B_{\sigma(i)}$, buys the set S_{lj^*} (with the same label), and matches S_{lj^*} with S_{lj}.

It is immediate that $c(A^*, \sigma) \leq c(A, \sigma)$, since A^* matches every set which it buys to a set which A buys. Therefore we need only show that A is feasible in each round i; that is, it never runs out of unmatched sets.

To see this, first note that projecting $\overline{\mathcal{C}}_i$ onto H gives a feasible cover, and so $|\overline{\mathcal{C}}_i| \geq k$. In particular, this means that A^* succeeds in the first round $i = 1$. Next observe that in any round $i > 1$, algorithm A^* has bought at most one S_{lj^*} which is live for each given $S_{j^*} \in \textsc{Opt}(H)$. This is because it only buys S_{lj^*} for j^* for which it does not currently have a live copy, and no sets go from not-live to live. Also note that any set which shares a label with some $S_{lj} \in \overline{\mathcal{C}}_i$ is live in round i. Therefore any S_{lj^*} matched to $S_{lj} \in \overline{\mathcal{C}}_i$ at the beginning of round i are live, since A^* ensures that matched sets share labels. Let Γ_i be the collection of these S_{lj^*}, and let $t := |\Gamma_i|$. Since A^* maintains at most one live set for each $S_{j^*} \in \textsc{Opt}(H)$ at once, the j^* for $S_{lj^*} \in \Gamma_i$ are distinct. Therefore to cover $B_{\sigma(i)}$, A^* must buy sets S_{lj^*} for the $k - t$ remaining $S_{j^*} \in \textsc{Opt}(H)$ not represented in Γ_i with live labels l. Fortunately there are $|\overline{\mathcal{C}}_i| - t \geq k - t$ unmatched sets with live labels which A has bought for A^* to choose from, and so A^* never gets stuck.

Therefore A^* is feasible for every round i, and so (4.9.1) holds.

We now lower-bound the performance of A^*. Since A^* is lazy and $\textsc{Opt}(H)$ is a minimal cover for H, for each arriving batch $B_{\sigma(i)}$ the algorithm A^* buys exactly one S_{lj^*} corresponding to each $S_{j^*} \in \textsc{Opt}(H)$ for which it does not already have a live copy. Therefore we can analyze the expected number of copies of S_{j^*} which A^* buys over the randomness of the batch arrival order for each $S_{j^*} \in \textsc{Opt}(H)$ independently.

Fix some $S^* = S_{j^*} \in \textsc{Opt}(H)$, and let C_N denote the expected number of copies of S^* which A^* buys, where the expectation is taken over the batch arrival order $\sigma \sim S_N$. In the first round $i = 1$, A^* buys some S_{lj^*} with label l, and uses this copy until the first batch arrives for which l is no longer live; let $P(l)$ denote the number of batches $B_{\sigma(1)} \ldots, B_{\sigma(P(l))}$ for which l remains live. Once l is no longer live, A^* must choose another copy $S_{l'j^*}$ for one of the remaining live l'. This is a sub-instance of the problem it faced at $i = 1$.

We will prove that $C_N \geq H_N/2$ by induction on N (where here H_N denotes the N^{th} harmonic number). By definition we have that $C_1 = 1$, and so this holds for $N = 1$. Now assume the claim $C_n \geq H_n/2$ holds for all integers $n \in [0, N-1]$. Using the observations above, we can express C_N by the following recurrence:

$$\begin{aligned} C_N &= 1 + \sum_{j=1}^{N} \mathbb{P}\left(P(l) = j\right) \cdot C_{N-j} \\ &= 1 + \frac{1}{N} \sum_{n=0}^{N-1} (H_N - H_n) \cdot C_n, \end{aligned}$$

where we take $H_0 = 0$. By computing C_{N-1} and substituting, we then obtain

$$\begin{aligned} C_N &= \frac{1}{N} + \frac{N-1}{N} C_{N-1} + \frac{1}{N^2} \sum_{n=0}^{N-1} C_n \\ &\geq \frac{1}{N} + \frac{N-1}{2N} \left(H_N - \frac{1}{N} \right) + \frac{1}{2N} (H_N - 1) \\ &\geq \frac{1}{2} H_N, \end{aligned}$$

where here we used that $\sum_{n=0}^{N-1} H_n = N(H_N - 1)$. Since $H_N \geq \log N$, we therefore have

$$\mathop{\mathbb{E}}_{\sigma \sim S_N} \left[\left| \left\{ S_{lj^*} \in \bigcup_i \mathcal{C}_i^* \right\} \right| \right] = C_N \geq \frac{1}{2} \log N.$$

Since this analysis holds for each of the k such sets $S_{j^*} \in \textsc{Opt}(H)$, the claim follows from linearity of expectation. □

4.10 Pseudocode

In this section we give pseudocode for secondary algorithms in this chapter.

4.10.1 The Exponential-Time SetCover Algorithm

First we give the algorithm from Section 4.2.

Algorithm 5 SimpleLearnOrCover

```
1: Initialize $\mathfrak{T}^0 \leftarrow \binom{S}{k}$ and $\mathcal{C}^0 \leftarrow \emptyset$.
2: for $t = 1, 2 \ldots, n$ do
3:     $v^t \leftarrow t^{th}$ element in the random order.
4:     if $v^t$ uncovered then
5:         Choose $\mathcal{T} \sim \mathfrak{T}^{t-1}$ uniformly at random, and choose $T \sim \mathcal{T}$ uniformly at random.
6:         Add $\mathcal{C}^t \leftarrow \mathcal{C}^{t-1} \cup \{S, T\}$ for any choice of $S$ containing $v^t$.
7:     Update $\mathfrak{T}^t \leftarrow \{\mathcal{T} \in \mathfrak{T}^{t-1} : v^t \in \bigcup \mathcal{T}\}$.
```

4.10.2 The SetCover Algorithm for Unit Costs

Next, we give a slightly simplified version of algorithm from Section 4.3 in the special case of unit costs. We give it here to illustrate the essential simplicity of our algorithm in the unit-weight case. (Much of the complication comes from managing the non-uniform set costs.)

Algorithm 6 UNITCOSTLEARNORCOVER

```
1: Initialize $x_S^0 \leftarrow \frac{1}{m}$, and $\mathcal{C}^0 \leftarrow \emptyset$.
2: for $t = 1, 2 \ldots, n$ do
3:     $v^t \leftarrow t^{th}$ element in the random order.
4:     if $v^t$ not already covered then
5:         Sample one set $R^t$ from distribution $x$, update $\mathcal{C}^t \leftarrow \mathcal{C}^{t-1} \cup \{R^t\}$.
6:         if $\sum_{S \ni v^t} x_S^{t-1} < 1$ then
7:             For every set $S \ni v^t$, update $x_S^t \leftarrow e \cdot x_S^{t-1}$.
8:             Normalize $x^t \leftarrow x^t / \|x^t\|_1$.
9:         else
10:            $x^t \leftarrow x^{t-1}$.
11:        Let $S_{v^t}$ be an arbitrary set containing $v^t$. Add $\mathcal{C}^t \leftarrow \mathcal{C}^t \cup \{S_{v^t}\}$.
```

4.11 Discussion of Claim in [GGL$^+$13]

[GGL$^+$13] writes in passing that SETCOVER in “the random permutation model (and hence any model where elements are drawn from an unknown distribution) [is] as hard as the worst case”.

Our algorithm demonstrates that the instance of [Kor04] witnessing the $\Omega(\log^2 n)$ lower bound in adversarial order can be easily circumvented in random order. Korman’s instance is conceptually similar to our hard instance for the batched case from Section 4.7.3 but with batches shown in order, deterministically.

A natural strategy to adapt the instance to the random order model is to duplicate the elements in the i^{th} batch C^{b-i} times for a constant $C > 1$ (where b is the number of batches). This ensures that elements of early batches arrive early with good probability. Indeed, this is the strategy used for the online Steiner Tree problem in random order (see e.g [GS20]). However, for the competitive ratio lower bound of [Kor04], which is $\log b \log s$, to be $\Omega(\log^2 n)$, the number of batches b and the number of distinct elements per batch s must be polynomials in the number of elements n. In this case the total number of elements after duplication is $n' = O(C^{\text{poly}(n)})$, which degrades the $\Omega(\log b \log s)$ bound to doubly logarithmic in n'.

4.12 Connections to Other Algorithms

In this section we discuss connections between LEARNORCOVER and other algorithms. Whether these perspectives are useful or merely spurious remains to be seen, but regardless, they provide interesting context.

Projection interpretation of [BGMN19]. In [BGMN19] it is shown that the original ONLINE-SETCOVER algorithm of [AAA$^+$09, BN09b] is equivalent to the following. Maintain a fractional solution x. On the arrival of every constraint $\langle a_i, x \rangle \geq 1$, update x to be the solution to the convex program

$$\begin{array}{ll} \operatorname{argmin}_y & \mathrm{KL}\left(y \,||\, x\right) \\ \text{s.t.} & A^{\leq i} y \geq 1 \\ & y \geq 0, \end{array} \tag{4.12.1}$$

where $A^{\leq i}$ is the matrix of constraints up until time i. Finally, perform independent randomized rounding online. The KKT and complementary slackness conditions ensure that the successive fractional solutions obtained this way are monotonically increasing, and in fact match the exponential update rule of [BN09b].

Interestingly, we may view LEARNORCOVER as working with the same convex program but *with an additional cost constraint* (recall that β is our guess for $c(\text{OPT})$).

$$\begin{array}{ll} \operatorname{argmin}_y & \mathrm{KL}\left(y \,||\, x\right) \\ \text{s.t.} & A^{\leq i} y \geq 1 \\ & \langle c, y \rangle \leq \beta \\ & y \geq 0, \end{array} \tag{4.12.2}$$

The extra packing constraint already voids the monotonicity guarantee of (4.12.1) which we argue is a barrier for [BN09b] in Sections 4.7.1 and 4.7.2. Furthermore, instead of computing the full projection, we fix the Lagrange multiplier of the most recent constraint $\langle a_i, x \rangle \geq 1$ to be κ_{v_i} (the cost of the cheapest set containing this last element), the multipliers of the other elements' constraints to 0, and the multiplier of the cost packing constraint such that y is normalized to cost precisely β. Hence we only make a partial step towards the projection, and sample from the new fractional solution regardless of whether it was fully feasible.

Stochastic Gradient Descent (SGD). Perhaps one reason ROSETCOVER is easier than the adversarial order counterpart is that, in the following particular sense, random order grants access to a stochastic gradient.

Consider the unit cost setting. Given fractional solution x, define the function

$$f(x) := \sum_v \max\left(0, 1 - \sum_{S \ni e} x_S\right),$$

in other words the fractional number of elements uncovered by x. Clearly we wish to minimize f, and if we assume that every update to x coincides with buying a set (as is the case in our algorithm), we wish to do so in the smallest number of steps.

The gradient of f evaluated at the coordinate S is

$$[\nabla f]_S = -|S \cap U^t|.$$

On the other hand, conditioning on the next arriving element v being uncovered, the random binary vector χ^v denoting the set membership of v has the property

$$\mathop{\mathbb{E}}_v[\chi^v_S] = \frac{|S \cap U^t|}{|U^t|},$$

meaning χ^v is a scaled but otherwise unbiased estimate of ∇f. Since LEARNORCOVER performs updates to the fractional solution using χ_v, it can be thought of as a form of stochastic gradient descent (more precisely of stochastic mirror descent with entropy mirror map since we use a multiplicative weights update scheme, see e.g. [Bub15]).

One crucial and interesting difference is that SGD computes a gradient estimate at every point to which it moves, whereas our algorithm is only allowed to query the gradient at the vertex of the hypercube corresponding to the sets bought so far. This analogy with SGD seems harder to argue in the non-unit cost setting, where the number of updates to the solution is no longer a measure of the competitive ratio.

Chapter 5

Block-Aware Caching

5.1 Introduction

CACHING (also known as paging) has been extensively studied since the early days of online computation and competitive analysis, establishing itself as a cornerstone problem in this field, see e.g., [ST85, FKL$^+$91, You91, You02, MS91, BBN12a, BBN12b, ACN00, AAK99, BBK99, BFT96, Ira96, Ira02b, Ira02a]. Recent years have witnessed increased activity on non-standard caching models, e.g., elastic caches [GKKP19], caching with time windows [GKP20], caching with dynamic weights [EMR18], caching with machine learning predictions [LV21], and writeback-aware caching [BGHM20, BNT21]. Many of the recent developments in competitive analysis, e.g., the online primal-dual method, projections, and mirror descent [BN09a, BCN14b, BCL$^+$18] are all rooted in online paging. We study here BLOCKAWARECACHING, a non-standard caching model studied recently, as well as its generalizations.

In the (classic) weighted paging problem there is a universe of n pages, a cache that can hold up to k pages, and each page is associated with a weight (fetch cost). At each time step a page is requested; if the requested page is already in cache then no cost is incurred, otherwise the page must be fetched into the cache, incurring a cost equal to its weight. The goal is to minimize the total cost incurred. This problem is well studied and understood, and we briefly mention the main results known for it.

Sleator and Tarjan [ST85], in their seminal paper on competitive analysis, showed that any deterministic algorithm is at least k-competitive, and that LRU (Least Recently Used) is precisely k-competitive for unweighted paging (i.e., all weights are equal). The k-competitive bound was later generalized to weighted paging as well [CL91, You94]. When randomization is allowed, Fiat et al [FKL$^+$91] gave the elegant Randomized Marking algorithm for unweighted paging, which is $\Theta(\log k)$-competitive against an oblivious adversary. For weighted paging, Bansal et al. [BBN12a] gave an $O(\log k)$-competitive randomized algorithm using the online primal-dual framework [BN09a, AAA$^+$09]. It uses a two-step approach. First, a deterministic competitive algorithm is designed for a fractional version of the problem. Then, a randomized online algorithm is obtained by *rounding* the deterministic fractional solution online.

BLOCKAWARECACHING. Real storage systems operate by constructing a hierarchy of memory levels, starting from a very fast and small memory (e.g., an SRAM cache) to a very large and slow memory (e.g., flash or disk). The data items in each level are typically organized in *blocks*, and fetching (or evicting) data items from the same block incurs the same cost as fetching (or evicting) just a single item from the block. Using fetching costs models scenarios in which data is read-only; using eviction costs models scenarios in which data must be written to slow memory upon eviction, and the writing cost dominates the reading cost (see e.g. [BGHM20, BNT21]).

Thus, a natural question is how can one optimize cache performance by taking advantage of granularity changes across different storage hierarchy levels. This question was recently raised by Beckmann et al. [BGM21], who defined the BLOCKAWARECACHING problem, generalizing the classic paging problem, as follows. Given a cache of size k, and a sequence of requests from n pages that are partitioned into given blocks of size $\beta \leq k$, minimize the total cost of fetching (or evicting) from the cache so as to serve the requests[1].

BLOCKAWARECACHING also arises in web and cloud settings, where data items can be aggregated into chunks (i.e., blocks) of data, such that accessing a whole chunk incurs the same cost as accessing just a single item. Consider a distributed cluster of servers, where a common cache of data items is maintained. One such example is the ZFS distributed file system that aggregates different devices into a single storage pool acting as an arbitrary data store. When accessing a server for a specific data item, the main cost paid (e.g., latency) is for accessing the server. The notion of a block of data in this setting corresponds to the largest chunk of data items that can be fetched from (or evicted to) a server, while maintaining that the cost of this operation is dominated by the cost of accessing the server. Web caching is another example of BLOCKAWARECACHING. Consider a content delivery network (CDN) that maintains a cache of data items and suppose the CDN connects to a website so as to access a data item (see, e.g., [HGDS14, SBLL20]). Typically, TCP/IP provides a time window for connecting to the website and accessing the data item. Hence, it might be beneficial to fetch (or evict) many data items that belong to the website, and not just the particular data item that is currently accessed. Thus, the notion of a block of data in this setting corresponds to the maximum number of such data items that can be sent without increasing the travel time.

In the GENERALIZEDCACHING problem [BBN12b, ACER19], pages are associated with both a size and a cost. At any point of time, the sum of the sizes of the pages in the cache cannot exceed the cache size. In the offline setting, GENERALIZEDCACHING is known to be NP-hard, and in the online setting the known competitive factors for GENERALIZEDCACHING [BBN12b, ACER19] match those of weighted CACHING. It is not hard to see that BLOCKAWARECACHING captures generalized GENERALIZEDCACHING as a special case. Replace a page p of size s by a block B of size s containing page p partitioned into unit size "slices". The cost of accessing each slice is equal to the cost of p. Now, a request to page p is replaced by many requests to the slices in B. Thus, an optimal solution to the BLOCKAWARECACHING problem generated has to fetch the full block B into the cache.

Eviction and fetching costs. In classic paging, costs can be associated with either evicting or fetching pages. Clearly, for a given request sequence, optimal eviction and fetching costs of serving the requests can differ by at most an additive constant that only depends on the initial

[1]We note that Beckmann et al. [BGM21] considered BLOCKAWARECACHING only in the fetching cost model.

contents of the cache. However, this is not the case for BLOCKAWARECACHING, as optimal eviction and fetching costs can differ significantly, separating the two cost models. (We provide an example in Section 5.2.) As discussed above, the two cost models are practically motivated for BLOCKAWARECACHING, and we thus study both of them in this chapter. We note that in the eviction cost model we are able to circumvent known lower bounds [BGM21] that hold in the fetching cost model.

5.1.1 Results and Techniques

Observe that if an algorithm is r-competitive for classical paging, then it is at most $\beta \cdot r$-competitive for BLOCKAWARECACHING in both fetching/eviction cost models; the reason is simply that OPT can be simulated by a classical paging algorithm that performs any single batched fetch/eviction in at most β rounds. With this in mind, our goal in this work is to beat this trivial linear dependence on β.

Indeed, for the eviction cost model, we give the first set of algorithms avoiding a trivial multiplicative β overhead over their classical paging counterparts. We also give (h, k)-bicriteria[2] algorithms for both fetching and eviction cost models, which we in turn use to adapt the lower bound of [BGM21] for the fetching cost model to randomized algorithms.

Eviction cost. We start in Section 5.3 with our main contributions: competitive algorithms for the eviction cost model. We show the following theorem.

Theorem 5.1.1. *For the* BLOCKAWARECACHING *problem with eviction cost, there exist:*

- *a k-competitive deterministic online algorithm.*
- *an $O(\log^2 k)$-competitive randomized (integral) online algorithm.*
- *an $O(\log k)$-approximate randomized offline algorithm.*

In fact, we study a more general version than the one introduced in [BGM21] in which every block B may have a separate cost c_B. For this more general weighted setting we get competitive ratios of k, $O(\log k \log(k\Delta))$ and $O(\log(k\Delta))$ for the deterministic online, randomized online, and randomized offline settings respectively (where Δ is the aspect ratio, i.e., the maximum cost ratio between any two blocks).

A first main technical ingredient is a linear programming relaxation for BLOCKAWARECACHING. It is tempting to use a formulation with a variable x_p^t for each page p and time t indicating whether p is present in cache in step t. However, in this case the eviction cost becomes a complicated non-linear function of the x_p^t. Instead, we define variable ϕ_B^t for each block B and time step t indicating whether we evict B at t. This is reminiscent of the linear program for classical paging of [BBN12a] in which every variable represents whether a page is present in cache between two subsequent requests to a page, only that it may now be necessary to evict pages at any point between subsequent requests.

A naïve linear programming formulation has an integrality gap of β (see Section 5.7): this is unsurprising since the naïve LP exhibits this gap even for the special case of generalized paging.

[2]In (h, k) paging, an online algorithm with cache size k competes against an offline cache of size h, where $k > h$.

To get around this, we express feasibility as the constraint that a particular sequence of monotone, submodular functions is maximized. We then make use of the LP relaxations for these submodular set function constraints which we presented in Section 2.4. Our formulation may be viewed as a generalization of the strengthened LP relaxation due to [BBN12b] for GENERALIZEDCACHING, which used the so-called knapsack cover (KC) inequalities. We first use the relaxation to give a k-competitive deterministic online algorithmic in Section 5.3.2.

Next, we develop an $O(\log k)$-competitive fractional algorithm in Section 5.3.3, followed by an $O(\log \Delta k)$-competitive online randomized rounding procedure in Section 5.3.4. Together these imply our $O(\log k \log \Delta k)$-competitive randomized online algorithm. It is natural to try to adapt the continuous online primal-dual framework of [BBN12a, BBN12b]; however, owing to the increased complexity of our LP, there are several technical roadblocks. For one, our formulation now has primal variables corresponding to the eviction of every block at every point in time, and a naïve adaptation of the continuous dynamics of [BBN12b] incurs loss that depends on the length of the request sequence. Nevertheless, we show how to carefully set the rate of increase of primal variables (with respect to the dual rate of increase) to construct a feasible solution with our claimed guarantee. For convenience, we do not present the fractional algorithm as online in the strict sense as we allow it to change decisions made in the past. However, it has the crucial property that it only increases LP variables, which suffices for the online rounding procedure.

For the rounding step, we forego maintaining an explicit distribution over cache states as in previous work [BBN12a, BBN12b, ACER19], since it is unclear how to control the cost of the rebalancing stage when updating the distribution in each time step. Instead, we use the method of random rounding with alterations. We crucially make use of the randomized rounding analysis of Chapter 3 for ONLINESUBMODULARCOVER; interestingly we are able to charge our alteration cost to the fractional *fetching* cost, even though this may be a factor β larger than the eviction cost.

Fetching cost. We turn in Section 5.4 to the fetching cost model, where we show strong lower bounds, implying that the integrality gap of $\Omega(\beta)$ of the natural LP formulation cannot be circumvented. We prove the following theorem for (h, k) block-aware caching[3].

Theorem 5.1.2. *When $k = O(h)$, no randomized online algorithm has competitive ratio better than $\Omega(\beta + \log k)$ for* BLOCKAWARECACHING *with fetching costs.*

Our main idea here is an online deterministic rounding procedure for fractional algorithms that incurs constant blowup in both cache usage and cost. This implies an online derandomization procedure for any randomized algorithm, which in turn strengthens the lower bounds for deterministic algorithms of [BGM21] to apply to randomized algorithms as well. Our lower bound implies that beating the trivial linear dependence on β is *not* possible for the fetching cost model.

Our deterministic rounding procedure immediately implies improved bounds for the offline (h, k) BLOCKAWARECACHING problem. In particular, when $k = 2h$, we match the performance of classical CACHING algorithms up to constant factors.

[3]Beckmann et al. [BGM21] showed several deterministic lower bounds on the competitive factor achievable for (h, k) block-aware caching, when $k \geq h + \beta$-1.

5.2 Model and Preliminaries

5.2.1 Problem Definition

In the BLOCKAWARECACHING problem, there is a cache of size k and n pages which are partitioned into blocks. Let $\mathcal{B}$ be the partition of the pages into blocks. Each block contains at most β pages, for some $\beta \in [k]$. For a block $B \in \mathcal{B}$, we denote by $c_B > 0$ its cost. At each time-step t, a page p_t is requested. To serve the request, the page p_t must be fetched into the cache if it is missing from the cache. The goal is to obtain a feasible cache policy while minimizing the total cost. We consider two different cost functions.

Eviction cost model. In this model, fetching into the cache is free, while evictions have a cost that can be aggregated: Evicting any subset A of a block B at a time-step has a cost of c_B. The goal is to minimize the total eviction cost.

Fetching cost model. In this model, evicting pages from the cache is free, while fetching of pages has a cost that can be aggregated. Fetching of any subset A of a block B at a time-step has a cost of c_B. The goal is to minimize the total fetching cost.

Unlike classic paging and its variants, the fetching cost and eviction cost models are not equivalent in BLOCKAWARECACHING. We show that the optimal fetching and eviction costs for the same request sequence may be off by a factor of β (in either direction!), and this bound is tight.

Claim 5.2.1. *There exist instances of* BLOCKAWARECACHING *for which the optimal fetching cost is β larger than the optimal eviction cost, and there exist instances where the optimal eviction cost is β larger than the optimal fetching cost.*

See Section 5.6 for the proof.

For a page p, define $B(p)$ to be the block containing p, and let $r(p,t)$ be the time of the last request to p up until (and including) time t; if there is no such request, then $r(p,t) := -\infty$. Define the aspect ratio $\Delta := c_{\max}/c_{\min}$ where $c_{\max} := \max_{B \in \mathcal{B}} c(B)$ and $c_{\min} := \min_{B \in \mathcal{B}} c(B)$. For convenience we will use the notation $[\ell] = \{1, \ldots, \ell\}$ and $[\ell]_0 = \{0, 1, \ldots, \ell\}$.

5.3 Eviction Cost

In this section we show our algorithmic results for BLOCKAWARECACHING with respect to eviction costs. Our proof uses the (online) primal-dual method and hence requires an LP for the eviction cost model. It is straightforward to write a simple LP relaxation; unfortunately, the naïve relaxation has an integrality gap of $\Omega(\beta)$ (see Section 5.7), and recall that our goal is to beat the trivial algorithm's linear dependence on β.

5.3.1 SUBMODULARCOVER LP Formulation

To circumvent the naïve LP barrier, we strengthen the formulation using ideas from Wolsey's SUBMODULARCOVER LP. We start with some notation.

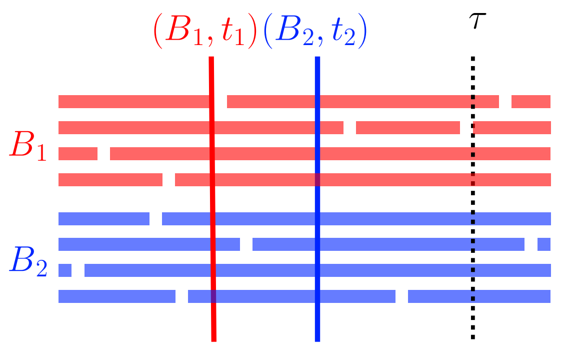


Figure 5.1: Illustration of the function f_τ. Each line represents a page, and each horizontal bar within the line represents an interval in which the page is not requested. Pages are grouped into their corresponding blocks. The solid vertical lines represent flushes. Suppose $n = 8$ and $k = 4$. Then $f_\tau(\{(B_1, t_1)\}) = 2$, $f_\tau(\{(B_2, t_2)\}) = 3$, but $f_\tau(\{(B_1, t_1), (B_2, t_2)\}) = 4$.

A *flush* is a tuple $(B, t) \in \mathcal{B} \times [T]_0$. The flush (B, t) corresponds to the event of evicting all cached pages of block B at time t. (There is no reason to only evict some of them, since they can be fetched back for free.) Let S be a set of flushes. We say that a page p is *missing* at time τ according to S if there exists $r(p, \tau) < t \leq \tau$ such that $(B(p), t) \in S$.[4] Crucially, this definition ensures that the page p_t requested at time t is not missing at time t. We say than an algorithm is *induced* by a set of flushes S if the algorithm evicts all pages of block B (except p_t) at time t if and only if $(B, t) \in S$, and always loads p_t at time t. Let n_t be the number of pages requested up until time t.

We use the above to define a set function $f_\tau : 2^{\mathcal{B} \times [T]_0} \to \mathbb{Z}$ on sets of flushes:

$$f_\tau(S) := \min(n - k,\ |\{p : p \text{ is } \textit{missing} \text{ at time } \tau \text{ according to } S\}|)$$

In words, $f_\tau(S)$ is the number of pages that are outside of the cache at time τ for the algorithm induced by S, where this number is capped at $n - k$. The algorithm induced by a set of flushes S is feasible at time τ iff $f_\tau(S) \geq n - k$ for all τ.

We show the following simple fact in Section 5.6:

Claim 5.3.1. *For every τ, the function f_τ is submodular.*

With the notation above, we can reformulate the BLOCKAWARECACHING problem with eviction cost as the solution to[5]

$$\boxed{\begin{aligned} &\min_{S \subseteq \mathcal{B} \times [T]_0} \sum_{\substack{(B,t) \in S \\ t \geq 1}} c_B \\ &\text{subject to} \\ &\forall \tau \in [T] : \quad f_\tau(S) \geq n - k. \end{aligned}} \tag{5.3.1}$$

[4]Adding to S all flushes of the form $(B, 0)$ ensures that also never-requested pages are missing by this definition.

[5]This formulation is reminiscent of the online and dynamic SUBMODULARCOVER problems of Chapter 3 and Chapter 6. However BLOCKAWARECACHING cannot be neatly reduced to Chapter 6, and the generic bounds from Chapter 3 are too coarse since they have a dependence on T.

Note that because $f_\tau(S)$ counts the number of pages evicted by S that are *not* p_τ, this single constraint captures both that the algorithm must flush at least $n-k$ pages in order to respect the cache size limit, and that the cache must contain page p_t at time t.

Note as well that we allow the algorithm to perform flushes at time 0, but only charge the cost for flushes performed after time 1. This conveniently allows the algorithm to clear the cache initially at no extra cost.

Finally, we are ready to write our LP, which is is the intersection of the SUBMODULARCOVER LPs of (2.4.1) for the functions f_τ, across all time steps $\tau \in [T]$.

Primal

$$\boxed{\begin{array}{rl} \min & \sum_{B,t\geq 1} c_B \cdot \phi_B^t \\ \text{subject to} & \\ \begin{array}{c}\forall S \subseteq \mathcal{B} \times [T]_0, \\ \forall \tau \in [T]\end{array} : & \begin{array}{l}\sum_{B,t} f_\tau((B,t) \mid S) \cdot \phi_B^t \\ \geq n-k-f_\tau(S)\end{array} \\ \forall B \in \mathcal{B}, t \in [T]_0 : & \phi_B^t \geq 0 \end{array}} \tag{P}$$

We will require the dual of this program, which is

Dual

$$\boxed{\begin{array}{rl} \max & \sum_{S,\tau} (n-k-f_\tau(S)) \cdot y_S^\tau \\ \text{subject to} & \\ \forall B \in \mathcal{B}, t \in [T] : & \sum_{S,\tau} f_\tau((B,t) \mid S) \cdot y_S^\tau \leq c_B \\ \begin{array}{c}\forall S \subseteq \mathcal{B} \times [T], \\ \forall \tau \in [T]\end{array} : & y_S^\tau \geq 0 \end{array}} \tag{D}$$

That (P) is a valid relaxation of (5.3.1) follows from Claim 2.4.1.

Claim 5.3.2 (Corollary of Claim 2.4.1)**.** *A set S has $f_\tau(S) = n-k$ for all τ if and only if χ_S, the characteristic vector of S, is a feasible integer solution to (P).*

We note for intuition's sake that even the constraints

$$\sum_{B,t} f_\tau((B,t)) \cdot \phi_B^t \geq n-k$$

alone already avoid the bad integrality gap example of Section 5.7. One reason is that truncating f_τ at $n-k$ prevents the LP from overestimating how much space will be saved by evictions.

Given an LP solution ϕ, we also define the fractional value of a page p missing from cache at time t to be

$$x_p^t := \begin{cases} 1 & \text{if } r(p,t) = -\infty. \\ \min\left\{1, \sum_{u=r(p,t)+1}^{t} \phi_{B(p)}^u\right\} & \text{otherwise.} \end{cases} \tag{5.3.2}$$

Intuitively, whenever some fraction δ of a flush (B, t) is chosen, we imagine increasing the fractional amount by which each page in B is evicted to extent δ. On the other hand when a page p_t is requested at time t, we reset its fractional value to $x_{p_t}^t = 0$.

5.3.2 A k-Competitive Deterministic Online Algorithm

Our first algorithmic result is a k-competitive deterministic online algorithm, which beats the trivial $k\beta$ competitive ratio obtained by running the deterministic online algorithm for classical paging. Our deterministic algorithm is given in Algorithm 7. The algorithm constructs simultaneously a primal solution x and a dual solution y to the LPs (P) and (D). We will ensure that these solutions satisfy all constraints known up to that time. We use $C(\tau)$ for the set of pages in cache at time τ, and we use S for the set of flushes performed by the algorithm so far. At the start of the algorithm, S is initialized as the set of all flushes of all blocks at time 0; this amounts to clearing the initial cache.The primal solution ϕ is set to the characteristic vector of S, and dual solutions y is initialized as the all-0-vector. At time τ, we first add the requested page p_τ to the cache. If this violates the cache constraint, we continuously increase the dual variable y_S^τ corresponding to the current set S and the current time τ until the dual constraint corresponding to some (B, t) with $f_\tau((B, t) \mid S) \geq 1$ becomes tight. Once this happens, we evict all pages of block B that are in cache (except p_τ, in case it belongs to this block) and update x and S to reflect that the flush (B, τ) has been performed.

Algorithm 7 Deterministic Online Algorithm

```
1: $S \leftarrow \{(B, 0) : B \in \mathcal{B}\}$
2: $\phi \leftarrow \chi_S,\ y \leftarrow \vec{0}$
3: for time $\tau = 1, \ldots, T$ do
4:     $C(\tau) \leftarrow C(\tau - 1) \cup \{p_\tau\}$
5:     if $|C(\tau)| > k$ then
6:         Increase $y_S^\tau$ until the dual constraint corresponding to some $(B, t)$ for which
               $f_\tau((B, t) \mid S) \geq 1$ is tight.
7:         $C(\tau) \leftarrow C(\tau) \setminus (B \setminus \{p_\tau\})$.
8:         $\phi_B^\tau \leftarrow 1$.
9:         $S \leftarrow S \cup \{(B, \tau)\}$.
```

We show that both primal and dual solutions are feasible, and that the cost of the primal is at most k times the cost of the dual. By weak duality, this implies:

Theorem 5.3.3. *Algorithm 7 is k-competitive.*

We begin by showing feasibility.

Lemma 5.3.4. *Algorithm 7 terminates. Upon termination, ϕ is feasible for* (P) *and y is feasible for* (D).

Proof. We first show that the algorithm terminates and the primal is feasible. Assume by induction that the algorithm maintains a feasible cache for every time step strictly less than τ (it is trivially feasible at time 0). Since exactly one page is requested per time step, if the cache is not feasible at the beginning of time step τ, then $|C(\tau)| = k + 1$. If this is the case, then there must exist some (B, t) such that $f_\tau((B, t) \mid S) \geq 1$, in which case the dual constraint corresponding to such a (B, t) will become tight after y^τ_S is increased sufficiently (in particular, the increase of y^τ_S terminates). Note that $t \leq \tau$ in this case. Then $f_\tau((B, \tau) \mid S) \geq f_\tau((B, t) \mid S) \geq 1$, so at least one page is evicted upon performing the flush (B, τ), and thus feasibility is restored at time τ. Hence the algorithm maintains a feasible cache state for every time τ, and by Claim 5.3.2 it follows that the primal is feasible for (P).

We now show feasibility of the dual. Clearly $y^\tau_S \geq 0$. Increasing y^τ_S could lead to a violation of the dual constraint corresponding to (B, t) only if $f_\tau((B, t) \mid S) \geq 1$, but in this case we stop once the constraint becomes tight. Thus, dual constraints are never violated. Note that dual variables y^τ_S corresponding to future time steps τ are 0, so it does not matter that the coefficients $f_\tau((B, t) \mid S)$ of future time steps are not known yet. ☐

Finally, we relate the primal and dual costs.

Lemma 5.3.5. *The cost of the primal is at most k times the cost of the dual.*

Proof. The algorithm sets $\phi^\tau_B = 1$ if and only if $(B, \tau) \in S$, so the primal cost is

$$P = \sum_{B,\tau} c_B \cdot \phi^\tau_B = \sum_{(B,\tau)\in S} c_B.$$

The algorithm adds (B, τ) to S only if a constraint corresponding to some $(B, t) = (B, t_\tau)$ with $f_\tau((B, t_\tau) \mid S) \geq 1$ becomes tight at time τ. Thus,

$$c_B = \sum_{S',u} f_u((B, t_\tau) \mid S') \cdot y^u_{S'}. \tag{5.3.3}$$

To account for our primal cost, for every flush $(B, \tau) \in S$, we charge $f_u((B, t_\tau) \mid S') \cdot y^u_{S'}$ of the cost of the flush to the dual variable $y^u_{S'}$. Since every non-zero dual variable $y^u_{S'}$ in our final solution has its coefficient in the objective $n - k - f_u(S') \geq 1$ (otherwise it would never have been increased), it suffices to argue that each dual variable $y^u_{S'}$ receives a total charge of at most $k \cdot y^u_{S'}$.

To see this, note that (B, τ) only charges its cost to variables $y^u_{S'}$ for which $u \in [t_\tau, \tau]$. Indeed, $f_u((B, t_\tau) \mid S') > 0$ only if $t_\tau \leq u$; moreover, when (5.3.3) becomes tight at time τ we have $y^u_{S'} = 0$ for $u > \tau$ and all S', and such $y^u_{S'}$ could subsequently increase only if $f_u((B, t_\tau) \mid S') = 0$ since otherwise the dual constraint would become violated. After (B, τ) is added to S, for all $t' \leq \tau$ and all $\tau' \geq \tau$ the multiplier $f_{\tau'}((B, t') \mid S) = 0$, and so $y^u_{S'}$ is charged at most once by every block B. Furthermore, if flush (B, τ) charges dual variable $y^u_{S'}$ then the coefficient $f_u((B, t) \mid S')$ is at most the number of pages from block B that were in cache at the end of time step $u - 1$. Since the total number of pages in cache at the end of time step $u - 1$ was at most k, the total amount charged to $y^u_{S'}$ is at most $k \cdot y^u_{S'}$. ☐

5.3.3 An $O(\log k)$-Competitive Monotone-Incremental Algorithm

In this section we give a competitive fractional algorithm for BLOCKAWARECACHING with eviction cost. For simplicity of presentation, our algorithm is not online in the strict sense, as at time τ we allow it to change the value of ϕ_B^t for $t < \tau$. However, it has the crucial property that it only increases LP variables ϕ_B^t. We call an algorithm with this property *monotone-incremental*. This property suffices for our rounding procedure in Section 5.3.4 to yield an online algorithm. We prove:

Theorem 5.3.6. *There is an $O(\log k)$-competitive monotone-incremental fractional algorithm for* BLOCKAWARECACHING *with eviction cost.*

To describe the algorithm, we define a flush (B, t) to be *alive* at time τ if $t = r(p, \tau) + 1$ for some $p \in B$. Intuitively, for an offline algorithm it is most beneficial to flush a block B only at time steps directly after some page from B was requested. Accordingly, our fractional algorithm will increase ϕ_B^t only for (B, t) that are alive. The algorithm is given in Algorithm 8. It starts by initializing S as the set of all flushes at time 0, ϕ as the corresponding characteristic vector, and y as the all-0-vector. At time τ, when some constraint (S', τ) is violated, we will show that this will also be the case for some $S' \supseteq S$, so that the condition of the while-loop will be true. We then increase the corresponding dual variable as well as primal variables corresponding to all alive flushes according to (5.3.4). While doing so, we occasionally add new flushes to the set S. As we will show later, all flushes (B, t) added to the set S will satisfy $\phi_B^t = 1$, i.e., they are chosen integrally by the fractional algorithm.

Algorithm 8 $O(\log k)$-competitive monotone-incremental fractional algorithm

```
1: $S \leftarrow \{(B, 0) : B \in \mathcal{B}\}$.
2: $\phi \leftarrow \chi_S,\ y \leftarrow \vec{0}$.
3: for time $\tau = 1, \ldots, T$ do
4:     while $\exists S' \supseteq S$ s.t. primal constraint of $(S', \tau)$ violated do
5:         Increase $y_{S'}^\tau$ continuously, and meanwhile for every alive $(B, t)$ increase $\phi_B^t$ at rate
               $$\frac{d\phi_B^t}{dy_{S'}^\tau} = \frac{\ln(k \cdot \beta + 1)}{c_B} \cdot f_\tau((B, t) \mid S') \cdot \left(\phi_B^t + \frac{1}{k \cdot \beta}\right) \qquad (5.3.4)$$
           until the dual constraint corresponding to some alive $(B_0, t_0)$ for which $f_\tau((B_0, t_0) \mid S') \geq 1$ is tight.
6:         $S \leftarrow S \cup \{(B_0, t_0)\}$.
```

Lemma 5.3.7. *Algorithm 8 terminates.*

Proof. If the condition of the while-loop is true, then $f_\tau(S') < n - k$, and thus there exists some alive (B_0, t_0) with $f_\tau((B_0, t_0) \mid S') \geq 1$. Increasing $y_{S'}^\tau$ sufficiently will eventually tighten a corresponding constraint, so each iteration of the while-loop terminates. When a new element is added to S at the end of an iteration, $f_\tau(S)$ increases by at least 1. When $f_\tau(S)$ has reached value $n - k$ (or earlier), the condition of the while-loop cannot be true any more. □

Lemma 5.3.8. *At the end of Algorithm 8, ϕ is feasible for (P) and y is feasible for (D).*

Before proving Lemma 5.3.8, we need the following claim, whose proof we leave for Section 5.6.

Definition 5.3.9. *Given a fractional solution ϕ, we say that the constraint (S, τ) is maximal-integral if for any (B, t) such that $\phi_B^t = 1$ it holds that $(B, t) \in S$.*

Claim 5.3.10. *If a fractional solution ϕ to* (P) *has no violated maximal-integral constraints, then ϕ is feasible.*

Proof of Lemma 5.3.8. We first show feasibility of the dual. Suppose the dual constraint corresponding to some (B, t) gets violated when $y_{S'}^{\tau}$ is increased. Let $t_0 \leq t$ be maximal such that (B, t_0) is alive at time τ. (If no such t_0 exists, then any pages evicted by the flush (B, t) are requested again in $[t, \tau]$; but then $f_\tau((B, t) \mid S') = 0$, so increasing $y_{S'}^{\tau}$ would not have led to a violation of the constraint corresponding to (B, t).) Since no pages of B are requested at times in $[t_0, t)$, we have $f_\tau((B, t_0) \mid S'') = f_\tau((B, t) \mid S'')$ for any S'', meaning that the dual constraint of (B, t_0) would become violated at the same time during the increase of $y_{S'}^{\tau}$. But then $f_\tau((B, t_0) \mid S') \geq 1$ (otherwise, increasing $y_{S'}^{\tau}$ would not increase the left-hand side of constraint (B, t_0)) and therefore we would have stopped increasing $y_{S'}^{\tau}$ when the constraint got tight.

To see that the primal is feasible, we will show that for all $(B, t) \in S$ we have $\phi_B^t = 1$. It then follows from Claim 5.3.10 that if a primal constraint (S', τ) is infeasible, then this is also the case for some $S' \supseteq S$; thus, the algorithm would not have terminated.

Consider the differential equation $dz/dy = \eta \cdot (z + \delta)$ for some constants $\eta \geq 0$ and $\delta > 0$. When y increases from a to b, we have

$$\ln(z(b) + \delta) - \ln(z(a) + \delta) = \eta \cdot (b - a). \tag{5.3.5}$$

For some (B, t) that eventually gets added to S, consider the dynamics of ϕ_B^t. It starts at 0, and increases with every $y_{S'}^{\tau}$ according to (5.3.4). Applying (5.3.5) for every such $y_{S'}^{\tau}$ and summing, we have

$$\ln\left(\phi_B^t + \frac{1}{k \cdot \beta}\right) - \ln\left(\frac{1}{k \cdot \beta}\right) = \sum_{S', \tau} \frac{\ln(k \cdot \beta + 1)}{c_B} \cdot f_\tau((B, t) \mid S') \cdot y_{S'}^{\tau}.$$

Taking exponents and solving, we have that

$$\phi_B^t = \frac{1}{k \cdot \beta} \cdot \left(\exp\left(\frac{\ln(k \cdot \beta + 1)}{c_B} \sum_{S', \tau} f_\tau((B, t) \mid S') \cdot y_{S'}^{\tau}\right) - 1\right).$$

In particular, when constraint (B, t) becomes tight and is added to S, the value of ϕ_B^t is 1. □

Lemma 5.3.11. *The cost of the primal is at most $O(\log k)$ times the cost of the dual.*

Proof. Consider the algorithm at a fixed time τ during a step in which $y_{S'}^{\tau}$ is increased by an infinitesimal amount $dy_{S'}^{\tau}$. The dual profit is $dy_{S'}^{\tau} \cdot (n - k - f_\tau(S'))$, and so it suffices bound the corresponding increase in primal cost. The primal cost increase is:

$$\sum_{B,t} c_B \cdot \frac{1}{c_B} \ln(k \cdot \beta + 1) \cdot f_\tau((B, t) \mid S') \cdot \left(\phi_B^t + \frac{1}{k \cdot \beta}\right) \cdot dy_{S'}^{\tau}$$
$$= \sum_{B,t} \ln(k \cdot \beta + 1) \cdot f_\tau((B, t) \mid S') \cdot \left(\phi_B^t + \frac{1}{k \cdot \beta}\right) \cdot dy_{S'}^{\tau}.$$

Since we only increase $y^{\tau}_{S'}$ if the corresponding constraint in the primal is not satisfied, we have

$$\sum_{B,t} f_{\tau}((B,t) \mid S') \cdot \phi^{t}_{B} < n - k - f_{\tau}(S'). \tag{5.3.6}$$

We claim that

$$\sum_{(B,t) \text{ alive}} \frac{f_{\tau}((B,t) \mid S')}{k \cdot \beta} \leq n - k - f_{\tau}(S'). \tag{5.3.7}$$

Inequalities (5.3.6) and (5.3.7) together imply that the increase in the primal cost is at most $2 \ln(k \cdot \beta + 1) \cdot (n - k - f_{\tau}(S)) \cdot dy^{\tau}_{S}$, which in turn implies the lemma statement since $\beta \leq k$.

To prove (5.3.7), we first show that

$$\sum_{B} \frac{f_{\tau}((B,\tau) \mid S')}{k} \leq n - k - f_{\tau}(S') \tag{5.3.8}$$

Note that $\sum_B f_{\tau}((B,\tau) \mid S') \leq n - f_{\tau}(S') - 1$. In the case that $n - f_{\tau}(S') \geq k+1$, (5.3.8) holds by the fact that $(z-1)/k \leq z - k$ for every $z \geq k+1$. Otherwise, when $n - f_{\tau}(S') < k+1$, then $f_{\tau}(S') = n - k$ (since f_{τ} is integer valued and truncated at $n-k$). Then (5.3.8) holds with both sides equal to 0.

We now obtain (5.3.7) via

$$\begin{aligned}
\sum_{(B,t) \text{ alive}} \frac{f_{\tau}((B,t) \mid S')}{k \cdot \beta} &\leq \sum_{(B,t) \text{ alive}} \frac{f_{\tau}((B,\tau) \mid S')}{k \cdot \beta} \\
&\leq \sum_{B} \frac{f_{\tau}((B,\tau) \mid S')}{k} \\
&\leq n - k - f_{\tau}(S')
\end{aligned}$$

where the second inequality uses that there are at most β flushes alive at any time, and the last inequality is (5.3.8). ☐

5.3.4 An $O(\log \Delta k)$-Competitive Online Randomized Rounding Scheme

Finally we show an online $O(\log \Delta k)$ randomized rounding scheme for our block caching LP (P). Recall the definition of the aspect ratio Δ from Section 5.2 and note that in the standard unweighted setting, $\Delta = 1$.

At time t, the algorithm evicts block B with probability $\gamma \cdot \phi^{\tau}_{B}$, where $\gamma = O(\log k\Delta)$. If the cache is still infeasible at time t, evict an arbitrary block so long as at least one of its pages has $x^{t}_{p} > 0$ (recall from the definition (5.3.2) that x^{t}_{p} is the amount missing from page p at time t). For clarity of exposition, the rounding procedure is written as if the underlying fractional solution (x, ϕ) is computed online. However the procedure can be carried out so long as the solution is monotone-incremental; at time τ, if the fractional solution increases any ϕ^{t}_{B} for $t < \tau$ by some amount δ_t, we can evict B at time τ with probability $\min(1, \gamma \cdot (\phi^{\tau}_{B} + \sum_{t<\tau} \delta_t))$.

Thus together with Theorem 5.3.6, our rounding scheme implies:

Theorem 5.3.12. *For* BLOCKAWARECACHING *with eviction cost, there exists an algorithm which is $O(\log k \log(k\Delta))$-competitive.*

Furthermore, using the round-or-separate procedure of Section 3.4, one can simultaneously solve and round the SUBMODULARCOVER LP (2.4.1) offline in polynomial time. Using the analysis of this section, this implies:

Theorem 5.3.13. *For* BLOCKAWARECACHING *with eviction cost, there exists an $O(\log(k\Delta))$-approximation algorithm.*

We perform the rounding assuming a few key properties of our fractional solution which we show we can assume (online) without changing our asymptotic guarantees.

Lemma 5.3.14. *Let (x, ϕ) be a fractional solution for LP (P). For an additional multiplicative constant factor to the competitive ratio, we can assume that (x, ϕ) has the following properties:*

- *For every time t, every page p has $x_p^t \in [0, 1/2] \cup \{1\}$.*
- *Every nonzero coordinate has $\phi_B^t \geq 1/4k^2$.*

We defer the proof to Section 5.6.

A key technical tool in this section is Lemma 3.2.5 from Section 3.2.

We can now present our rounding scheme.

Algorithm 9 $O(\log k)$-Approximate Rounding

```
1: for time $\tau \in [T]$ do
2:     For every block $B$ evict the set $\{p \in B \mid x_p^t > 0\}$ with probability $\min(1, \gamma \cdot \phi_B^t)$.
3:     Fetch $p_\tau$ if it is missing from the cache.
4:     while the cache is infeasible do
5:         Let $B$ be an arbitrary block in the cache that has a page $p$ with $x_p^t > 0$, evict the set of
               pages $\{p \in B \mid x_p^t > 0\}$.
```

Note that we assume that the fractional solution on which Algorithm 9 executes is one that has the properties given by Lemma 5.3.14.

We now prove our main rounding lemma.

Lemma 5.3.15. *For $\gamma = \log(4k^2\beta\Delta)$, given a feasible fractional solution (x, ϕ) with cost $c(\phi)$, Algorithm 9 produces a feasible integral cache policy of cost $O(\log k\Delta) \cdot c(\phi)$.*

To prove Lemma 5.3.15, we charge the cost of the algorithm to the fetching cost of the fractional solution. To relate this fractional fetching cost to the fractional eviction cost, we need a claim which we prove in Section 5.6.

Claim 5.3.16. *Let $c_{\text{FETCH}}(z)$ be the fetching cost of a fractional solution z. Then*

$$c_{\text{FETCH}}(z) \leq \beta \left(c(z) + \sum_{B \in \mathcal{B}} c_B \right).$$

Proof of Lemma 5.3.15. Let x, ϕ be a fractional solution given by Lemma 5.3.14, and let S be the set of flushes performed by our algorithm.

The algorithm produces a feasible cache policy by construction, as we always fetch p_t and we always run the eviction loop in Lines 4 and 5 until the cache is feasible. Note that there is always a block to evict with a page p that has $x_p > 0$, otherwise x is integral, and is the characteristic vector of the pages the algorithm has in cache, in which case the algorithm's cache is already feasible since the fractional solution is feasible. Furthermore, the expected cost of the evictions due to the randomized rounding step at Line 2 is at most $\gamma \cdot c(\phi) = O(\log k\Delta) \cdot c(\phi)$.

It remains to show that the total cost due to alterations in the eviction loop in Lines 4 and 5 is bounded. We now show that it is at most $O(c(\phi))$.

Let Λ be the set of times τ such that p_τ is not already fully in the fractional cache. Our algorithm maintains the invariant that if $x_p^t = 0$, then it is also fully in cache of the integral solution produced by our algorithm at time t. This means that at times $\tau \notin \Lambda$, neither the fractional solution nor the rounding algorithm incur a cost increase. Hence we focus on the case where $\tau \in \Lambda$.

For every τ, the solution ϕ is feasible for the LP (2.4.1) with the function f^τ, so by Lemma 3.2.5

$$\mathbb{E}[f_\tau(S)] \geq n - k - \frac{1}{4k^2\beta\Delta}.$$

In particular, this holds for all $\tau \in \Lambda$. In words, the expected number of pages in cache is bounded by $k + \frac{1}{4k^2\beta\Delta}$. Since every eviction due to Line 5 costs at most $c_{\max}$ and evicts at least one page, the expected cost of the alteration while loop at time τ is bounded by $c_{\max}/(4k^2\beta\Delta) = c_{\min}/4k^2\beta$.

On the other hand, since $\tau \in \Lambda$, the page p_τ is not fully in cache, and since by Lemma 5.3.14 the fractional solution evicts pages in increments of at least $1/(4k^2)$, it holds that $x_{p_\tau}^\tau \geq 1/(4k^2)$. This means that the *fetching* cost of the fractional solution at time τ is at least $c_{\min}/4k^2$.

Hence the expected cost of the alteration step in time τ is at most the fractional fetching cost at time τ, divided by β. Summing this inequality over time, the total cost paid by the algorithm over all all time due to Line 5 is at most $c_{\text{FETCH}}(\phi)/\beta$. By Claim 5.3.16, the fetching cost $c_{\text{FETCH}}(\phi) \leq \beta(c(\phi) + \sum_{B\in\mathcal{B}} c_B)$, and hence the total cost of alterations is at most $c(\phi) + \sum_{B\in\mathcal{B}} c_B$. This completes the proof. $\square$

5.4 Fetching Cost

We present our $\Omega(\beta)$ lower bound against randomized algorithms for online BLOCKAWARECACHING with fetching costs. We first present a bicriteria rounding algorithm for the naïve LP of Section 5.7. We then argue that this procedure can be used to derandomize any randomized algorithm for BLOCKAWARECACHING with fetching costs. Together with the lower bound against deterministic algorithms given by [BGM21], this implies a lower bound against randomized algorithms.

5.4.1 Bicriteria Online Rounding Algorithm

Consider the following deterministic online rounding scheme. For every page p, evict p from the cache at time t if $x_p^t > 1/2$. If a page p_t is not in cache upon request at time t, then at time t fetch all pages from $B(p_t)$ such that $x_p^t \leq 1/2$.

Theorem 5.4.1. *Given a feasible fractional solution x to the* BLOCKAWARECACHING *problem, the procedure above produces an integral solution that uses at most $2k$ cache space at any point in time, and whose fetching cost is at most twice the fetching cost of x.*

Proof. The procedure produces a feasible solution by construction, since $x^t_{p_t} = 0 \leq 1/2$. It also violates the cache size constraint by at most a factor of 2, since no page is present in the integral cache unless $x^t_p \leq 1/2$, meaning the fractional cache usage is at least half the integral cache usage.

Finally, to justify that the integral solution has cost at most twice the fractional cost, charge the cost of integrally loading $B(p_t)$ to the fractional decrease of x^t_B since the last time t' at which $B(p_t)$ was loaded. Since p_t had $x^{t'}_p > 1/2$ (otherwise we would have loaded it earlier), the fractional cost incurred since time t' was at least $1/2 \cdot c_{B(p)}$. □

Corollary 5.4.2. *When $k = 2h$, there is a 2-competitive offline algorithm for* BLOCKAWARECACHING *with fetching cost.*

We mention briefly that a similar rounding procedure produces a cache policy that is 2-competitive with the eviction cost of the fractional solution, and also uses at most a factor 2 more space. If p_t is not in cache at time t, fetch it. On the other hand if any page in cache at time t has fractional value $x^t_p < 1/2$, evict all of $B(p)$.

5.4.2 Lower Bounds for Randomized Algorithms

Finally we turn to showing our lower bound. Our starting point is the lower bound of [BGM21] against deterministic algorithms.

Theorem 5.4.3 (Theorem 4.1 of [BGM21])**.** *The competitive ratio of any deterministic online policy for* BLOCKAWARECACHING *with fetching costs is at least*

$$\frac{k + (B-1)(h-1)}{k - h + 1}$$

for $h \leq k - B + 1$.

We now show how to use the online deterministic rounding procedure of Section 5.4.1 to derandomize any online algorithm for BLOCKAWARECACHING with fetching costs. This proves the main claim of this section:

Theorem 5.4.4. *The competitive ratio of any randomized policy for* BLOCKAWARECACHING *with fetching costs is at least*

$$\frac{2k + (B-1)(h-1)}{4k - 2h + 2}$$

for $h \leq k - B + 1$.

Proof. Suppose there is a randomized online algorithm $\mathcal{R}$ for (h, k)-BLOCKAWARECACHING with fetching costs with expected cost $c_{\mathcal{R}}$. Then we can convert this randomized cache policy online to a fractional solution x. To do so, set x^t_p be the expected value of the indicator of whether page p is loaded at time t. Note that these expectations can be computed using only the sequence of requests up to and including time t. This solution x is feasible to the simple fetching cost LP (5.7.1), and furthermore has LP cost $c_{\mathcal{R}}$.

Applying Theorem 5.4.1 to the fractional solution x produces an integral cache policy cost at most $2 \cdot c_{\mathcal{R}}$ and space $2k$. The claim follows by using the lower bound on the cost of any such policy given by Theorem 5.4.3, and solving for $c_{\mathcal{R}}$. □

Combining this with the well known $\Omega(\log k)$ lower bound for randomized algorithms for classical paging, we obtain the following consequence.

Corollary 5.4.5. *When $k = O(h)$, no randomized algorithm has competitive ratio better than $\Omega(B + \log k)$.*

5.5 Conclusion

In this chapter we significantly expand on known results of [BGM21] for the block-aware caching problem. For the eviction cost model, we give the first set of algorithms avoiding a trivial multiplicative β overhead over their classical paging counterparts. We also give (h, k)-bicriteria algorithms for both fetching and eviction cost models, which we in turn use to adapt the lower bound of [BGM21] for the fetching cost model to randomized algorithms.

We leave the following open questions. Can our $O(\log^2 k)$ competitive ratio for block-caching in the eviction cost model be improved to $O(\log k)$ to match the lower bound from classical paging? Note that even for the special case of generalized caching, the first result of [BBN12b] achieved competitive ratio $O(\log^2 k)$. The subsequent improvement due to [ACER19] required non-trivial ideas which seem difficult to adapt to the more general block-aware caching problem. For the fetching cost model, is there an algorithm matching the lower bound of $\beta + \log k$, or can the lower bound be strengthened to match the trivial $\beta \log k$ upper bound? Finally, are there constant-approximation *offline* algorithms for block-caching in either the fetching or eviction cost models?

This chapter is demonstrates the versatility of the ideas from Chapter 3; more generally it illustrates the power of the abstraction of submodularity in the context of online algorithms.

5.6 Deferred Proofs

Claim 5.2.1. *There exist instances of* BLOCKAWARECACHING *for which the optimal fetching cost is β larger than the optimal eviction cost, and there exist instances where the optimal eviction cost is β larger than the optimal fetching cost.*

Proof. Consider the following instance. For any β, let $n = 2\beta^2$ pages be organized into 2β blocks of size β. Let P be the first β blocks and Q be the second β blocks. We set $k = \beta^2$, and fill the cache initially with all the pages of the P blocks. The request sequence consists of rounds. For $i = 1, \ldots, \beta$, in round i request the first $\beta - i$ pages of each P block, and the first i of the Q blocks in their entirety, and repeat this sequence L times within the round. For sufficiently large constant L, the optimal solution must have precisely the requested pages of a round in its cache. Thus, in round i it evicts the $(\beta - i + 1)^{th}$ page of each P block and fetches the i^{th} Q block in its entirety. The fetching cost of this solution is β, while the eviction cost is β^2.

To see the other direction, observe that if we instead start the cache with the pages of the Q blocks, and for $i = 1, \ldots, \beta$ we make round i request precisely the pages *not* requested in round i above (and once again repeat this sequence L times), the optimal solution will always evict one Q block in its entirety and fetch a single page from each P block in each round. Here the fetching cost is β^2 and the eviction cost is β. □

Claim 5.3.1. *For every τ, the function f_τ is submodular.*

Proof. Consider the function g^τ, where

$$\begin{aligned} g^\tau(S) &:= |\{p : p \text{ is } \textit{missing} \text{ at time } \tau \text{ according to } S\}| \\ &= \left| \bigcup_{\phi \in S} \{p : p \text{ is } \textit{missing} \text{ at time } \tau \text{ according to } \phi\} \right| . \end{aligned}$$

g^τ is a coverage function, and hence it is submodular. The function f^τ is the minimum of g^τ and the constant function $n - k$, and so f^τ is also submodular. □

Claim 5.3.10. *If a fractional solution ϕ to* (P) *has no violated maximal-integral constraints, then ϕ is feasible.*

Proof. It suffices to show that if ϕ violates a constraint (S, τ), and (B_0, t_0) is such that $\phi_{B_0}^{t_0} = 1$, then x also violates $(S \cup \{(B_0, t_0)\}, \tau)$. Since x is violated,

$$\begin{aligned} n - k - f_\tau(S) &> \sum_{B,t} f_\tau((B,t) \mid S) \cdot \phi_B^t \\ &= \sum_{(B,t) \neq (B_0,t_0)} f_\tau((B,t) \mid S) \cdot \phi_B^t + f_\tau((B_0,t_0) \mid S) \\ &\geq \sum_{(B,t) \neq (B_0,t_0)} f_\tau((B,t) \mid S \cup \{(B_0,t_0)\}) \cdot \phi_B^t \\ &\quad + f_\tau((B_0,t_0) \mid S) \end{aligned}$$

where the second inequality above used submodularity. Rearranging gives that

$$\sum_{(B,t) \neq (B_0,t_0)} f_\tau((B,t) \mid S \cup \{(B_0,t_0)\}) \cdot \phi_B^t < n - k - f_\tau(S \cup \{(B_0,t_0)\}). \qquad \square$$

Lemma 5.3.14. *Let (x, ϕ) be a fractional solution for LP* (P)*. For an additional multiplicative constant factor to the competitive ratio, we can assume that (x, ϕ) has the following properties:*

- *For every time t, every page p has $x_p^t \in [0, 1/2] \cup \{1\}$.*
- *Every nonzero coordinate has $\phi_B^t \geq 1/4k^2$.*

Proof. To guarantee the first property, every time a page from a block B is evicted to extent $1/2$, evict the entire block B for a cost of c_B. Charge this eviction to the evictions that caused this page to go from 0 to $1/2$, which cost at least $c_B/2$.

To ensure the second property, consider the following online algorithm.

Algorithm 10 Structure Solution

```
1: Define solution $\widetilde{x}$ such that $\widetilde{\phi}_B^t = \phi_B^t + \mathbb{1}\{\phi_B^t \geq 1/2\} \cdot (1 - \phi_B^t)$.
2: for block $B$ do
3:     Set $t_B \leftarrow 0$.
4:     for time $t \in [T]$ do
5:         Let $\Delta = \sum_{t'=t_B+1}^{t} \widetilde{\phi}_B^{t'}$.
6:         if $\Delta \geq 1/4k^2$ then
7:             $\varphi_B^t \leftarrow \Delta$.
8:             $t_B \leftarrow t$.
9: Output $\widetilde{\varphi} = \min(2 \cdot \varphi, 1)$.
```

The structural guarantee that every nonzero coordinate has $\varphi_B^t \geq 1/4k^2$ holds by construction. The cost is also less than $2 \cdot c(\phi)$ by construction.

It remains to show φ is feasible. Consider any constraint of the form:

$$\sum_{B,t} f_\tau((B,t) \mid S) \cdot \phi_B^t \geq n - k - f_\tau(S)$$

For a block B, let τ_B be the last time before τ that t_B was set to in Line 8. By construction

$$\sum_{t=\tau_B+1}^{\tau} \phi_B^t \leq \frac{1}{4k^2}.$$

Since $f((B,t)|S) \leq k$, and by the property that every p has $x_p^t \in [0, 1/2] \cup \{1\}$, there are at most $2k$ blocks with nonzero pages in cache. This also means

$$\sum_B \sum_{t=\tau_B+1}^{\tau} f((B,t) \mid S) \cdot \phi_B^t \leq \frac{1}{2}$$

and hence

$$\sum_{B,t} f_\tau((B,t) \mid S) \cdot \varphi_B^t \geq n - k - f_\tau(S) - \frac{1}{2}.$$

If $n - k - f_\tau(S) = 0$, then the constraint is also trivially satisfied by φ. Else $n - k - f_\tau(S) \geq 1$. To conclude, note that since S is maximal-integral, and ϕ has no coordinates $\phi_B^t \in (1/2, 1)$:

$$\begin{aligned}\sum_{t \leq \tau_0} \sum_B f((B,t) \mid S) \cdot \varphi_B^t &= 2 \sum_{t \leq \tau_0} \sum_B f((B,t) \mid S) \cdot \phi_B^t \\ &\geq 2\left(n - k - f_\tau(S) - \frac{1}{2}\right) \\ &\geq n - k - f_\tau(S)\end{aligned}$$

If φ satisfies all integral-maximal constraints, Claim 5.3.10 implies it also satisfies all other constraints, and the lemma statement follows. □

Claim 5.3.16. *Let $c_{\text{FETCH}}(z)$ be the fetching cost of a fractional solution z. Then*

$$c_{\text{FETCH}}(z) \leq \beta \left(c(z) + \sum_{B \in \mathcal{B}} c_B \right).$$

Proof. Let $\overline{c}(z)$ and $\overline{c}_{\text{FETCH}}(z)$ be the classic paging eviction/fetching cost of z, i.e. the cost if page fetches/evictions cannot be batched in blocks. The difference between the total fetching cost and eviction cost paid for a single page is at most $c(B(p))$, and hence $\overline{c}(z) = \overline{c}_{\text{FETCH}}(z) \pm \sum_{B \in \mathcal{B}} c_B \cdot \beta$. On the other hand, $\overline{c}(z) \leq \beta \cdot c(z)$. Combining these observations:

$$c_{\text{FETCH}}(z) \leq \overline{c}_{\text{FETCH}}(z) \leq \overline{c}(z) + \beta \cdot \sum_{B \in \mathcal{B}} c_B \leq \beta \cdot \left(c(z) + \sum_{B \in \mathcal{B}} c_B \right). \qquad \square$$

5.7 The Natural LP has $\Omega(\beta)$ Integrality Gap

Consider the following simple LP formulation, where $\sigma \in \{-1, 1\}$ is a fixed constant. We use x_p^t for the fraction of page p missing from the cache at time t.

$$\begin{array}{|rl|}
\hline
\min\limits_{\phi, x} & \sum\limits_{B,t} c_B \cdot \phi_B^t \\
\text{subject to} & \\
\forall t \in [T]: & x_{p(t)}^t = 0 \\
\forall t \in [T], \forall B \in \mathcal{B}, \forall p \in B: & \phi_B^t \geq \sigma \left(x_p^t - x_p^{t-1} \right) \\
\forall t \in [T]: & \sum_p x_p^t \geq n - k \\
\forall t \in [T], \forall B \in \mathcal{B}: & \phi_B^t \in [0,1] \\
\forall t \in [T], \forall p \in [n]: & x_p^t \in [0,1] \\
\hline
\end{array} \tag{5.7.1}$$

If $\sigma = 1$, this is the eviction cost model and ϕ_B^t denote the fractional extent to which B is evicted at time t; if $\sigma = -1$, this is the fetching cost model and ϕ_B^t denote the fractional extent to which B is fetched at time t.

Unfortunately, for both fetching and eviction cost models, this LP has an integrality gap of $\Omega(\beta)$. Consider the following instance in which $n = 2\beta$ pages are divided into two blocks B_1 and B_2. The cache is of size $k = 2\beta - 1$, and is initially empty. The request sequence repeats for several rounds. In each round, it requests first all pages from B_1 and then all pages from B_2.

The integral algorithm must pay at least 1 per round, since 2β pages are requested and the cache is of size $2\beta - 1$. On the other hand, the fractional solution begins by loading both blocks to extent $(\beta - 1)/\beta$. Subsequently, when B_i is requested for $i \in \{1, 2\}$, it loads B_i to extent 1 and B_j (where $j \neq i$) to extent $(\beta - 1)/\beta$. Hence both the fractional fetching and eviction costs per round are $2/\beta$. We summarize this observation in the following theorem.

Theorem 5.7.1. *The simple LP relaxation for* BLOCKAWARECACHING *in both fetching/eviction cost models has an integrality gap of* $\Omega(\beta)$.

Part II

Dynamic Algorithms

Chapter 6

Fully-Dynamic Submodular Cover

6.1 Introduction

We consider the SUBMODULARCOVER problem in a fully-dynamic setting, where the notion of coverage changes over time. At each time, the underlying submodular function changes from $f^{(t)}$ to $f^{(t+1)}$, and the algorithm may have to change its solution from S_t to S_{t+1} to cover this new function. We do not want our solutions to change wildly if the function changes by small amounts. The goal of this work is to develop algorithms where this "churn" $|S_t \triangle S_{t+1}|$ is small (perhaps in an amortized sense), while maintaining the requirement that each solution S_t is a good approximate solution to the function $f^{(t)}$. The change $|S_t \triangle S_{t+1}|$ is often called *recourse* in the literature. We call this problem DYNAMICSUBMODULARCOVER.

This problem has been posed and answered in the special case of SETCOVER—for clarity, from now on this chapter we consider the equivalent HYPERGRAPHVERTEXCOVER (a.k.a. the HITTINGSET) problem. In this problem, hyperedges arrive to and depart from an active set over time, and we must maintain a small set of vertices that hit all active hyperedges. We know an algorithm that maintains an $O(\log n_t)$-approximation which has constant amortized recourse [GKKP17]; here n_t refers to the number of active hyperedges at time t. In other words, the total recourse—the total number of changes over T edge arrivals and departures—is only $O(T)$. The algorithm and analysis are based on a delicate token-based argument, which gives each hyperedge a constant number of tokens upon its arrival, and moves these tokens in a careful way between edges to account for the changes in the solution. What do we do in the more general SUBMODULARCOVER case, where there is no notion of sets any more?

We focus on the bag-of-functions model (see Section 2.6): at time t a submodular function g^t is added or removed from the *active set* $G^{(t)}$ at each timestep, and this defines the current submodular function $f^{(t)} := \sum_{g \in G_t} g$. The algorithm must maintain a subset $S_t \subseteq \mathcal{N}$ such that $f^{(t)}(S_t) = f^{(t)}(\mathcal{N})$ with cost $c(S_t)$ being within a small approximation factor of the optimal SUBMODULARCOVER for $f^{(t)}$, such that the total recourse $\sum_t |S_t \triangle S_{t+1}|$ remains small[1]. Recall that to model dynamic HYPERGRAPHVERTEXCOVER, each arriving/departing edge A_t should correspond to a submodular function g_t taking on value 1 for any set S that hits A_t, and value zero

[1]We emphasize the difference between this problem and the one defined in Chapter 3: in that setting, functions may only be *inserted* into the active set, and no recourse is allowed.

otherwise (i.e., $g_t(S) = \mathbb{1}[S \cap A_t \neq \emptyset]$).

6.1.1 Our Results

Our main result is the following:

Theorem 6.1.1 (Informal)**.** *There is an $e^2 \cdot (1 + \log f_{\max})$-competitive deterministic algorithm for* DYNAMICSUBMODULARCOVER *with total recourse:*

$$O\bigg(\sum_t g_t(\mathcal{N}) \log\Big(\frac{c_{\max}}{c_{\min}}\Big)\bigg),$$

where $g_t(\mathcal{N})$ is the largest value of the function considered at time t, and $c_{\max}, c_{\min}$ are the maximum and minimum element costs.

Let us parse this result. Firstly, the approximation factor almost matches Wolsey's result up to the multiplicative factor of e^2; this is best possible in polynomial time unless $\mathsf{P} = \mathsf{NP}$ even in the offline setting. Secondly, the amortized recourse bound should be thought of as being logarithmic—indeed, specializing it to HITTINGSET where each $g_t(\mathcal{N}) = 1$, we get a total recourse of $O(T \cdot \log(c_{\max}/c_{\min}))$ over T timesteps, or an amortized recourse of $O(\log(c_{\max}/c_{\min}))$ per timestep. Hence this recourse bound is weaker by a log-of-cost-spread factor, while generalizing to all monotone submodular functions. Finally, since we are allowed to give richer functions g_t at each step, it is natural that the recourse scales with $\sum_t g_t(\mathcal{N})$, which is the total "volume" of these functions. In particular, this problem captures fully-dynamic HYPERGRAPHVERTEXCOVER where at each round a batch of k edges appears all at once (this is in contrast to the standard fully-dynamic model where hyperedges appear one at a time). In this case the algorithm may have to buy up to $k = g_t(\mathcal{N})/f_{\min}$ new vertices in general to maintain coverage.

Our results can be seen as an extension/improvement of the main result in Chapter 3 when recourse is allowed: not only can our algorithm handle the fully-dynamic case with insertions and deletions, but we improve the competitive ratio from $O(\log n \cdot \log f_{\max}/f_{\min})$ to $O(\log f_{\max}/f_{\min})$, which is best possible even in the offline case.

We next show that for coverage functions (and hence for the HYPERGRAPHVERTEXCOVER problem), a variation on the algorithm from Theorem 6.1.1 can remove the log-of-cost-spread factor in terms of recourse, at the cost of a slightly coarser competitive ratio. E.g., for HYPERGRAPHVERTEXCOVER the new competitive ratio corresponds to an $O(\log n)$ guarantee versus an $O(\log D_{\max})$ guarantee, where $D_{\max}$ being the largest degree of any vertex.

Theorem 6.1.2 (Informal)**.** *There is an $O(\log f(\mathcal{N}))$-competitive deterministic algorithm for* DYNAMICSUBMODULARCOVER*, in the setting where the g^t function are 3-increasing in addition to monotone and submodular, with total recourse:*

$$O\bigg(\sum_t g_t(\mathcal{N})\bigg).$$

Indeed, this result holds not just for coverage functions, but for the broader class of 3-*increasing, monotone, submodular functions* [FH05], which are the functions we have been considering, with the additional property that have positive discrete mixed third-derivatives. At a high level, these are functions where the mutual coverage does not increase upon conditioning.

6.1.2 Techniques and Overview

The most widely known algorithm for SUBMODULARCOVER is the greedy algorithm; this repeatedly adds to the solution an element maximizing the ratio of its marginal coverage to its cost. It is natural to try to use the greedy algorithm in our dynamic setting; the main issue is that out-of-the-box, greedy may completely change its solution between time steps. In their result on recourse-bounded HYPERGRAPHVERTEXCOVER, [GKKP17] showed how to effectively imitate the greedy algorithm without sacrificing more than a small amount of recourse. A barrier to making greedy algorithms dynamic is their sequential nature, and hyperedge inserts/deletes can play havoc with this. So they give a local-search algorithm that skirts this sequential nature, while simultaneously retaining the key properties of the greedy algorithm. Unfortunately, their analysis hinges on delicately assigning edges to vertices. In the more general SUBMODULARCOVER setting, there are no edges to track, and this approach breaks down.

Our first insight is to return to the sequential nature of the greedy algorithm. Our algorithm maintains an ordering π on the elements of $\mathcal{N}$, which induces a solution to the SUBMODULARCOVER problem: we define the output of the algorithm as the prefix of elements that have non-zero marginal value. We maintain this permutation dynamically via local updates, and argue that only a small amount of recourse is necessary to ensure the solution is competitive. To bound the competitive ratio, we imagine that the permutation corresponds to the order in which an auxiliary offline algorithm selects elements, i.e. a *stack trace*. We show that our local updates maintain the invariant that this is the stack trace of an approximate greedy algorithm for the currently active set of functions. We hope that this general framework of doing local search *on the stack trace of an auxiliary algorithm* finds uses in other online/dynamic algorithms.

Our main technical contribution is to give a potential function to argue that our algorithm needs bounded recourse. The potential measures the *(generalized) entropy* of the *coverage vector* of the permutation π. This coverage vector is indexed by elements of the universe $\mathcal{N}$, and the value of each coordinate is the marginal coverage (according to permutation π) of the corresponding element. Entropy is often used as a potential function (notably, in recent developments in online algorithms) but in a qualitatively different way. In many if not all cases, these algorithms follow the maximum-entropy principle and seek *high entropy* solutions subject to feasibility constraints; the potential function then tracks the convergence to this goal. On the other hand, in our setting the cost function is the support size of the coverage vector, and minimizing this roughly corresponds to *low entropy*. We show entropy decreases sufficiently quickly with every change performed during the local search, and increases by a controlled amount with insertion/deletion of each function g, thus proving our recourse bounds.

We find our choice of entropy to be interesting for several reasons. For one, we use a suitably chosen *Tsallis entropy* [Tsa88], which is a general parametrized family of entropies, instead of the classical Shannon entropy. The latter also yields recourse bounds for our problem, but they are substantially weaker (see Section 6.9). Tsallis entropy has appeared in several recent algorithmic areas, for example as a regularizer in bandit settings [SS14], and as an approximation to Shannon entropy in streaming contexts [HNO08]. Secondly, it is well known that for certain problems, minimizing an ℓ_1-objective is an effective proxy for minimizing sparsity [CRT06]. In our dynamics, the ℓ_1 mass stays constant since total coverage does not change when elements are reordered. However, entropy is a good proxy for sparsity, in the sense that it decreases monotonically (and quickly!) as our algorithm moves within the ℓ_1 level set towards sparse vectors.

In Section 6.2, we study the unit cost case to lay out our framework and highlight the main ideas. We show a $\log f_{\max}/f_{\min}$ competitive algorithm for fully-dynamic SUBMODULARCOVER with $O(\sum_t g_t(\mathcal{N})/f_{\min})$ total recourse. Then in Section 6.3, we turn to general cost functions. We again show a $\log(f_{\max}/f_{\min})$ competitive algorithm, this time with $\log(c_{\max}/c_{\min})\cdot\sum_t g_t(\mathcal{N})/f_{\min}$ recourse. The algorithm template is the same as before, but with a suitably generalized potential function and analysis. Here we also require a careful choice of the Tsallis entropy parameter, near (but not quite at) the limit point at which Tsallis entropy becomes Shannon entropy.

In Section 6.4, we show how to remove the $\log(c_{\max}/c_{\min})$ dependence and achieve optimal recourse for a structured family of monotone, submodular functions: the class of 3-*increasing (monotone, submodular) set functions* [FH05]. These are set functions, all of whose discrete mixed third derivatives are nonnegative everywhere. Submodular functions in this class include *measure coverage functions*, which generalize set coverage functions, as well as entropies for distributions with positive interaction information (see, e.g., [Jak05] for a discussion). Since this class includes SETCOVER, this recovers the optimal $O(1)$ recourse bound of [GKKP17]. For this result we use a more interesting generalization of the potential function from Section 6.2 that reweights the coverage of elements in the permutation non-uniformly as a function of their mutual coverage with other elements of the permutation.

In Section 6.5, we show how to get improved randomized algorithms when the functions g_t are assumed to be r-juntas. This is in analogy to approximation algorithms for SETCOVER under the frequency parametrization. In Section 6.6, we also show how to run an online "combiner" algorithm that gets the best of all worlds, with a competitive ratio of $O(\min(r, \log f_{\max}/f_{\min}))$. In Section 6.7, we demonstrate the generality of our framework by using it to recover known recourse bounded algorithms for fully-dynamic MINIMUMSPANNINGTREE and MINIMUMSTEINERTREE. These achieve $O(1)$ competitive ratios, and recourse bounds of $O(\log D)$ where D is the aspect ratio of the metric. Our proofs here are particularly simple and concise.

6.1.3 Related Work

The definition of m-*increasing functions* is due to Foldes and Hammer [FH05]. Bach [Bac13] gave a characterization of the class of *measure coverage functions* (which we define later) in terms of its iterated discrete derivatives. This class generalizes coverage functions, and is contained in the class of 3-increasing functions. [IKBA21] give several additional examples of 3-increasing functions. Several papers [IKBA21, CM18, WMWP15, WWLP13] have given algorithms specifically for 3-increasing submodular function optimization.

Online and Dynamic Algorithms. There is a still budding series of work on recourse-bounded algorithms. Besides [GKKP17] which is most directly related to our work, researchers have studied the Steiner Tree problem [IW91, GK14, GGK16, LOP$^+$15], clustering [GKLX20, CHP$^+$19], matching [GKKV95, CDKL09, BLSZ14], and scheduling [PW93, Wes00, AGZ99, SSS09, SV10, EL14, GKS14].

A rich parallel line of work has studied how to minimize update time for problems in the dynamic or fully-dynamic setting. In [GKKP17], the authors give an $O(\log n)$ competitive and $O(F \log n)$ update time algorithm for fully-dynamic HYPERGRAPHVERTEXCOVER. An ongoing program of research for the frequency parametrization of HYPERGRAPHVERTEXCOVER

[BHI18, BCH17, BK19, AAG+19, BHN19, BHNW21] has so far culminated in an $F(1+\epsilon)$ competitive algorithm with $\mathrm{poly}(F, \log c_{\max}/c_{\min}, 1/\epsilon)$ update time (where F is the frequency).

Our work is related to work on convex body chasing (e.g., [AGTG21, Sel20]) in spirit but not in techniques. For an online/dynamic covering problems, the set of feasible fractional solutions within distance α of the optimal solution at a given time step form a convex set: our goal is similarly to "chase" these convex bodies, while limiting the total movement traversed. The main difference is that we seek *absolute bounds* on the recourse, instead of recourse that is competitive with the optimal chasing solution. (We can give such bounds because our feasible regions are structured and not arbitrary convex bodies).

6.2 Unit Cost SubmodularCover

6.2.1 The Algorithm

We now present our first algorithm for unit-cost fully-dynamic SubmodularCover. We will show the following:

Theorem 6.2.1. *For any $\gamma > e$, there a is a $\gamma \cdot (\log f_{\max}^{(t)}/f_{\min}^{(t)} + 1)$-competitive deterministic algorithm for unit-cost* DynamicSubmodularCover *with total recourse:*

$$2 \cdot \frac{e \log \gamma}{\gamma - e \log \gamma} \cdot \frac{\sum_t g_t(\mathcal{N})}{f_{\min}} = O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}}\right).$$

The algorithm and its analysis are particularly clean; we will build on these in the following sections. We begin by describing the algorithm. We maintain a permutation π of the elements in $\mathcal{N}$, and assign to each element its marginal coverage given what precedes it in the permutation. We write this marginal value assigned to element π_i as

$$\mathfrak{F}_\pi(\pi_i) := f(\pi_i \mid \pi_{1:i-1}). \tag{6.2.1}$$

We consider two kinds of local search moves:

1. **Swaps:** transform π to π' by moving an element at position i to position $i-1$ on the condition that $\mathfrak{F}(\pi_i) \geq \mathfrak{F}(\pi_{i-1})$. In words, this is a *bubble* operation (as in bubble-sort).
2. γ**-moves**: transform the permutation π to π' by moving an element u from a position q to some other position $p < q$ on the condition that for all $i \in \{p, \dots, q-1\}$,

$$\mathfrak{F}_{\pi'}(\pi'_p) \geq \gamma \cdot \mathfrak{F}_\pi(\pi_i).$$

   In words, when u moves ahead in line, it "steals" coverage from other elements along the way; we require that the amount covered by u *after the move* (which is given by $\mathfrak{F}_{\pi'}(\pi'_p)$ since u now sits at position p) to be at least a γ factor larger than the coverage *before the move* of any element that u jumps over. (See Figure 6.1.)

The dynamic algorithm is the following. When a new function $g^{(t)}$ arrives or departs, update the coverages $\mathfrak{F}_\pi$ of all the elements in the permutation. Subsequently, while there is a local search move available, perform the move. Output the prefix of π of all elements with non-zero coverage. This is summarized in Algorithm 11, with a setting of $\gamma > e$.

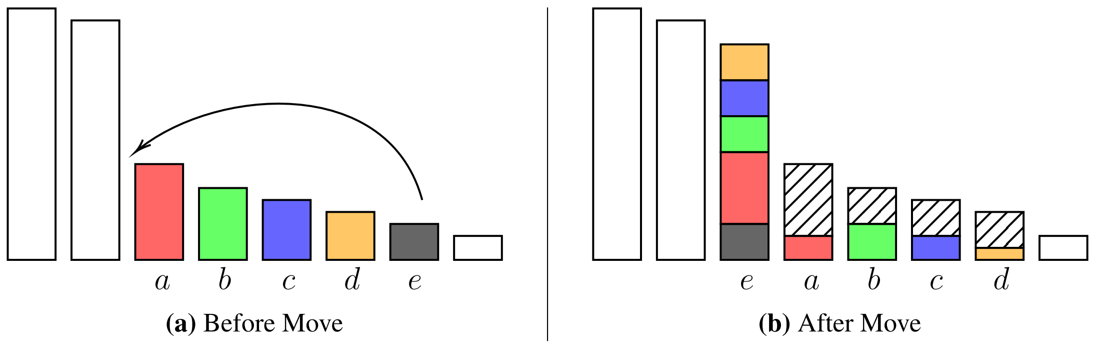


Figure 6.1: Illustration of a legal γ-move. Each rectangle represents the marginal coverage of an element of the permutation. The height of the item that moves must be at least γ times the height of anyone it cuts in line.

Algorithm 11 FULLYDYNAMICSUBMODULARCOVER

```
1: $\pi \leftarrow$ arbitrary initial permutation of elements $\mathcal{N}$.
2: for $t = 1, 2, \ldots, T$ do
3:     $t^{th}$ function $g_t$ arrives/departs.
4:     while there exists a legal $\gamma$-move or a swap for $\pi$ do
5:         Perform the move, and update $\pi$.
6:     Output the collection of $\pi_i$ such that $\mathfrak{F}_\pi(\pi_i) > 0$.
```

6.2.2 Bounding the Cost

Let us start by arguing that if the algorithm terminates, it must produce a competitive solution.

Lemma 6.2.2. *Suppose no γ-moves are possible, then for every index i such that $\mathfrak{F}_\pi(\pi_i) > 0$, and for every index $i' > i$, the following holds. Let π' be the permutation where $\pi_{i'}$ is moved to position i. Then*

$$\mathfrak{F}_\pi(\pi_i) > \frac{\mathfrak{F}_{\pi'}(\pi_{i'})}{\gamma} \tag{6.2.2}$$

Proof. Suppose there are elements π_i and $\pi_{i'}$ such that condition (6.2.2) does not hold, i.e. $\mathfrak{F}_{\pi'}(\pi_j) \geq \gamma \cdot \mathfrak{F}_\pi(\pi_i)$. Since by assumption there are no swaps available, the permutation π is in non-increasing order of $\mathfrak{F}_\pi(\pi_i)$ values, so for all indices $j > i$ it also holds that $\mathfrak{F}_\pi(\pi_{i'}) \geq \gamma \cdot \mathfrak{F}_\pi(\pi_j)$. Hence moving i' from its current position to position i is a legal γ-move, which is a contradiction. □

Corollary 6.2.3. *The output at every time step is $\gamma \cdot (\log f^{(t)}_{\max}/f^{(t)}_{\min} + 1)$-competitive.*

Proof. Lemma 6.2.2 implies that the solution is equivalent to greedily selecting an element whose marginal coverage is within a factor of $1/\gamma$ of the largest marginal coverage currently available. Given this, the standard analyses of the greedy algorithm for SUBMODULARCOVER [Wol82] imply that the solution is $\gamma \cdot (\log f^{(t)}_{\max}/f^{(t)}_{\min} + 1)$ competitive. □

6.2.3 Bounding the Recourse

We move on to showing that the algorithm must terminate with $O(g(\mathcal{N})/f_{\min})$ amortized recourse. For this analysis, we define a potential function parameterized by a number $\alpha \in (0,1)$ to be fixed later:

$$\Phi_\alpha(f,\pi) := \sum_{i\in N} \left(\mathfrak{F}_\pi(\pi_i)\right)^\alpha .$$

As noted in the introduction, up to scaling and shifting, this is the Tsallis entropy with parameter α. We show several properties of this potential:

Properties of Φ_α:

(I) Φ_α increases by at most $g_t(\mathcal{N}) \cdot (f_{\min})^{\alpha-1}$ with the addition of function g_t to the active set.

(II) Φ_α does not increase with deletion of functions from the system.

(III) Φ_α does not increase during swaps.

(IV) For an appropriate range of γ, the potential Φ_α decreases by at least $(\gamma/(e\log\gamma)-1)\cdot(f_{\min})^\alpha = \Omega((f_{\min})^\alpha)$ with every γ-move.

Lemma 6.2.4. *If $\alpha = (\log\gamma)^{-1}$, then Φ_α respects properties (I)–(IV).*

Proof. For brevity, define $h(z) := z^\alpha$. Since $\alpha \in (0,1)$, this function is concave and non-decreasing.

We start with property (I). When a function g is added to the system, for some set of $i \in [n]$, it increases $k_i := \mathfrak{F}_\pi(\pi(i))$ by some amount Δ_i. Observe that $\sum_i \Delta_i = g(\mathcal{N})$. By the concavity of h:

$$\sum_i h(k_i+\Delta_i) - \sum_i h(k_i) \le \sum_i h(\Delta_i) \le \frac{\sum_i \Delta_i}{f_{\min}} \cdot h(f_{\min}) = g(\mathcal{N})\cdot (f_{\min})^{\alpha-1}.$$

Property (II) follows since h is non-decreasing.

For property (III), we wish to show that if u immediately precedes v in π and $\mathfrak{F}_\pi(u) \le \mathfrak{F}_\pi(v)$, then swapping u and v does not increase the potential. Let $\widehat{\pi}$ denote the permutation after the swap. Note that $\mathfrak{F}_{\widehat{\pi}}(u) \le \mathfrak{F}_\pi(v)$ and $\mathfrak{F}_{\widehat{\pi}}(v) \ge \mathfrak{F}_\pi(v)$, since u may only have lost some amount of coverage to v. Suppose this amount is k, i.e. $k = \mathfrak{F}_\pi(u) - \mathfrak{F}_{\widehat{\pi}}(u) = \mathfrak{F}_{\widehat{\pi}}(v) - \mathfrak{F}_\pi(v)$. Then:

$$\begin{aligned}\Phi_\alpha(f,\widehat{\pi}) - \Phi_\alpha(f,\pi) &= h(\mathfrak{F}_{\widehat{\pi}}(u)) + h(\mathfrak{F}_{\widehat{\pi}}(v)) - h(\mathfrak{F}_\pi(u)) - h(\mathfrak{F}_\pi(v)) \\ &= h(\mathfrak{F}_\pi(u)-k) + h(\mathfrak{F}_\pi(v)+k) - h(\mathfrak{F}_\pi(u)) - h(\mathfrak{F}_\pi(v))\end{aligned}$$

which is non-positive due to the concavity of h.

It remains to prove property (IV). Suppose we perform a γ-move on a permutation π. Let u be the element moving to some position p from some position $q > p$, and let π' denote the permutation after the move. For convenience, also define:

$$\begin{aligned} v_i &:= \mathfrak{F}_\pi(\pi_i), && \text{(the original coverage of the } i^{th} \text{ set)} \\ a_i &:= \mathcal{I}_f(\pi_i; u \mid \pi_{1:i-1}) = \mathfrak{F}_\pi(\pi_i) - \mathfrak{F}_{\pi'}(\pi_i). && \text{(the loss in coverage of the } i^{th} \text{ set)}\end{aligned}$$

Note that for all $i \notin \{p, \dots, q\}$, we necessarily have $a_i = 0$. Also note that $\mathfrak{F}_{\pi'}(S) = \sum_i a_i$, and by the definition of a γ-move, for any j we have $\sum_i a_i \geq \gamma \cdot v_j$. Then the change in potential is:

$$\begin{aligned}
\Phi_\alpha(f, \pi') - \Phi_\alpha(f, \pi) &= \Big(\sum_i a_i\Big)^\alpha + \sum_i (v_i - a_i)^\alpha - \sum_i v_i^\alpha \\
&\leq \Big(\sum_i a_i\Big)^\alpha - \sum_i a_i \cdot \alpha \cdot v_i^{\alpha-1} && (6.2.3) \\
&\leq \Big(\sum_i a_i\Big)^\alpha - \Big(\sum_i a_i\Big)^\alpha \cdot \alpha \cdot \gamma^{1-\alpha} && (6.2.4) \\
&\leq -\Big(\frac{\gamma}{e \log \gamma} - 1\Big)(f_{\min})^\alpha. && (6.2.5)
\end{aligned}$$

Above, (6.2.3) holds since h is concave and thus $h(v_i - a_i) - h(v_i) \leq \nabla h(v_i) \cdot (-a_i)$. Line (6.2.4) holds by the definition of a γ-move, since $\sum_i a_i \geq \gamma v_j$ for every $j \in \{p, \dots, q\}$. Finally, (6.2.5) comes from our choice of $\alpha = (\log \gamma)^{-1}$ and the fact that $\sum_i a_i \geq f_{\min}$. □

Finally, we return to proving the main theorem.

Proof of Theorem 6.2.1. By Lemma 6.2.2, if Algorithm 11 (using Definition 6.2.1 for $\mathfrak{F}_\pi$) terminates then it is $\gamma \cdot (\log f_{\max}^{(t)}/f_{\min}^{(t)} + 1)$-competitive.

By **(I)**–**(IV)**, the potential Φ_α increases by at most $g_t(\mathcal{N})(f_{\min})^{\alpha-1}$ for every function g_t inserted to the active set, decreases by $(f_{\min})^\alpha \cdot (\gamma/(e \log \gamma) - 1)$ per γ-move, and otherwise does not increase. By inspection, $\Phi_\alpha \geq 0$. The number of elements e with $\mathfrak{F}_\pi(e) > 0$ grows by 1 only during γ-moves in which $\mathfrak{F}_\pi(e)$ was initially 0. Otherwise, this number never grows. We account for elements leaving the solution by paying recourse 2 upfront when they join the solution.

Hence, the number of changes to the solution is at most:

$$2 \cdot \frac{\sum_t g_t(\mathcal{N})}{(f_{\min})^{1-\alpha}} \cdot \frac{e \log \gamma}{(f_{\min})^\alpha (\gamma - e \log \gamma)} = O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}}\right). \qquad \square$$

Our algorithm gives a tunable tradeoff between approximation ratio and recourse depending on the choice of γ. Note that if we wish to optimize the competitive ratio, setting $\gamma = e(1+\delta)$ gives a recourse bound of

$$\left[\left(1 + \frac{\log(1+\delta)}{\delta - \log(1+\delta)}\right)\right] \frac{\sum_t g_t(\mathcal{N})}{f_{\min}} = O\left(\frac{1}{\delta^2}\right) \frac{\sum_t g_t(\mathcal{N})}{f_{\min}}$$

as δ approaches 0. For simplicity one can set $\gamma = e^2$ to get the bound in Theorem 6.2.1.

6.3 General Cost SubmodularCover

6.3.1 The Algorithm

We now turn to the general costs case and show the main result of our chapter:

Theorem 6.3.1. *There is an* $e^2 \cdot (\log f^{(t)}_{\max}/f^{(t)}_{\min} + 1)$*- competitive deterministic algorithm for* DYNAMICSUBMODULARCOVER *with total recourse:*

$$O\left(\sum_t \frac{g_t(\mathcal{N})}{f_{\min}} \log\left(\frac{c_{\max}}{c_{\min}}\right)\right).$$

Given the last section, our description of the new algorithm is very simple. We reuse Algorithm 11, except we redefine:

$$\mathfrak{F}_\pi(\pi_i) := \frac{f(\pi_i \mid \pi_{1:i-1})}{c(\pi_i)}.$$

We will specify the last free parameter γ shortly.

6.3.2 Bounding the Cost

To bound the competitive ratio, note that (6.2.2) did not use any particular properties of $\mathfrak{F}_\pi$, except that in the solution output by the algorithm, there are no γ-moves or swaps with respect to $\mathfrak{F}_\pi$ in permutation π. The analog of Corollary 6.2.3 is nearly identical:

Corollary 6.3.2. *The output at every time step is* $\gamma \cdot (\log f^{(t)}_{\max}/f^{(t)}_{\min} + 1)$*-competitive.*

Proof. Lemma 6.2.2 implies that the solution is equivalent to greedily selecting an element whose marginal coverage/cost ratio is within a factor of $1/\gamma$ of the largest marginal coverage/cost ratio currently available. The standard analyses of the greedy algorithm for SUBMODULARCOVER [Wol82] imply that the solution is $\gamma \cdot (\log f^{(t)}_{\max}/f^{(t)}_{\min} + 1)$ competitive. □

6.3.3 Bounding the Recourse

To make our analysis as modular as possible, we will write a general potential function, parameterized by a function $h : \mathsf{R} \to \mathsf{R}$:

$$\Phi_h(f, \pi) := \sum_{i \in N} c(\pi_i) \cdot h\left(\mathfrak{F}_\pi(\pi_i)\right).$$

With foresight, we require several properties of h:

Properties of h:

- **(i)** h is monotone and concave.
- **(ii)** $h(0) = 0$.
- **(iii)** h satisfies $x \cdot h'(x/\gamma) \geq (1 + \epsilon_\gamma)h(x)$.
- **(iv)** h satisfies $y \cdot h(x/y)$ is monotone in y.

To bound the recourse, our goal will again be to show the following properties of our potential function Φ_h:

Properties of Φ_h:

(I) Φ_h increases by at most
$$\frac{g_t(\mathcal{N})}{f_{\min}} \cdot c_{\max} \cdot h\left(\frac{f_{\min}}{c_{\max}}\right)$$
with the addition of function g_t to the active set.

(II) Φ_h does not increase with deletion of functions from the system.

(III) Φ_h does not increase during sorting.

(IV) For an appropriate range of γ, the potential Φ_h decreases by at least
$$\epsilon_\gamma \cdot c_{\min} \cdot h\left(\frac{f_{\min}}{c_{\min}}\right)$$
with every γ-move.

Together, the statements imply a total recourse bound of:
$$\frac{\sum_t g_t(\mathcal{N})}{\epsilon_\gamma \cdot f_{\min}} \cdot \frac{c_{\max}}{c_{\min}} \cdot \frac{h(f_{\min}/c_{\max})}{h(f_{\min}/c_{\min})}.$$

Lemma 6.3.3. *If h respects properties (i)–(iv) then Φ_h respects properties (I)–(IV).*

Proof. We start with property (I). When a function g_t is added to the system, for some set of $i \in [n]$, it increases $k_i := \mathfrak{F}_\pi(\pi(i))$ by some amount Δ_i. Then the potential increase is:

$$\begin{aligned}
\Phi_h(f^{(t)}, \pi) - \Phi_h(f^{(t-1)}, \pi) &= \sum_{i \in [n]} c(\pi_i) \cdot h\left(\frac{k_i + \Delta_i}{c(\pi_i)}\right) - \sum_{i \in [n]} c(\pi_i) \cdot h\left(\frac{k_i}{c(\pi_i)}\right) \\
&\leq \sum_{i \in [n]} c(\pi_i) \cdot h\left(\frac{\Delta_i}{c(\pi_i)}\right) \qquad (6.3.1) \\
&\leq \sum_{i \in [n]} c_{\max} \cdot h\left(\frac{\Delta_i}{c_{\max}}\right) \qquad (6.3.2) \\
&\leq \frac{\sum_i \Delta_i}{f_{\min}} \cdot c_{\max} \cdot h\left(\frac{f_{\min}}{c_{\max}}\right) \qquad (6.3.3) \\
&= \frac{g_t(\mathcal{N})}{f_{\min}} \cdot c_{\max} \cdot h\left(\frac{f_{\min}}{c_{\max}}\right).
\end{aligned}$$

Above step (6.3.1) is by properties (i) and (ii), step (6.3.2) is by property (iv) and step (6.3.3) is by property (i).

Property (II) follows since h is non-decreasing.

The proof of property (III) is similar to the one in Lemma 6.2.4. Suppose u immediately precedes v in π but $\mathfrak{F}_\pi(u) \leq \mathfrak{F}_\pi(v)$, and let $\widehat{\pi}$ denote the permutation after the swap. We have that $\mathfrak{F}_{\widehat{\pi}}(u) \leq \mathfrak{F}_\pi(v)$ and $\mathfrak{F}_{\widehat{\pi}}(v) \geq \mathfrak{F}_\pi(v)$, since u may only have lost some amount of coverage to v. Suppose this amount is k, i.e. $k = \mathfrak{F}_\pi(u) - \mathfrak{F}_{\widehat{\pi}}(u) = \mathfrak{F}_{\widehat{\pi}}(v) - \mathfrak{F}_\pi(v)$. Then:

$$\Phi_h(f, \widehat{\pi}) - \Phi_h(f, \pi) = c(u) \cdot h\left(\mathfrak{F}_{\widehat{\pi}}(u)\right) + c(v) \cdot h(\mathfrak{F}_{\widehat{\pi}}(v)) - c(u) \cdot h(\mathfrak{F}_\pi(u)) - c(v) \cdot h(\mathfrak{F}_\pi(v))$$

$$
\begin{aligned}
&= c(u)\left(h\left(\mathfrak{F}_\pi(u) - \frac{k}{c(u)}\right) - h(\mathfrak{F}_\pi(u))\right) \\
&\quad + c(v)\left(h\left(\mathfrak{F}_\pi(v) + \frac{k}{c(v)}\right) - h(\mathfrak{F}_\pi(v))\right) \\
&\le k \cdot (h'(\mathfrak{F}_\pi(v)) - h'(\mathfrak{F}_\pi(u)))
\end{aligned}
$$

which is non-positive due to the concavity of h and the fact that $\mathfrak{F}_\pi(v) \ge \mathfrak{F}_\pi(u)$.

Finally the proof of property **(IV)** is also similar to the version in the last section. Suppose we perform a γ-move on a permutation π. Let u be the element moving to some position p from some position $q > p$, and let π' denote the permutation after the move. Then:

$$
\begin{aligned}
\Phi_h(f,\pi') - \Phi_h(f,\pi) &= c(u)\cdot h\left(\frac{\sum_{i\in[n]} a_i}{c(u)}\right) + \sum_{i\in[n]} c(\pi_i)\cdot\left(h\left(\frac{v_i - a_i}{c(\pi_i)}\right) - h\left(\frac{v_i}{c(\pi_i)}\right)\right) \\
&\le c(u)\cdot h\left(\frac{\sum_{i\in[n]} a_i}{c(u)}\right) - \sum_{i\in[n]} a_i \cdot h'\left(\frac{v_i}{c(\pi_i)}\right) && (6.3.4) \\
&\le c(u)\cdot h\left(\frac{\sum_{i\in[n]} a_i}{c(u)}\right) - c(u)\sum_{i\in[n]} \frac{a_i}{c(u)} \cdot h'\left(\frac{\sum_{i\in[n]} a_i}{\gamma\cdot c(u)}\right) && (6.3.5) \\
&\le c(u)\cdot h\left(\frac{\sum_{i\in[n]} a_i}{c(u)}\right) - (1+\epsilon_\gamma)\cdot c(u)\cdot h\left(\frac{\sum_{i\in[n]} a_i}{c(u)}\right) && (6.3.6) \\
&\le -\epsilon_\gamma \cdot c(u)\cdot h\left(\frac{\sum_{i\in[n]} a_i}{c(u)}\right) \\
&\le -\epsilon_\gamma \cdot c_{\min}\cdot h\left(\frac{f_{\min}}{c_{\min}}\right). && (6.3.7)
\end{aligned}
$$

Above step (6.3.4) uses the concavity of h. Step (6.3.5) uses the fact that moving u was a legal γ-move and thus $\sum_{i\in[n]} a_i/c(u) \ge \gamma v_i/c(\pi_i)$. Step (6.3.6) follows from property **(iii)**. Finally, (6.3.7) uses property **(iv)** again. □

With this setup, the proof of the main theorem boils down to identifying an appropriate function h, and a suitable choice of γ.

Proof of Theorem 6.3.1. Recall the general recourse bound is

$$
\frac{\sum_t g_t(\mathcal{N})}{\epsilon_\gamma \cdot f_{\min}} \cdot \frac{c_{\max}}{c_{\min}} \cdot \frac{h(f_{\min}/c_{\max})}{h(f_{\min}/c_{\min})}.
$$

A good choice for h is $h(x) = x^{1-\delta}/(1-\delta)$ for $\delta = (\log(c_{\max}/c_{\min}) + 1)^{-1}$, and with $\gamma = e^2$. Properties **(i)**–**(iv)** are easy to verify. To see that this implies the bound

$$
O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}} \log\left(\frac{c_{\max}}{c_{\min}}\right)\right),
$$

note that $\gamma = e^2 \ge ((1+\delta)/(1-\delta))^{1/\delta}$, in which case $\epsilon_\gamma = \gamma^\delta(1-\delta) - 1 \ge \delta$. Furthermore, $(c_{\max}/c_{\min})^\delta = O(1)$. □

6.4 Improved bounds for 3-increasing functions

In this section we show to remove the $\log(c_{\max}/c_{\min})$ dependence from the recourse bound when f is assumed to be from a structured class of set functions: the class of 3-increasing functions. We start with some definitions.

6.4.1 Higher Order Monotonicity of Boolean Functions

Recall the definition of m-increasing functions Definition 2.2.3.

We will soon show improved algorithms for the class $\mathcal{D}_3^+$, that is the class of 3-increasing set functions. The following derivation gives some intuition for these functions:

$$\begin{aligned}\frac{df}{d\{x,y,z\}}(S) &= \sum_{B\subseteq\{x,y,z\}} (-1)^{|B|} f((S\cup\{x,y,z\})\backslash B)\\ &= f(x\mid S) - f(x\mid S\cup\{y\}) - (f(x\mid S\cup\{z\}) - f(x\mid S\cup\{y,z\}))\\ &= \mathcal{I}_f(x,y\mid S) - \mathcal{I}_f(x,y\mid S\cup\{z\}). \qquad (6.4.1)\end{aligned}$$

Thus a function f in contained in $\mathcal{D}_3^+$ if and only if mutual coverage decreases after conditioning.

Bach [Bac13, Section 6.3] shows that a nonnegative function is in $\mathcal{D}_{2m-1}^+ \cap \mathcal{D}_{2m}^-$ simultaneously for all $m \geq 1$ if and only if it is a *measure coverage function*[2]: each element $i \in \mathcal{N}$ is associated with some measurable set S_i under a measure μ, and $f(S_1,\ldots,S_t) = \mu\left(\bigcup_{i=1}^t S_i\right)$.

6.4.2 The Algorithm

We now show our second main result:

Theorem 6.4.1. *There is an $O(\log f(N)/f_{\min})$-competitive deterministic algorithm for* DYNAMICSUBMODULARCOVER*, in the setting where the g^t function are 3-increasing in addition to monotone and submodular, with total recourse:*

$$O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}}\right).$$

We reuse Algorithm 11 from previous sections, only this time we redefine $\mathfrak{F}_\pi$ more substantially. Given two permutations α and β on $\mathcal{N}$, we define the (i,j) *mutual affinity of* (α,β) (and its conditional variant) as

$$\begin{aligned}\mathfrak{I}_{\alpha,\beta}(\alpha_i,\beta_j) &:= \mathcal{I}_f(\alpha_i,\beta_j \mid \alpha_{1:i-1}\cup\beta_{1:j-1}),\\ \mathfrak{I}_{\alpha,\beta}(\alpha_i,\beta_j\mid S) &:= \mathcal{I}_f(\alpha_i,\beta_j \mid \alpha_{1:i-1}\cup\beta_{1:j-1}\cup S).\end{aligned}$$

Recall that $\mathcal{I}_f$ denotes the Mutual Coverage. To give some insight into these definitions, observe that mutual affinity telescopes cleanly:

[2]We avoid the term *set coverage function* used by [Bac13], since we already use this terminology to mean the special case of a counting measure defined by a finite set system, as in SETCOVER.

Observation 6.4.2. *The chain rule implies that*

$$\sum_{j\in[n]} \mathfrak{I}_{\alpha,\beta}(\alpha_i,\beta_j) = f(\alpha_i \mid \alpha_{1:i-1}),$$
$$\sum_{i\in[n]} \mathfrak{I}_{\alpha,\beta}(\alpha_i,\beta_j) = f(\beta_j \mid \beta_{1:j-1}),$$
$$\sum_{i,j\in[n]} \mathfrak{I}_{\alpha,\beta}(\alpha_i,\beta_j) = f(\mathcal{N}).$$

Let ψ denote the ordering of $\mathcal{N}$ in *increasing order of cost*. Then:

$$\mathfrak{F}_\pi(\pi_i) := \sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i,\psi_j)}{c(\pi_i)\cdot c(\psi_j)}. \tag{6.4.2}$$

The following observation gives some intuition for this definition.

Observation 6.4.3. *The expression $\mathfrak{I}_{\alpha,\beta}(\alpha_i,\beta_j)$ is nonzero only if (a) element α_i precedes element β_j in permutation α, and also (b) element β_j precedes element α_i in permutation β.*

In light of these remarks, $\mathfrak{F}_\pi(\pi_i)$ decomposes the marginal coverage of π_i into its mutual affinity with all the elements $\psi_j \in \mathcal{N}$ that are simultaneously cheaper than π_i and that follow π_i in the permutation π, and weights each of these affinities by $(c(\pi_i)c(\psi_j))^{-1}$.

With this re-definition of $\mathfrak{F}_\pi$, Algorithm 11 is fully specified (though it is still parametrized by γ). We move to proving formal guarantees.

6.4.3 Bounding the Cost

Since our definition of $\mathfrak{F}_\pi$ (and thus the behavior of the algorithm) has significantly changed, we need to reprove the competitive ratio guarantee. Our goal will be the following Lemma:

Lemma 6.4.4. *If no swaps or γ-moves are possible, the solution is $\gamma^2 \cdot \log(f^{(t)}(\mathcal{N})/f^{(t)}_{\min})$-competitive.*

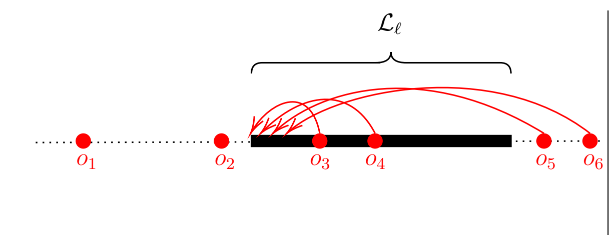
For the remainder of Section 6.4.3, we drop the superscript and refer to $f^{(t)}$ as simply f.

Define a level $\mathcal{L}_\ell$ be the collection of elements u such that $\mathfrak{F}_\pi(u) \in [\gamma^\ell, \gamma^{\ell+1})$. Note that since no swaps were possible, permutation π is sorted in decreasing order of $\mathfrak{F}_\pi$, and thus $\mathcal{L}_\ell$ forms a contiguous interval of indices in π. Our proof strategy will be to show that for any level ℓ, the total cost of all elements in $\mathcal{L}_\ell$ is at most $\gamma^2 \cdot c(\textsc{Opt})$. Subsequently, we will argue that there are at most $O(\log f(\mathcal{N})/f_{\min})$ non-trivial levels.

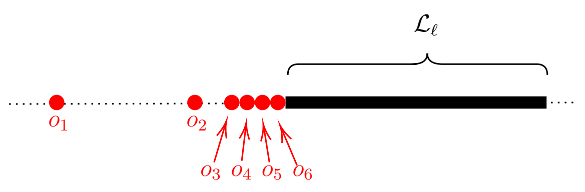
Lemma 6.4.5 (Each Level is Inexpensive)**.** *If there are no swaps or γ-moves for π, then for any $\ell > 0$, the total cost of all elements in $\mathcal{L}_\ell$ is at most $\gamma^2 \cdot c(\textsc{Opt})$.*

Proof. Suppose some level $\mathcal{L}_\ell$ has cost $c(\mathcal{L}_\ell) > \gamma^2 \cdot c(\textsc{Opt})$. We will argue that in this case there must be a legal γ-move available. To start, using Fact 2.1.1 we observe that:

$$\gamma^\ell \le \min_{u\in\mathcal{L}_\ell} \mathfrak{F}_\pi(u) \overset{\text{(def)}}{=} \min_{u\in\mathcal{L}_\ell} \frac{\sum_{j\in[n]} \mathfrak{I}_{\pi,\psi}(u,\psi_j)/c(\psi_j)}{c(u)} \le \frac{\sum_{u\in\mathcal{L}_\ell}\sum_{j\in[n]} \mathfrak{I}_{\pi,\psi}(u,\psi_j)/c(\psi_j)}{\sum_{u\in\mathcal{L}_\ell} c(u)},$$



(a) Permutation π.



(b) Permutation π', with $\textsc{Opt}_\geq$ moved ahead of $\mathcal{L}_\ell$.

Figure 6.2: Illustration of the proof of Lemma 6.4.5.

which by rearranging means:

$$\sum_{u\in\mathcal{L}_\ell}\sum_{j\in[n]}\frac{\mathfrak{I}_{\pi,\psi}(u,\psi_j)}{c(\psi_j)} \geq \gamma^\ell\cdot c(\mathcal{L}_\ell) > \gamma^{\ell+2}\cdot c(\textsc{Opt}). \tag{6.4.3}$$

Let $\pi_<$ be the set of items preceding $\mathcal{L}_\ell$ in π, and $\pi_\geq$ be the set of items in or succeeding $\mathcal{L}_\ell$. We also define $\textsc{Opt}_\geq := \textsc{Opt}\cap\pi_\geq$.

First, we imagine moving all the elements of $\textsc{Opt}_\geq$, simultaneously and in order according to π, to positions just before $\mathcal{L}_\ell$. Let π' be this new permutation. We first show that some $o\in\textsc{Opt}$ has a high value $\mathfrak{F}_{\pi'}(o)$ after this move. Then we show that this element o also constitutes a potential γ-move in π, a contradiction.

For the first claim, we start by using Fact 2.1.1 again:

$$\begin{aligned}
\max_{o\in\textsc{Opt}_\geq}\mathfrak{F}_{\pi'}(o) &\geq \frac{\sum_{o\in\textsc{Opt}_\geq}\sum_{j\in[n]}\mathfrak{I}_{\pi',\psi}(o,\psi_j\mid\pi_<)/c(\psi_j)}{\sum_{o\in\textsc{Opt}_\geq}c(o)}\\
&= \frac{1}{c(\textsc{Opt}_\geq)}\cdot\sum_{o\in\textsc{Opt}_\geq}\sum_{j\in[n]}\frac{\mathfrak{I}_{\pi',\psi}(o,\psi_j\mid\pi_<)}{c(\psi_j)}\\
&= \frac{1}{c(\textsc{Opt}_\geq)}\cdot\sum_{j\in[n]}\frac{f_\psi(\psi_j\mid\pi_<)}{c(\psi_j)} \qquad (6.4.4)\\
&\geq \frac{1}{c(\textsc{Opt}_\geq)}\cdot\sum_{u\in\mathcal{L}_\ell}\sum_{j\in[n]}\frac{\mathfrak{I}_{\pi,\psi}(u,\psi_j\mid\pi_<)}{c(\psi_j)} \qquad (6.4.5)
\end{aligned}$$

The lines (6.4.4) and (6.4.5) above follow by Observation 6.4.2 (note that (6.4.5)) is an inequality because $\mathcal{L}_\ell$ may be a strict subset of $\pi_\geq$). Since $u\in\mathcal{L}_\ell\subseteq\pi_\geq$, this is:

$$\begin{aligned}
&= \frac{1}{c(\textsc{Opt}_\geq)}\cdot\sum_{u\in\mathcal{L}_\ell}\sum_{j\in[n]}\frac{\mathfrak{I}_{\pi,\psi}(u,\psi_j)}{c(\psi_j)} \qquad (6.4.6)\\
&> \gamma^{\ell+2}, \qquad (6.4.7)
\end{aligned}$$

where line (6.4.7) used the inequality (6.4.3). In summary, $\mathfrak{F}_{\pi'}(o)$ is a factor γ bigger than any of the $\mathfrak{F}_\pi(u)$ values for $u\in\mathcal{L}_\ell$.

However, since permutation π' is obtained by moving many elements of π and not a single γ-move, we need to show that moving this element o alone gives a legal γ-move in π. In fact, observe that we have not yet used that $f\in\mathcal{D}_3^+$: we will use it now.

Claim 6.4.6. *Let π^o be the permutation derived from π where a single element $o \in \textsc{Opt}_{\geq}$ is moved before $\mathcal{L}_\ell$. Then $\mathfrak{F}_{\pi^o}(o) \geq \mathfrak{F}_{\pi'}(o)$.*

Proof. Start from π' and move an element $o' \in \textsc{Opt}_{\geq}$ with $o' \neq o$ back to its original position in π. Call this permutation π''. Then:

$$\begin{aligned}
\mathfrak{F}_{\pi''}(o) &= \sum_{j\in[n]} \frac{\mathfrak{I}_{\pi'',\psi}(o,\psi_j)}{c(o)c(\psi_j)} \\
&= \sum_{j\in[n]} \frac{\mathfrak{I}_{\pi',\psi}(o,\psi_j) + (\mathfrak{I}_{\pi'',\psi}(o,\psi_j) - \mathfrak{I}_{\pi'',\psi}(o,\psi_j \mid \{o'\}))}{c(o)c(\psi_j)} \\
&\geq \sum_{j\in[n]} \frac{\mathfrak{I}_{\pi',\psi}(o,\psi_j)}{c(o)c(\psi_j)},
\end{aligned}$$

where the second equality used $\mathfrak{I}_{\pi',\psi}(o,\psi_j) = \mathfrak{I}_{\pi'',\psi}(o,\psi_j \mid \{o'\})$, and where the inequality used equation (6.4.1), that $\mathcal{I}_f(a,b) - \mathcal{I}_f(a,b \mid c) \geq 0$ for 3-increasing functions. Repeating this process inductively until all elements but o have been returned to their original positions in π yields the permutation π^o, which proves the claim. □

Now we can complete the proof of Lemma 6.4.5. This means that if $\mathfrak{F}_{\pi'}(o) \geq \gamma^{\ell+2}$, then there is a legal γ-move (namely the one which moves o ahead of $\mathcal{L}_\ell$), because by assumption every $u \in \pi_{\geq}$ has $\mathfrak{F}_\pi(u) \leq \gamma^{\ell+1}$. This contradicts the assumption that none existed. □

Next, we argue that the cost of all elements with very high or very low values of $\mathfrak{F}_\pi$ is small.

Lemma 6.4.7 (Extreme Values Lemma). *If there are no swaps or γ-moves for π, the following hold:*

- ***(i)*** *There are no elements u such that $0 < \mathfrak{F}_\pi(u) \leq f_{\min}/(\gamma \cdot (c(\textsc{Opt}))^2)$.*
- ***(ii)*** *The total cost of all elements u such that $\mathfrak{F}_\pi(u) \geq (f(\mathcal{N}))^2/(f_{\min} \cdot (c(\textsc{Opt}))^2)$ is at most $\sqrt{\gamma} \cdot \textsc{Opt}$.*

Proof of Lemma 6.4.7. To prove item **(i)**, we observe that if permutation π has no local moves, then every element u must have high enough $\mathfrak{F}_\pi(u)$ to prevent any elements that follow it from cutting it in line.

Claim 6.4.8. *If there are no swaps or γ-moves for π, then for every $\pi_i \in \mathcal{N}$, and every $j \in [n]$ such that $\mathfrak{I}_{\pi,\psi}(\pi_i,\psi_j) > 0$ we have:*

$$\mathfrak{F}_\pi(\pi_i) > \frac{f(\psi_j \mid \pi_{1:i-1})}{\gamma \cdot (c(\psi_j))^2}.$$

Proof. If there are no swaps or γ moves, then $\mathfrak{F}_{\pi'}(\psi_j) < \gamma \cdot \mathfrak{F}_{\pi_i}$ (where π' is the permutation obtained from π by moving the element ψ_j to the position ahead of u). Expanding definitions:

$$\mathfrak{F}_{\pi'}(\psi_j) \overset{\text{(def)}}{=} \sum_{j'\in[n]} \frac{\mathfrak{I}_{\pi',\psi}(\psi_j,\psi_{j'})}{c(\psi_j)c(\psi_{j'})} \geq \sum_{j'\in[n]} \frac{\mathfrak{I}_{\pi',\psi}(\psi_j,\psi_{j'})}{(c(\psi_j))^2} = \frac{f(\psi_j \mid \pi_{1:i-1})}{(c(\psi_j))^2}.$$

In the first inequality we used that $c(\psi_{j'}) \leq c(\psi_j)$ by Observation 6.4.3, in the second equality we used Observation 6.4.2. Rearranging terms yields the claim. □

Now item (i) follows by setting ψ_j to be the cheapest element that succeeds u in the permutation. Note that $c(\psi_j) \leq c(\textsc{Opt})$.

For item (ii), let S be the set of indices i with $\mathfrak{F}_\pi(\pi_i) \geq (f(\mathcal{N}))^2/(f_{\min} \cdot (c(\textsc{Opt}))^2)$. Then by Fact 2.1.1:

$$\frac{f(\mathcal{N})}{c(\textsc{Opt})\sqrt{f_{\min}}} \leq \min_{i\in S} \sqrt{\mathfrak{F}_\pi(\pi_i)} = \min_{i\in S} \frac{\mathfrak{F}_\pi(\pi_i)}{\sqrt{\mathfrak{F}_\pi(\pi_i)}} \tag{6.4.8}$$

$$\overset{\text{(def)}}{=} \frac{1}{c(\pi_i)} \sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j)}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\pi_i)}} \tag{6.4.9}$$

$$\leq \frac{1}{c(S)} \cdot \sum_{i\in S}\sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j)}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\pi_i)}}.$$

Rearranging to bound the cost:

$$\begin{aligned}
c(S) &\leq \frac{c(\textsc{Opt})\sqrt{f_{\min}}}{f(\mathcal{N})} \cdot \sum_{i\in S}\sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j)}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\pi_i)}} \\
&\leq \frac{\sqrt{\gamma}\cdot c(\textsc{Opt})\sqrt{f_{\min}}}{f(\mathcal{N})} \cdot \sum_{i\in S}\sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j)}{\sqrt{f(\psi_j \mid \pi_{1:i-1})}} && (6.4.10) \\
&\leq \frac{\sqrt{\gamma}\cdot c(\textsc{Opt})\sqrt{f_{\min}}}{f(\mathcal{N})} \cdot \sum_{i\in S}\sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j)}{\sqrt{f_{\min}}} \\
&\leq \frac{\sqrt{\gamma}\cdot c(\textsc{Opt})}{f(\mathcal{N})} \cdot \sum_{i\in S}\sum_{j\in[n]} \mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j) \\
&= \sqrt{\gamma}\cdot c(\textsc{Opt}). && (6.4.11)
\end{aligned}$$

Above, step (6.4.10) is due to Claim 6.4.8 again and noting that if the denominator is 0 then the numerator is also 0, and step (6.4.11) is due to Observation 6.4.2. □

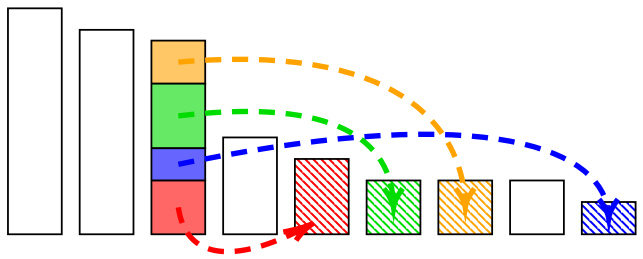
Using these ingredients, we can now wrap up the proof of the competitive ratio.

Proof of Lemma 6.4.4. The number of levels ℓ such that γ^ℓ lies between $f_{\min}/(\gamma(c(\textsc{Opt}))^2)$ and $(f(\mathcal{N}))^2/(f_{\min}\cdot(c(\textsc{Opt}))^2)$ is $O(\log_\gamma(f(\mathcal{N})/f_{\min})^2) = O(\log(f(\mathcal{N})/f_{\min}))$. By Lemma 6.4.5, each level in this range has cost at most $O(c(\textsc{Opt}))$. By Lemma 6.4.7, there are no elements with non-zero coverage in levels below this range, and the total cost of all elements above this range is $O(c(\textsc{Opt}))$. Thus the total cost of the solution is $O(\log(f(\mathcal{N})/f_{\min}))\cdot c(\textsc{Opt})$. □

6.4.4 Bounding the Recourse

We follow our recipe of using the modified Tsallis entropy as a potential (with a reminder that the underlying definitions of $\mathfrak{F}_\pi$ have changed) with α fixed to $1/2$:

$$\Phi_{1/2}(\pi) := \sum_{i=1}^{n} c(\pi_i)\sqrt{\mathfrak{F}_\pi(\pi_i)}.$$



$$\Phi_{1/2} = \sum_i \sqrt{\frac{c(\pi_i)}{c(\psi_1)} \cdot \mathfrak{I}_\pi(\pi_i;\psi_1) + \frac{c(\pi_i)}{c(\psi_2)} \cdot \mathfrak{I}_\pi(\pi_i;\psi_2) + \frac{c(\pi_i)}{c(\psi_3)} \cdot \mathfrak{I}_\pi(\pi_i;\psi_3) + \frac{c(\pi_i)}{c(\psi_4)} \cdot \mathfrak{I}_\pi(\pi_i;\psi_4) + \ \ldots}$$

Figure 6.3: Illustration of $\Phi_{1/2}$. Elements are arranged in order of π.

It is worthwhile to interpret these quantities in the context of HYPERGRAPHVERTEXCOVER. In this case, ψ_i and ψ_j correspond to vertices. Let $\Gamma(v)$ (and $\Gamma(V)$) denote the edge-neighborhood of v (and the union of the edge neighborhoods of vertices in the set V), then

$$\mathfrak{I}_{\pi,\psi}(\pi_i, \psi_j) = |(\Gamma(\pi_i) \cap \Gamma(\psi_j)) \backslash (\Gamma(\pi_{1:i-1} \cup \psi_{1:j-1}))|,$$

and if we use $c(e)$ to denote the cost of the cheapest vertex hitting edge e, we can simplify

$$c(\pi_i) \cdot \sqrt{\mathfrak{F}_\pi(\pi_i)} := \sqrt{\sum_{e \in \Gamma(\pi_i) \backslash \Gamma(\pi_{1:i-1})} \frac{c(\pi_i)}{c(e)}}.$$

In words, we are reweighting each hyperedge by the ratio of costs between the current vertex covering it, and its cheapest possible vertex that could cover it. Intuitively, this means the potential will be high when many elements are in sets that are significantly more expensive than the cheapest sets they could lie in.

This time the properties of $\Phi_{1/2}$ we show are:

Properties of $\Phi_{1/2}$:

(I) $\Phi_{1/2}$ increases by at most $g(\mathcal{N})/\sqrt{f_{\min}}$ with every addition of a function to the system.

(II) $\Phi_{1/2}$ does not increase with deletion of functions from the system.

(III) $\Phi_{1/2}$ does not increase during swaps.

(IV) If $\gamma > 4$, then $\Phi_{1/2}$ decreases by at least $\Omega(\sqrt{f_{\min}})$ with every γ-move.

Together, these will yield a recourse bound of $\sum_t g_t(\mathcal{N})/f_{\min}$.

Lemma 6.4.9. $\Phi_{1/2}$ *satisfies property (I).*

Proof. Consider a step in which f changes to f' because the function g was added to the system. For convenience, let $\widehat{\mathfrak{F}}_\pi$ and $\widehat{\mathfrak{I}}_{\pi,\psi}$ denote the quantities $\mathfrak{F}_\pi$ and $\mathfrak{I}_{\pi,\psi}$ after g has been added. Then the potential increase is:

$$\Phi_{1/2}(f^{(t)}, \pi) - \Phi_{1/2}(f^{(t-1)}, \pi) = \sum_{i \in [n]} c(\pi_i) \frac{\widehat{\mathfrak{F}}_\pi(\pi_i)}{\sqrt{\widehat{\mathfrak{F}}_\pi(\pi_i)}} - \sum_{i \in [n]} c(\pi_i) \frac{\mathfrak{F}_\pi(\pi_i)}{\sqrt{\mathfrak{F}_\pi(\pi_i)}}$$

$$= \sum_{i\in[n]}\sum_{j\in[n]} \frac{\widehat{\mathfrak{I}}_{\pi,\psi}(\pi_i,\psi_j)}{c(\psi_j)\sqrt{\widehat{\mathfrak{F}}_\pi(\pi_i)}} - \sum_{i\in[n]}\sum_{j\in[n]} \frac{\mathfrak{I}_{\pi,\psi}(\pi_i,\psi_j)}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\pi_i)}}.$$

Since $\mathfrak{F}_\pi(\pi_i)$ can only increase when a function g is added to the active set, this is

$$\begin{aligned}
&\leq \sum_{i\in[n]}\sum_{j\in[n]} \frac{\widehat{\mathfrak{I}}_{\pi,\psi}(\pi_i,\psi_j) - \mathfrak{I}_{\pi,\psi}(\pi_i,\psi_j)}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\pi_i)}} \\
&\overset{\text{(def)}}{=} \sum_{i\in[n]}\sum_{j\in[n]} \frac{\mathcal{I}_g(\pi_i,\psi_j \mid \pi_{1:i-1}\cup\psi_{1:j-1})}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\pi_i)}} \\
&\leq \sum_{j\in[n]} \frac{\sum_{i\in[n]}\mathcal{I}_g(\pi_i,\psi_j \mid \pi_{1:i-1}\cup\psi_{1:j-1})}{c(\psi_j)\sqrt{\mathfrak{F}_\pi(\psi_j)}} \qquad (6.4.12) \\
&= \sum_{j\in[n]} g(\psi_j \mid \psi_{1:j-1}) \left(\sum_{j'\in[n]} \frac{c(\psi_j)\mathfrak{I}_{\pi,\psi}(\psi_j,\psi_{j'})}{c(\psi_{j'})}\right)^{-1} \\
&\leq \sum_{j\in[n]} g(\psi_j \mid \psi_{1:j-1}) \left(\sum_{j'\in[n]} \mathfrak{I}_{\pi,\psi}(\psi_j,\psi_{j'})\right)^{-1} \qquad (6.4.13) \\
&\leq \frac{\sum_{j\in[n]} g(\psi_j \mid \psi_{1:j-1})}{\sqrt{f_{\min}}} \qquad (6.4.14) \\
&= \frac{g(N)}{\sqrt{f_{\min}}}.
\end{aligned}$$

Step (6.4.12) uses the relationship $\mathfrak{F}_\pi(\pi_i) \geq \mathfrak{F}_\pi(\psi_j)$, which holds because summands are nonzero only if ψ_j succeeds π_i in permutation π (by Observation 6.4.3), or the numerator is 0, and the fact that elements are sorted in decreasing order of $\mathfrak{F}_\pi$. Step (6.4.13) uses $c(\psi_{j'}) \leq c(\psi_j)$, also by Observation 6.4.3. Finally step (6.4.14) uses Observation 6.4.2. □

Lemma 6.4.10. *$\Phi_{1/2}$ satisfies property **(III)**.*

Proof. Consider an index i such that swapping π_i and π_{i+1} increases the potential. Then for some quantity $\delta := \sum_{j\in[n]}(\mathfrak{I}_{\pi,\psi}(\pi_i,\psi_j) - \mathfrak{I}_{\pi',\psi}(\pi_i,\psi_j))/c(j) > 0$ (since $f \in \mathcal{D}_3^+$) we have:

$$\begin{aligned}
&0 < \Delta\Phi_{1/2} \\
&0 < c(\pi_{i+1})\left(\sqrt{\mathfrak{F}_\pi(\pi_{i+1}) + \frac{\delta}{c(\pi_{i+1})}} - \sqrt{\mathfrak{F}_\pi(\pi_{i+1})}\right) + c(\pi_i)\left(\sqrt{\mathfrak{F}_\pi(\pi_i) - \frac{\delta}{c(\pi_i)}} - \sqrt{\mathfrak{F}_\pi(\pi_i)}\right) \\
&0 < \frac{\delta}{2\sqrt{\mathfrak{F}_\pi(\pi_{i+1})}} - \frac{\delta}{2\sqrt{\mathfrak{F}_\pi(\pi_i)}} \qquad (6.4.15) \\
&\Rightarrow \quad \mathfrak{F}_\pi(\pi_{i+1}) \leq \mathfrak{F}_\pi(\pi_i).
\end{aligned}$$

Above, (6.4.15) holds since square root is a concave function and thus $\sqrt{a+b} - \sqrt{a} \leq b/(2\sqrt{a})$. This implies that the local move was not a legal swap. □

Lemma 6.4.11. *If $\gamma > 4$, then $\Phi_{1/2}$ satisfies property **(IV)**.*

Proof. Suppose the local move changes the permutation π to π' by moving element u from position q to p. For notational convenience, define the following quantities:

$$\begin{aligned} v_i &:= \mathfrak{F}_\pi(\pi_i), \\ a_{ij} &:= \mathfrak{I}_\pi(\pi_i, \psi_j) - \mathfrak{I}_{\pi'}(\pi_i, \psi_j) \overset{\text{(def)}}{=} \mathbb{1}[p \le i \le q] \cdot (\mathfrak{I}_\pi(\pi_i, \psi_j) - \mathfrak{I}_\pi(\pi_i, \psi_j \mid \{u\})) . \end{aligned}$$

Recall that by (6.4.1), the quantity a_{ij} is (crucially) nonnegative when f is 3-increasing. Also, by expanding the definition of Mutual Coverage, for all indices $i \in \{p, \dots, q\}$ we can rewrite a_{ij} as:

$$\begin{aligned} a_{ij} &= f(\psi_j \mid \pi_{1:i-1} \cup \psi_{1:j-1}) - f(\psi_j \mid \pi_{1:i} \cup \psi_{1:j-1}) \\ &\quad - [f(\psi_j \mid \pi_{1:i-1} \cup \psi_{1:j-1} \cup \{u\}) - f(\psi_j \mid \pi_{1:i} \cup \psi_{1:j-1} \cup \{u\})] \\ &= \mathcal{I}_f(u, \psi_j \mid \pi_{1:i-1} \cup \psi_{1:j-1}) - \mathcal{I}_f(u, \psi_j \mid \pi_{1:i} \cup \psi_{1:j-1}). \end{aligned}$$

Thus, by the Chain Rule these terms telescopes such that:

$$\sum_{i \in \{p,\dots,q\}} a_{ij} = \mathcal{I}_f(u, \psi_j \mid \pi_{1:p-1}, \psi_{1:j-1}) \overset{\text{(def)}}{=} \mathfrak{I}_{\pi',\psi}(u, \psi_j).$$

With this, we can bound the potential change after a local move as

$$\begin{aligned} &\Phi_{1/2}(f, \pi') - \Phi_{1/2}(f, \pi) \\ &= c(u)\sqrt{\mathfrak{F}_{\pi'}(u)} + \sum_{i \in [n]} c(\pi_i) \sqrt{v_i - \sum_{j \in [n]} \frac{a_{ij}}{c(\pi_i)c(\psi_j)}} - \sum_{i \in [n]} c(\pi_i)\sqrt{v_i} \\ &\le c(u)\sqrt{\mathfrak{F}_{\pi'}(u)} - c(\pi_i) \sum_{i \in [n]} \frac{1}{2\sqrt{v_i}} \cdot \sum_{j \in [n]} \frac{a_{ij}}{c(\pi_i)c(\psi_j)} \qquad (6.4.16) \\ &\le c(u)\sqrt{\mathfrak{F}_{\pi'}(u)} - \frac{\sqrt{\gamma}}{2} \cdot \frac{c(\pi_i)}{\sqrt{\mathfrak{F}_{\pi'}(u)}} \cdot \sum_{i \in [n]} \sum_{j \in [n]} \frac{a_{ij}}{c(\pi_i)c(\psi_j)} \qquad (6.4.17) \\ &= c(u)\sqrt{\mathfrak{F}_{\pi'}(u)} - \frac{\sqrt{\gamma}}{2} \cdot \frac{c(u)}{\sqrt{\mathfrak{F}_{\pi'}(u)}} \cdot \sum_{j \in [n]} \frac{\sum_{i \in [n]} a_{ij}}{c(u)c(\psi_j)}. \end{aligned}$$

Above, step (6.4.16) holds since square root is a concave function and thus $\sqrt{a+b} - \sqrt{a} \le b/(2\sqrt{a})$. The next step (6.4.17) is due to the definition of γ-moves which ensure that $v_i \le \mathfrak{F}_{\pi'}(u)/\gamma$ (note that this step also makes use of the nonnegativity of a_{ij}). Using the telescoping property of the a_{ij}, we can continue:

$$\begin{aligned} &= c(u)\sqrt{\mathfrak{F}_{\pi'}(u)} - \frac{\sqrt{\gamma}}{2} \cdot \frac{c(u)}{\sqrt{\mathfrak{F}_{\pi'}(u)}} \cdot \sum_{j \in [n]} \frac{\mathfrak{I}_{\pi',\psi}(u, \psi_j)}{c(u)c(\psi_j)} \\ &\overset{\text{(def)}}{=} - \left(\frac{\sqrt{\gamma}}{2} - 1\right) \sqrt{\sum_{j \in [n]} \frac{c(u) \cdot \mathfrak{I}_{\pi',\psi}(u, \psi_j)}{c(\psi_j)}} \\ &\le - \left(\frac{\sqrt{\gamma}}{2} - 1\right) \sqrt{\sum_{j \in [n]} \mathfrak{I}_{\pi',\psi}(u, \psi_j)} \qquad (6.4.18) \end{aligned}$$

$$\leq -\left(\frac{\sqrt{\gamma}}{2} - 1\right)\sqrt{f_{\min}}. \tag{6.4.19}$$

Step (6.4.18) comes from Observation 6.4.3, and finally step (6.4.19) follows by Observation 6.4.2 and the fact that $f_{\min}$ is a lower bound on marginal coverage for any element with nonzero marginal coverage. $\square$

We wrap up with the proof of the main theorem.

Proof of Theorem 6.4.1. Set $\gamma = 5 > 4$. By Lemma 6.4.4, if Algorithm 11 (using Definition 6.4.2 for $\mathfrak{F}_\pi$) terminates then it is $O(\log f(N)/f_{\min})$-competitive.

By **(I)**–**(IV)**, the potential $\Phi_{1/2}$ increases by at most $g_t(\mathcal{N}/\sqrt{f_{\min}}$ for every function g_t inserted to the active set, decreases by $\sqrt{f_{\min}} \cdot \left(\sqrt{\gamma}/2 - 1\right)$ per γ-move, and otherwise does not increase. By inspection, $\Phi_\alpha \geq 0$. The number of elements e with $\mathfrak{F}_\pi(e) > 0$ grows by 1 only during γ-moves in which $\mathfrak{F}_\pi(e)$ was initially 0. Otherwise, this number never grows. We account for elements leaving the solution by paying recourse 2 upfront when they join the solution.

Hence, the number of changes to the solution is at most:

$$2 \cdot \frac{\sum_t g_t(\mathcal{N})}{\sqrt{f_{\min}}} \cdot \frac{2}{\sqrt{f_{\min}}(\sqrt{\gamma} - 2)} = O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}}\right). \qquad \square$$

6.5 Algorithm for r-bounded instances

We can achieve a better approximation ratio if each function g is an r-junta for small r. Recall that an r-junta is a function that depends on at most r variables. In this section we prove the theorem:

Theorem 6.5.1. *There is a randomized algorithm that maintains an r-competitive solution to* DYNAMICSUBMODULARCOVER*, in the setting where these functions are each r-juntas, with total recourse:*

$$\frac{\sum_t g_t(\mathcal{N})}{f_{\min}}.$$

Our proof follows the framework of [GKKP17], which is itself a dynamic implementation of the algorithm of [Pit85].

We start with some notation. Let $V_g = \{u \in \mathcal{N} \mid g(u) \neq 0\}$ be the elements influencing g. Our assumption says that $|V_g| \leq r$. Let S be the solution maintained by the algorithm. Each function $g \in G^{(t)}$ maintains a set $U_g \subseteq V_g$ of elements assigned to it. We say that g is *responsible* for U_g. We also define the following operation:

Probing a function. Sample one element $u \in V_g \backslash S$ with probability:

$$\frac{\frac{1}{c(u)}}{\sum_{v \in V_g \backslash S} \frac{1}{c(v)}}.$$

Add the sampled element u to the current solution S, and to U_g.

Given these definitions, we are ready to explain the dynamic algorithm.

Function arrival. When a function g arrives, initialize its element set U_g to $\emptyset$. Then, while $g(S) \neq g(\mathcal{N})$, probe g_t.

Function departure. When a function g departs, remove all its assigned elements U_g from S. This may leave some set of functions $g_1, \ldots, g_s$ uncovered. For each of these functions g_t in order of arrival, while $g_t(S) \neq g(\mathcal{N})$, probe g_t.

The $\sum_t g_t(\mathcal{N})/f_{\min}$ recourse bound is immediate, since the total number of probes can be at most $\sum_t g_t(\mathcal{N})/f_{\min}$ in total. It remains to bound the competitive ratio.

We prove the following:

Lemma 6.5.2. *For any element $u \in \mathcal{N}$:*

$$\mathbb{E}\left[\sum_{\substack{g \in G^{(t)}: \\ u \in V_g}} \sum_{v \in U_g} c(v)\right] \leq r \cdot c(u).$$

This will imply as a consequence:

$$\mathbb{E}[c(S)] \leq \sum_{o \in \textsc{Opt}_t} \mathbb{E}\left[\sum_{\substack{g \in G^{(t)}: \\ o \in V_g}} \sum_{v \in U_g} c(v)\right] \leq r \cdot \sum_{o \in \textsc{Opt}} c(o) = r \cdot c(\textsc{Opt}_t).$$

Proof of Lemma 6.5.2. The proof is by induction. Fix $u \in \mathcal{N}$, and consider the functions $g \in G^{(t)}$ for which $u \in V_g$. Let X_i be the random variable that is the i^{th} function probed. Let Y_i be the random variable that is the element of N sampled during the i^{th} probe. With this notation, the inductive hypothesis is:

$$\mathbb{E}\left[\sum_{i \geq j} c(Y_i) \mid X_1, Y_1, \ldots X_{j-1}, Y_{j-1}\right] \leq r \cdot c(u).$$

For the base case $i = m$, note that given $X_1, Y_1, \ldots, X_{m-1}, Y_{m-1}$, the variable X_m is determined. Suppose $X_m = g$. Then:

$$\begin{aligned}
\mathbb{E}[c(Y_m) \mid X_1, Y_1, \ldots X_{m-1}, Y_{m-1}] &= \mathbb{E}[c(Y_m) \mid X_1, Y_1, \ldots X_{m-1}, Y_{m-1}, X_m = g] \\
&= \sum_{v \in V_g} \frac{1}{c(v)\left(\sum_{v' \in V_g \setminus S_t} \frac{1}{c(v')}\right)} \cdot c(v) \\
&\leq \frac{|V_g|}{\sum_{v' \in V_g \setminus S_t} \frac{1}{c(v')}} \\
&\leq r \cdot c(u).
\end{aligned}$$

For the inductive step, suppose the claim holds for $j+1$, and consider the case for j:

$$\mathbb{E}\left[\sum_{i \geq j} c(Y_i) \mid X_1, Y_1, \ldots X_{j-1}, Y_{j-1}\right]$$

$$
\begin{aligned}
&= \mathbb{E}[c(Y_j) \mid X_1, Y_1, \dots X_{j-1}, Y_{j-1}] + \mathbb{E}\left[\sum_{i \geq j+1} c(Y_i) \mid X_1, Y_1, \dots X_{j-1}, Y_{j-1}\right] \\
&\leq \frac{r}{\sum_{v' \in V_g \setminus S_t} \frac{1}{c(v')}} + \sum_{u' \neq u} \mathbb{E}\left[\sum_{i \geq j+1} c(Y_i) \mid X_1, \dots Y_{j-1}, Y_j = u'\right] \cdot \mathbb{P}\left(Y_j = u' \mid X_1, \dots, Y_{j-1}\right) \\
&\leq \frac{r}{\sum_{v' \in V_g \setminus S_t} \frac{1}{c(v')}} + r \cdot c(u) \cdot \left(1 - \frac{1}{c(u) \sum_{v' \in V_g \setminus S_t} \frac{1}{c(v')}}\right) \\
&= r \cdot c(u). \qquad \square
\end{aligned}
$$

6.6 Combiner Algorithm

We show that we can adapt the combiner algorithm of [GKKP17] to our general problem.

Theorem 6.6.1. *Let A_G be an $O(\log f(N)/f_{\min})$-competitive algorithm for fully-dynamic* SUBMODULARCOVER *with amortized recourse R_G. Let A_{PD} be an $O(r)$-competitive algorithm for fully-dynamic* SUBMODULARCOVER *when all functions are r-juntas with amortized recourse R_{PD}. Then there is an algorithm achieving an approximation ratio of $O(\min(\log(f(N)/f_{\min}), r)$ for fully-dynamic* SUBMODULARCOVER *when all functions are r-juntas, and it has total recourse $O(R_G + R_{PD})$.*

Proof. The idea is to partition the functions into different buckets based on their junta-arity, in powers of 2 up to $\log f(N)/f_{\min}$. We run a copy of A_{PD} which we call $A_{PD}^{(\ell)}$ on each bucket B_ℓ separately, and run A_G one single time on the set of remaining functions.

Formally, for every index $0 < \ell < \lceil \log\log(f(N)/f_{\min}) \rceil$, maintain a bucket B_ℓ representing the set of functions g such that g is a k-junta, for $k \in [2^\ell, 2^{\ell+1})$. Also maintain the bucket B_G for any remaining functions. When a functions arrives, we insert it into exactly one appropriate bucket and update the appropriate algorithm.

Lemma 6.6.2. *The total cost of the solution maintained by the algorithm is $O(\min(\log f(N), r)$.*

Proof. If $r \leq \log(f(N)/f_{\min})$, the algorithm never runs A_G. Each algorithm $A_{PD}^{(\ell)}$ is $O(2^{\ell+1})$-competitive, and thus maintains a solution of cost no more than $O(2^{\ell+1})c(\textsc{Opt})$. The largest bucket index is $\ell_{\max} = \lceil \log r \rceil$. Hence the total cost of the solution is:

$$\sum_{\ell=1}^{\ell_{\max}} O(2^{\ell+1}) \cdot c(\textsc{Opt}) = O(r) \cdot c(\textsc{Opt}).$$

Otherwise if $r > \log(f(N)/f_{\min})$, then the largest bucket index is $\ell_{\max} = \lceil \log\log(f(N)/f_{\min}) \rceil$. The total cost of the A_{PD} algorithms is then

$$\sum_{\ell=1}^{\ell_{\max}} O(2^{\ell+1}) \cdot c(\textsc{Opt}) = O(\log f(N)/f_{\min}) \cdot c(\textsc{Opt}).$$

Meanwhile, the total cost of the solution maintained by A_G on the remaining functions has cost $O(\log f(N)/f_{\min}) \cdot c(\textsc{Opt})$. Thus the global solution maintained by the combiner algorithm is also $O(\log f(N)/f_{\min})$-competitive. $\square$

The recourse bound is immediate since each function g arrives to/departs from exactly one bucket, so at most one algorithm among $\{A_{PD}^{(\ell)}\}_\ell \cup \{A_G\}$ has to update its solution at every time step. $\square$

6.7 Further Applications

In this section, we show how to recover several known results on recourse bounded algorithms using our framework. We hope this is a step towards unifying the theory of low recourse dynamic algorithms.

6.7.1 Online Metric Minimum Spanning Tree

In this problem, vertices in a metric space are added online to an active set. Let A_t denote the active set at time t. After every arrival, the algorithm must add/remove edges to maintain a spanning tree S_t for A_t that is competitive with the MINIMUMSPANNINGTREE. We show:

Theorem 6.7.1. *There is a deterministic algorithm for Online Metric* MINIMUMSPANNINGTREE *that achieves a competitive ratio of $O(1)$ and an amortized recourse bound of $O(\log D)$, where D is the ratio of the maximum to minimum distance in the metric.*

[GGK16] show how to get an $O(1)$ worst case recourse bound.

MINIMUMSPANNINGTREE is a special case of SUBMODULARCOVER in which $\mathcal{N}$ is the set of edges of the graph, and f is the rank function of a graphic matroid. The main difference between the dynamic version of this problem and our setting is that here vertices arrive online *along with all their incident edges*. Hence not only is the submodular function changing, but $\mathcal{N}$ is also growing. We show that this detail can be handled easily.

We define the submodular function $f^{(t)}$ to be the rank function for the current graphic matroid, i.e. $f(S) = |A_t| - c_t$, where c_t is the number of connected components induced by S on the set of vertices seen thus far. Note that $f_{\max} = f_{\min} = 1$, so an edge having nonzero coverage is equivalent to the edge being in our current solution, S_t.

Now the algorithm is:

Algorithm 12 FULLYDYNAMICMST

```
1: $\pi \leftarrow$ arbitrary initial permutation of edges.
2: for $t = 1, 2, \dots, T$ do
3:     When vertex $v_t$ arrives, add edges incident to $v_t$ to tail of permutation in arbitrary order,
           and update $f^{(t)}$.
4:     while there exists a legal $\gamma$-move or a swap for $\pi$ do
5:         Perform the move, and update $\pi$.
6:     Output the collection of $\pi_i$ such that $\mathfrak{F}_\pi(\pi_i) > 0$.
```

As in Corollary 6.2.3, if the algorithm terminates then it represents the stack trace of an approximate greedy algorithm for MINIMUMSPANNINGTREE. Hence the solution is $O(1)$ competitive. To bound the recourse, we use the general potential Φ_h from Section 6.3. As before, local moves decrease the potential Φ_h by $\epsilon_\gamma \cdot c_{\min} \cdot h\left(\frac{1}{c_{\min}}\right)$, so it suffices to show that the potential does not increase by too much when $f^{(t)}$ is updated. Exactly one new edge will have increased marginal coverage, and its coverage will increase from 0 to 1. Thus the increase in potential is at most $c_{\max} \cdot h(1/c_{\max})$. Together, these imply an amortized recourse bound of:

$$\frac{1}{\epsilon_\gamma} \cdot \frac{c_{\max}}{c_{\min}} \cdot \frac{h(1/c_{\max})}{h(1/c_{\min})}.$$

Setting $h(x) = x^{1-\delta}/(1-\delta)$ along with $\delta = (\ln(c_{\max}/c_{\min}+1))^{-1}$ and $\gamma = \epsilon^2$ as in Theorem 6.3.1, we have $\epsilon_\gamma \geq \delta$, and hence we get a recourse bound of $O(\ln(c_{\max}/c_{\min})) = O(\ln D)$.

6.7.2 Fully-Dynamic Metric Minimum Steiner Tree

We show that we can also fit into our framework the harder problem of maintaining a tree that spans a set of vertices in the fully-dynamic setting where vertices can both arrive and depart. We must produce a tree S_t that spans the current set of active vertices A_t, but we allow ourselves to use Steiner vertices that are not in the active set. We show:

Theorem 6.7.2. *There is a deterministic algorithm for Fully-Dynamic Metric* MINIMUMSTEINERTREE *that achieves a competitive ratio of $O(1)$ and an amortized recourse bound of $O(\log D)$, where D is the ratio of the maximum to minimum distance in the metric.*

This guarantee matches that of [LOP+15]. Separately [GK14] showed how to improve the bound to $O(1)$ amortized recourse.

Our algorithm is the same local search procedure as before, with one twist. We maintain a set of vertices L we call the live set. This set is the union of the active terminals we need to span, and any Steiner vertices currently being used. We define $f^{(t)}$ similarly to before as $f^{(t)}(S) = |L| - c_t$, where c_t is the number of connected components induced by the edge set S on the set of vertices in L. Note that this function is submodular, because it is the rank function of the graphic matroid on the live vertex set L.

Now when a vertex v departs, we mark it as a Steiner vertex but leave it in the live set. If at any point during the local search $\deg(v) = 2$, we replace v with the edge that shortcuts between v's two neighbors. If at any point point $\deg(v) = 1$, we delete v and its neighboring edge.

Algorithm 13 FULLYDYNAMICSTEINERTREE

```
1:  $\pi \leftarrow$ arbitrary initial permutation of edges.
2:  for $t = 1, 2, \ldots, T$ do
3:      if vertex $v_t$ arrives then
4:          Add edges incident to $v_t$ to tail of permutation in arbitrary order, and update $f^{(t)}$.
5:      else if vertex $v_t$ departs then
6:          Mark $v_t$ as a Steiner vertex. Run CLEANSTEINERVERTICES.
7:      while there exists a legal $\gamma$-move or a swap for $\pi$ do
8:          Perform the move, and update $\pi$.
9:          Run CLEANSTEINERVERTICES.
10:     Output the collection of $\pi_i$ such that $\mathfrak{F}_\pi(\pi_i) > 0$.
```

```
1: procedure CleanSteinerVertices
2:     while there is a Steiner vertex $v$ with $\deg(v) = 2$ do
3:         Let $u_1$ and $u_2$ be the neighbors of $v$.
4:         Add the edge $(u_1, u_2)$ to the position of $(v, u_1)$ in $\pi$.        ▷ this shortcuts $v$
5:         Delete all edges incident to $v$ from $\mathcal{N}$, remove $v$ from the live set, and update $f^{(t)}$.
6:     while there is a Steiner vertex $v$ with $\deg(v) = 1$ do
7:         Delete all edges incident to $v$ from $\mathcal{N}$, remove $v$ from the live set, and update $f^{(t)}$.
```

To show the competitive ratio we can rely on known results [IW91, GK14]. If Algorithm 13 terminates, the output tree is known as a γ-*stable extension tree* for the terminal set S.

Lemma 6.7.3 (Lemma 5 of [IW91]). *If T is a γ-stable extension tree for A_t, then:*

$$c(T) \le 4\gamma \cdot c(\textsc{Opt}(A_t))$$

where $\textsc{Opt}(A_t)$ *is the optimal Steiner tree for terminal set A_t.*

Since we set $\gamma = e^2$, this gives us a competitive ratio of $4e^2 = O(1)$.

It remains to show the recourse bound. Deleting degree 1 and 2 vertices requires a constant number of edge changes, so this can be charged to each vertex's departure. We show that the potential argument from before is not hampered by the changes to the algorithm.

Claim 6.7.4. *The procedure* CleanSteinerVertices *does not increase the potential.*

Proof. When a degree 1 Steiner vertex is deleted, the incident edge is removed from the permutation and no other edge's marginal coverage changes.

When a degree 2 Steiner vertex is deleted, the edges (v, u_1) and (v, u_2) are replaced by the edge (u_1, u_2). Recall that our choice of potential is:

$$\Phi_h(\pi) = \sum_{e \in S_t} c(e)^\delta$$

for $0 < \delta < 1$. By triangle inequality, and concavity of h:

$$d(u_1, u_2)^\delta \le (d(v, u_1) + d(v, u_2))^\delta \le d(v, u_1)^\delta + d(v, u_2)^\delta.$$

Thus this replacement only decreases the potential. □

Otherwise, the potential increases during vertex arrivals and decreases during γ-moves exactly as in Section 6.7.1. We are left with the same recourse bound of $O(\ln D)$.

6.8 Conclusion

In this chapter, we gave an algorithm for fully-dynamic SubmodularCover which generalizes the work of [GKKP17] to arbitrary submodular covering problems. In analogy to Chapter 3, this ports any black box use of SubmodularCover to the dynamic setting. Though the problems studied in this chapter and in Chapter 3 are very similar, it is interesting that the techniques we use to achieve optimal results in both are completely unrelated.

The results in this chapter suggest several clear avenues for future work. One clear question is whether it is possible to remove the $\log(c_{\max}/c_{\min})$ in the recourse bound of Theorem 6.3.1. Another interesting orthogonal question is whether there is a fully-dynamic algorithms for SUBMODULARCOVER with fast *update time* (as opposed to just recourse). The naive implementation of our local search requires polynomial time, since it must greedily search for local moves, as well as preform bubble sort between any two such moves. Since known algorithms for dynamic SETCOVER with update time bounds ([GKKP17]) explicitly maintain an assignment of elements to sets, it is conceivable that one may need more than black box access to a value oracle. This question is interesting even for special cases of SUBMODULARCOVER such as partial cover, or the SIMULTANEOUSSOURCELOCATION problem we discuss in Chapter 1.

6.9 Bounds using the Shannon entropy potential

We show that Shannon entropy also works as a potential, albeit with the weaker recourse bound of:

$$O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}} \log\left(\frac{c_{\max}}{c_{\min}} \cdot \frac{f_{\max}}{f_{\min}}\right)\right).$$

Define the Shannon Entropy potential to be the expression:

$$\Phi_1(f,\pi) := \sum_{i\in N} \mathfrak{F}_\pi(\pi_i) \log \frac{c(\pi_i)}{\mathfrak{F}_\pi(\pi_i)}.$$

In order to ensure that Φ_1 remains nonnegative and monotone in each $\mathfrak{F}_\pi(\pi_i)$, scale c by $1/c_{\min}$ and f by $1/(e \cdot f_{\max})$ such that all costs are greater than 1 and all coverages are less than $1/e$. We will account for this scaling at the end.

Note that Φ_1 is Φ_h from Section 6.3 with $h(x) = x\log(1/x)$. This h satisfies properties **(i)**, **(ii)** and **(iv)** but not **(iii)**.

Properties of Φ_1:

(I) Φ_1 increases by at most $g_t(\mathcal{N}) \cdot \log(c_{\max}/f_{\min})$ with the addition of function g_t to the active set.

(II) Φ_1 does not increase with deletion of functions from the system.

(III) Φ_1 does not increase during swaps.

(IV) If $\gamma > e$, then Φ_1 decreases by at least $f_{\min}\log(\gamma/e)$ with every γ-move.

The proofs that Φ_1 satisfies properties **(I)**–**(III)** follows directly from Lemma 6.3.3, since these do not use property **(iii)**. It remains to show the last property.

Lemma 6.9.1. *If $\gamma > e$, every γ-move decreases Φ_1 by at least $f_{\min} \cdot \log(\gamma/\epsilon)$.*

Proof. Suppose we perform a γ-move on a permutation π. Let u be the element moving to some position p from some position $q > p$, and let π' denote the permutation after the move. For convenience, also define:

$v_i := \mathfrak{F}_\pi(\pi_i),$ (the original coverage of the i^{th} set)

$$a_i := \mathcal{I}_f(\pi_i; u \mid \pi_{1:i-1}) = \mathfrak{F}_\pi(\pi_i) - \mathfrak{F}_{\pi'}(\pi_i). \qquad \text{(the loss in coverage of the } i^{th} \text{ set)}$$

Then:

$$\begin{aligned}
&\Phi_1(f, \pi') - \Phi_1(f, \pi) \\
&= \sum_{i=1}^{n} (v_i - a_i) \log \frac{c(\pi_i)}{v_i - a_i} + \sum_{i=1}^{n} a_i \log \frac{c(u)}{\sum_{i=1}^{n} a_i} \\
&\quad - \sum_{i=1}^{n} v_i \log \frac{c(\pi_i)}{v_i} \\
&\le -\sum_{i=1}^{n} a_i \log \left(\frac{c(\pi_i)}{e \cdot v_i} \right) + \sum_{i=1}^{n} a_i \log \frac{c(u)}{\sum_{i=1}^{n} a_i} && (6.9.1) \\
&\le -\sum_{i=1}^{n} a_i \log \left(\frac{\gamma}{e} \cdot \frac{c(u)}{\sum_i a_i} \right) + \sum_{i=1}^{n} a_i \log \frac{c(u)}{\sum_{i=1}^{n} a_i} && (6.9.2) \\
&= -\sum_{i=1}^{n} a_i \log \left(\frac{\gamma}{e} \right) \\
&= -f_{\min} \cdot \log \left(\frac{\gamma}{e} \right).
\end{aligned}$$

Step (6.9.1) follows because, by concavity of the function $h(x) = x \log x$, we have $h(a + b) - h(a) \le b \cdot h'(a)$. Step (6.9.2) follows because u moving to position p is a γ-move, hence $\sum_j a_j / c(u) \ge \gamma \cdot v_i / c(\pi_i)$. □

We now show the weaker recourse bound. By **(I)**, the potential Φ_h increases by at most

$$g_t(\mathcal{N}) \cdot \log \left(\frac{c_{\max}}{f_{\min}} \right)$$

with the addition of function g_t to the active set. By **(IV)**, it decreases by $\Omega(f_{\min})$ with every move that costs recourse 1, and otherwise does not increase. Since we scaled costs by $1/c_{\min}$ and coverages by $1/(e \cdot f_{\max})$, this implies a recourse bound of:

$$O\left(\frac{\sum_t g_t(\mathcal{N})}{f_{\min}} \log \left(\frac{c_{\max}}{c_{\min}} \cdot \frac{f_{\max}}{f_{\min}} \right) \right).$$

Part III

Streaming Algorithms

Chapter 7

Streaming Submodular Matching

7.1 Introduction

In this chapter we turn to the study of submodular maximization subject to combinatorial constraints, in particular streaming maximum submodular matching (MSM) and related problems. That is, we study the problem of computing high-value matchings where the value is determined by a submodular function, i.e. a function which captures *diminishing returns*.

Submodular function maximization has a long history. For example, it has been known since the 70s that the greedy algorithm yields an $e/(e-1) \approx 1.582$ approximation for monotone submodular maximization subject to a cardinality constraint [NWF78]. This is optimal among polytime algorithms with value oracle access [NW78], or assuming standard complexity-theoretic conjectures [Fei98, DS14, Man20]. The same problem for *non-monotone* submodular functions is harder; it is hard to approximate to within a 2.037 factor [GV11]. Much work has been dedicated to improving the achievable approximation [LMNS10, Von13, GV11, FNS11, BFNS14, EN16, BF19, GRST10]; the best currently stands at 2.597 [BF19].

Closer to our work is the study of submodular maximization subject to *matching* constraints. For this problem, the greedy algorithm has long been known to be 3-approximate for monotone functions [FNW78]. Improved approximations have since been obtained [LSV10, LMNS09, GRST10, FNSW11], with the current best being $(2+\epsilon)$ and $(4+\epsilon)$ for monotone and non-monotone MSM respectively [FNSW11]. The papers above studied rich families of constraints (e.g. matroid intersection, matchoids, exchange systems), some of which were motivated explicitly by matching constraints (see [FNSW11]). Beyond theoretical interest, the MSM problem also has great practical appeal, since many natural objectives exhibit diminishing returns behavior. Applications across different fields include: machine translation [LB11], Internet advertising [DSSX19, KMZ18], combinatorial auctions more broadly [Von08, LLN06, CCPV11], and any matching problem where the goal is a submodular notion of utility such as diversity [AGHI09, ADF17].

The proliferation of big-data applications such as those mentioned above has spurred a surge of interest in algorithms for the regime where the input is too large to even store in local memory. To this end, it is common to formulate problems in the *streaming* model. Here the input is presented element-by-element to an algorithm that is restricted to use $\tilde{O}(S)$ memory, where S is the maximum size of any feasible solution. We study MSM in this model.

For our problem when the objective is *linear*, a line of work [FKM+05, McG05, ELSW18, CS14, PS19, GW19] has shown that a $(2+\epsilon)$-approximation is possible in the streaming model [PS19, GW19]. Meanwhile, for submodular objectives under *cardinality* constraints (which are a special case of MSM in complete bipartite graphs), a separate line of work [BMKK14, KMZ+19, AEF+20, CGQ15, FKK18, MJK18] has culminated in the same $(2+\epsilon)$ approximation ratio, for both monotone and non-monotone functions (the latter taking exponential time, as is to be expected from the lower bound of [GV11]); moreover, this $(2+\epsilon)$ bound was recently proven to be tight [FNSZ20, NTM+18, AEF+20]. On the other hand, for fully general MSM, the gap between known upper and lower bounds remain frustratingly large. [CK15] gave a 7.75-approximate algorithm for MSM with monotone functions. For non-monotone functions, [CGQ15] gave a $(12+\epsilon)$-approximate algorithm, later improved by [FKK18] to $5+2\sqrt{6} \approx 9.899$. The only known lower bound for monotone MSM is $\frac{e}{e-1} \approx 1.582$ for streaming or polytime algorithms, implied respectively by [Kap13] and [Fei98, NW78]. For non-monotone functions, [GV11] implies a hardness of 2.037. Closing these gaps, especially from the algorithmic side, seems to require new ideas.

7.1.1 Results

We present a number of improved results for streaming maximum submodular matching (MSM) and related problems.

Our first result is an improvement on the 7.75 approximation of [CK15] for monotone MSM.

Theorem 7.1.1. *There exists a deterministic quasilinear-time streaming* MSM *algorithm for monotone functions which is* $3+2\sqrt{2} \approx 5.828$ *approximate.*

Our algorithm extends in various ways: First, it yields the same approximation ratio for submodular *b-matchings*, where each node v can be matched b_v times, improving on the previous best 8-approximations [CGQ15, FKK18]. For the special case of linear functions (MWM), our algorithm—with appropriate parameters—recovers the $(2+\epsilon)$-approximate algorithm of [PS19]. For weighted b-matching (MWBM), a slight modification of our algorithm yields a $(3+\epsilon)$-approximate algorithm, improving on the previous best $(4+\epsilon)$-approximation [CS14].[1]

Next, we improve on the $5+2\sqrt{6} \approx 9.899$ approximation of [FKK18] for non-monotone MSM.

Theorem 7.1.2. *There exists a randomized linear-time streaming* MSM *algorithm for non-monotone functions which is* $4+2\sqrt{3} \approx 7.464$ *approximate.*

Our non-monotone MSM algorithm's approximation ratio is better than the previous state-of-the-art 7.75-approximate *monotone* MSM algorithm [CK15]. Moreover, when applied to monotone functions, the algorithm of Theorem 7.1.2 yields the same approximation ratio as the deterministic algorithm of Theorem 7.1.1.

We turn to proving hardness for monotone MSM. As stated before, the previous best lower bounds for this problem were $\frac{e}{e-1} \approx 1.582$. These lower bounds applied to either space-bounded [Kap13] or time-bounded algorithms [NW78, Fei98]. We show that the problem becomes harder for algorithms which are both space bounded and time bounded. This answers an open problem

[1]Independently, Mohsen Ghaffari (private communication) obtained the same $(3+\epsilon)$-approximate for MWBM.

posed in the Bertinoro Workshop on Sublinear Algorithms 2014 [MSM], at least for time bounded algorithms.[2]

Theorem 7.1.3. *No polytime streaming* MSM *algorithm for monotone functions is better than* 1.914 *approximate.*

Finally, to demonstrate that our techniques have the potential for wider applicability, we also use them to provide an alternative and unified proof of the results of [CK15] and [FKK18] for MSM, in Section 7.9.

7.1.2 Techniques and Overview

Our starting point is the breakthrough result of [PS19] for a special case of our problem—maximum weight matching (MWM). They gave a $(2+\epsilon)$-approximate streaming algorithm by extending the local-ratio technique [BBF+01]. Subsequently, [GW19] simplified and slightly improved the analysis of [PS19], by re-interpreting their algorithm in terms of the primal-dual method.[3] The primal-dual method is ubiquitous in the context of approximating linear objectives. We show that this method is also useful in the context of streaming submodular optimization, where to the best of our knowledge, it has not yet been used. For our primal-dual analysis, we rely on the *concave-closure extension* for submodular functions which has a "configuration LP"-like formulation. In particular, using this extension, we find that a natural generalization of the MWM algorithm of [PS19] (described in Section 7.3) yields improved bounds for monotone MSM and its generalization to b-matchings. Our primal-dual analysis is robust in the sense that it allows for extensions and generalizations, as we now outline.

Our approach in a nutshell (Sections 7.3 and 7.4). Our approach is to keep monotone dual solutions (initially zero), and whenever an edge arrives, discard it if its dual constraint is already satisfied. Edges whose dual constraint is not satisfied are added to a stack S, and relevant dual variables are increased, so as to satisfy their dual constraint. Finally, we unwind the stack S, constructing a matching M greedily. The intuition here is that the latter edge in the stack incident on a common edge have higher marginal gain than earlier such edges in the stack. More formally, we show that this matching M has value at least some constant times the dual objective cost. Weak LP duality and the choice of LP imply that $f(M) \geq \frac{1}{\alpha} \cdot f(S \cup OPT)$ for some $\alpha > 1$, which implies our algorithm is α-approximate for monotone MSM.

Extension 1 (Section 7.5). Extending our approach, which gives $f(M) \geq \frac{1}{\alpha} \cdot f(S \cup OPT)$, to non-monotone functions f seems challenging, since for such functions $f(S \cup OPT)$ can be arbitrarily smaller than $f(OPT)$. To overcome this challenge, we note that our dual updates over-satisfy dual constraints of edges in S. We can therefore afford to randomly discard edges whose dual is not satisfied on arrival (and not add them to S), resulting in these edges' dual constraints holding *in expectation*. This allows us to argue, via a generalization of the randomized primal-dual method of [DJK13] (on which we elaborate in Section 7.2), that $\mathsf{E}[f(M)] \geq \frac{1}{\alpha} \cdot \mathsf{E}[f(S \cup OPT)]$. As S contains each element with probability at most some q, a classic lemma of [BFNS14] allows us to

[2]We note briefly that such a bound does not follow from space lower bounds for cardinality constrained submodular maximization [FNSZ20, NTM+18] (a special case of our setting, with a complete bipartite graph on n and k nodes), since a bound for that problem cannot be superlinear in n.

[3]An equivalence between the local-ratio and primal-dual methods was established in [BR05], though it does not capture the extension of the local ratio method in [PS19].

show that $\mathsf{E}[f(S \cup OPT)] \geq (1-q) \cdot f(OPT)$, from which we get our results for non-monotone MSM.[4] Given the wide success of the randomized-primal dual method of [DJK13] in recent years [HKT+18, HTWZ19, HPT+19, FHTZ20, HZ20, HTWZ20, GKKP17, TWZ20, HZZ20], we believe that our extension of this method in the context of submodular optimization will likely find other applications.

Extension 2 (Section 7.6). For maximum weight b-*matching* (MWBM), the dual updates when adding an edge to the stack are not high enough to satisfy this edge's dual constraint. However, since we do cover each edge *outside* the stack S, weak duality implies that a maximum-weight b-matching M in the stack S has value at least as high as $f(M) \geq \frac{1}{2+\epsilon} \cdot f(OPT \setminus S)$, and trivially at least as high as $f(M) \geq f(OPT \cap S)$. Combining these lower bounds on $f(M)$ imply our improved $(3+\epsilon)$ approximation ratio for MWBM. This general approach seems fairly general, and could find uses for other sub-additive objectives subject to downward-closed constraints.

Unifying Prior Work (Section 7.9). To demonstrate the usefulness of our primal-dual analysis, we also show that this (randomized) primal-dual approach gives an alternative, unified way to analyze the MSM algorithms of [CK15, FKK18].

Lower bound (Section 7.7). Our lower bound instance makes use of two sources of hardness: computational hardness under ETH ([Fei98, DS14]) and information-theoretic hardness resulting form the algorithm not knowing the contents or order of the stream in advance ([GKK12]). In particular, our proof embeds a submodular optimization problem (specifically, SETCOVER) in parts of the linear instance of [GKK12], and hence exploits the submodularity in the MSM objective. Interestingly, our lower bound of 1.914 is higher than any convex combination of the previous hardness results we make use of, both of which imply a lower bound no higher than $e/(e-1)$.

7.2 Preliminaries

The maximum submodular matching (MSM) problem is defined by a non-negative submodular function $f : 2^E \to \mathbb{R}_{\geq 0}$, where E is the edge-set of some n-node graph $G = (V, E)$, and feasible sets are matchings in G. The more general maximum submodular b-matching (MSBM) problem has as feasible sets subgraphs in which the degree of each vertex v does not exceed b_v, for some input vector $\vec{b}$. Our objective is to design algorithms with low approximation ratio $\alpha \geq 1$, that is algorithms producing solutions M such that $\mathbb{E}[f(M)] \geq \frac{1}{\alpha} \cdot f(OPT)$ for the smallest possible value of α, where OPT is an optimal solution.

For *streaming* MSM, edges of E are presented one at a time, and we are tasked with computing a matching in G at the end of the stream, using little memory. In particular, we give algorithms using space within the output-sensitive bound of $\tilde{O}(M_{\max})$, where $M_{\max}$ is the maximum size of a b-matching (compared to the entire graph size, which can be $\Omega(n^2)$). We note that for matchings, this is a finer requirement than the usual $\tilde{O}(n)$ space bound, since $M_{\max} = O(n)$. We note also that for monotone objectives, an optimal b-matching is maximal, and so $M_{\max} \leq 2|OPT|$.

[4]Incidentally, for monotone functions, for which $E[f(M)] \geq \frac{1}{\alpha} \cdot E[f(S \cup OPT)] \geq \frac{1}{\alpha} \cdot f(OPT)$, this algorithm is α approximate. This is somewhat surprising, as this algorithm runs an α-approximate monotone algorithm (and this analysis is tight, by Section 7.10) on a random q-fraction of the input, suggesting an α/q approximation. Nonetheless, we show that for q not too small in terms of α, we retain the same approximation ratio even after this sub-sampling.

Consequently, such space is a trivial lower bound for outputting constant-approximate solutions.

On a technical note, we will only allow the algorithm to query the value oracle for f on subsets currently stored in memory. We assume the range of f is polynomially bounded. More precisely, we assume that $f_{\max}/f_{\min} = n^{O(1)}$ (these max and min marginals are defined below). This implies in particular that we can store a value $f_S(e)$ using $O(\log n)$ bits.

Useful Notation Throughout this chapter we will rely on the following notation. First, we denote by $e^{(1)}, e^{(2)}, \ldots,$ the edges in the stream, in order. For edges $e = e^{(i)}, e' = e^{(j)}$, we write $e < e'$ if and only if $i < j$, i.e., if e arrived before e'. Similarly to [CGQ15, FKK18], we will also use $f(e : S) := f_{S \cap \{e' < e\}}(e)$ as shorthand for the marginal gain from adding e to the set of elements which arrived before e in S. One simple yet useful property of this notation is that $\sum_{e \in S} f(e : S) = f(S)$ ([CGQ15, Lemma 1].) Other properties of this notation we will make use of, both easy consequences of submodularity, are $f(e : S) \leq f_S(e)$, as well as monotonicity of $f(e : S)$ in S, i.e., $f(e : A) \geq f(e : B)$ for $A \subseteq B$. Finally, for any edge e, we denote by $N(e) := \{e' \mid e' \cap e \neq \emptyset\}$ the set of neighboring edges of e.

7.2.1 The Primal-Dual Method in Our Setting

As discussed in Section 7.1.2, the main workhorse of our algorithms is the primal-dual method. In this method, we consider some linear program (LP) relaxation, and its dual LP. We then design an algorithm which computes a (primal) solution of value P, and a feasible solution of value D, and show that $P \geq \frac{1}{\alpha} \cdot D$, which implies an approximation ratio of α, by weak duality.

For linear objectives, the first step of the primal-dual method—obtaining an LP relaxation—is often direct: write some integer linear program for the problem and drop the integrality constraints. For submodular objective functions the first step of defining a relaxation usually requires extending f to real vectors. For this, we use the *concave closure* (recall the definition from Section 2.2). To define an LP relaxation for submodular maximization of some function g subject to some linear constraints $A\vec{x} \leq \vec{c}$, we simply consider $\max \left\{ g^+(\vec{x}) \;\middle|\; A\vec{x} \leq \vec{c} \right\}$. For MSBM, we obtain the primal and dual programs given below.

Primal(P)

$$\begin{array}{rl} \max & \displaystyle\sum_{T \subseteq E} \alpha_T \cdot g(T) \\ \text{subject to} & \\ \forall T \subseteq E: & \displaystyle\sum_{T \ni e} \alpha_T = x_e \\ & \displaystyle\sum_{T \subseteq E} \alpha_T = 1 \\ \forall v \in V: & \displaystyle\sum_{e \ni v} x_e \leq b_v \\ \forall e \in E, T \subseteq E: & x_e, \alpha_T \geq 0 \end{array}$$

Dual(D)

$$\begin{array}{rl} \min & \displaystyle\mu + \sum_{v \in V} b_v \cdot \phi_v \\ \text{subject to} & \\ \forall T \subseteq E: & \displaystyle\mu + \sum_{e \in T} \lambda_e \geq g(T) \\ \forall e \in E: & \displaystyle\sum_{v \in e} \phi_v \geq \lambda_e \\ \forall v \in V: & \phi_v \geq 0 \end{array}$$

7.2.2 Non-Monotone MSM: the Randomized Primal-Dual Method

To go from monotone to non-monotone function maximization, we make use of our dual updates resulting in dual solutions which over-satisfy (some) dual constraints. This allows us to randomly sub-sample edges with probability q when deciding whether to insert them into S, and still have a dual solution which is feasible *in expectation* over the choice of S. This is akin to the randomized primal-dual method of [DJK13], who introduced this technique in the context of maximum *cardinality* and *weighted* matching. However, unlike in [DJK13] (and subsequent work [HKT+18, HTWZ19, HPT+19, FHTZ20, HZ20, HTWZ20, GKKP17, TWZ20, HZZ20]), for our problem the LP is not fixed. Specifically, we consider a different submodular function in our LP based on S, denoted by $g^S(T) := f(T \cup S)$. This results in *random* primal and dual LPs, depending on the random set S. We show that our (randomized) dual solution is feasible for the obtained (randomized) dual LP in expectation over S. Consequently, our expected solution's value is at least as high as some multiple of an expected solution to the dual LP, implying

$$\mathbb{E}_S[f(M)] \geq \frac{1}{\alpha} \cdot \mathbb{E}_S[D] \geq \frac{1}{\alpha} \cdot \mathbb{E}_S[f(S \cup OPT)]. \qquad (7.2.1)$$

Equation (7.2.1) retrieves our bound for monotone functions, for which $\mathbb{E}_S[f(S \cup OPT)] \geq f(OPT)$. To obtain bounds for non-monotone functions, we show that $\mathbb{E}_S[f(S \cup OPT)] \geq (1-q) \cdot f(OPT)$, by relying on the following lemma, due to [BFNS14, Lemma 2.2].

Lemma 7.2.1 ([BFNS14]). *Let $h : 2^N \to R_{\geq 0}$ be a non-negative submodular function, and let B be a random subset of N containing every element of N with probability at most q (not necessarily independently), then $\mathbb{E}[h(B)] \geq (1-q) \cdot h(\emptyset)$.*

7.3 Our Basic Algorithm

In this section we describe our monotone submodular b-matching algorithm, which we will use with slight modifications and different parameter choices in coming sections.

The algorithm maintains a stack of edges S, initially empty, as well as vertex potentials $\vec{\phi} \in \mathbb{R}^{|V|}$. To avoid storing zero-valued potentials (increasing space to $\tilde{O}(n)$, rather than $\tilde{O}(M_{\max})$), we implicitly represent these by keeping a mapping from vertices with non-zero potentials to their potentials, by using e.g., a hash table or balanced binary search tree. When an edge e arrives, we compare the marginal value of this arriving edge with respect to the stack to the sum of vertex potentials of the edge's endpoints times a slack parameter C. If $C \cdot \sum_{v \in e} \phi_v$ is larger, we continue to the next edge. Otherwise, with probability q we add the edge to the stack and increment the endpoint vertex potentials. At the end of the stream, we construct a b-matching greedily by unwinding the stack *in reverse order*. The pseudocode is given in Algorithm 14.

Algorithm 14 clearly outputs a feasible b-matching. In subsequent sections we analyze this algorithm for various values of the parameters C and q. Before doing so, we note that this algorithm when run with $C = 1 + \Omega(1)$ is indeed a streaming algorithm, i.e., its space usage is as follows.

Lemma 7.3.1. *For any constant $\epsilon > 0$, Algorithm 14 run with $C = 1 + \epsilon$ uses $\tilde{O}(M_{\max})$ space.*

Algorithm 14 The MSbM Algorithm

```
Initialization
1:  $S \leftarrow$ emptystack
2:  $\forall v \in V : \phi_v^{(0)} \leftarrow 0$ (implicitly)
In Stream
3:  for $t \in \{1, \ldots, |E|\}$ do
4:      $e \leftarrow e^{(t)}$, $t^{th}$ edge in stream.
5:      for each $v \in V$, set $\phi_v^{(t)} \leftarrow \phi_v^{(t-1)}$.
6:      if $C \cdot \sum_{v \in e} \phi_v^{(t-1)} \geq f(e : S)$ then
7:          Skip edge $e$ and continue.
8:      else
9:          with probability $q$ do
10:             $S.push(e)$
11:             for $v \in e$ do
12:                 $w_{ev} \leftarrow \frac{f(e:S) - \sum_{v \in e} \phi_v^{(t-1)}}{b_v}$.
13:             for $v \in e$ do
14:                 $\phi_v^{(t)} \leftarrow \phi_v^{(t-1)} + w_{ev}$.
Post-Processing
15: $M \leftarrow \emptyset$
16: while $S \neq$ emptystack do
17:     $e \leftarrow S.pop()$
18:     if $|M \cap N(e)| < b_v$ for all $v \in e$ then
19:         $M \leftarrow M \cup \{e\}$
20: return $M$
```

The proof relies on each vertex v only having $\tilde{O}(b_v)$ edges in the stack, since every edge incident on vertex v inserted to the stack increases ϕ_v by a multiplicative factor of $(C-1)/b_v$, while the minimum and maximum non-zero values which ϕ_v can take are polynomially bounded in each other, due to f being polynomially bounded. Since this bound holds in particular for vertices of a minimum vertex cover, the bound follows due to minimum vertex covers having size at most twice that of maximal b-matchings in the same graph. See Section 7.11 for the complete proof.

We further note that Algorithm 14 runs in time (quasi)linear in $|E|$, times evaluation time of f.

Lemma 7.3.2. *A randomized (deterministic) implementation of Algorithm 14 requires $O(1)$ ($O(\log n)$) operations and $O(1)$ function evaluations per arrival, followed by $\tilde{O}(M_{\max})$ time post-processing. Using $\tilde{O}(n)$ space, the deterministic time per arrival can be decreased to $O(1)$, by storing all ϕ_v.*

7.4 Monotone MSbM

In this section we will consider a *deterministic* instantiation of Algorithm 14 (specifically, we will set $q = 1$) in the context of monotone submodular b-matching.

To argue about the approximation ratio, we will fit a dual solution to this algorithm. Define the auxiliary submodular functions $g^S : 2^E \to \mathsf{R}^+$ to be $g^S(T) := f(S \cup T)$. We will work with the dual LP (D) for the function g^S, and consider the following dual solution.

$$\mu := f(S) = g^S(\emptyset),$$
$$\phi_v := C \cdot \phi_v^{(|E|)}$$

$$\lambda_e := \begin{cases} f(e:S) & e \notin S \\ 0 & e \in S. \end{cases}$$

We start by showing that the above is indeed dual feasible.

Lemma 7.4.1. *The dual solution* $(\mu, \vec{\phi}, \vec{\lambda})$ *is feasible for the LP* (D) *with function* g^S.

Proof. To see that the first set of constraints is satisfied, note that by submodularity of f

$$\sum_{e \in T} \lambda_e = \sum_{e \in T \setminus S} f(e:S) \geq \sum_{e \in T \setminus S} f_S(e) \geq f_S(T \setminus S) = f(S \cup T) - f(S) = g^S(T) - \mu.$$

For the second set of constraints, note that an edge $e = e^{(t)}$ is not added to the stack if and only if the check at Line 6 fails. Therefore, since $\phi_v^{(t)}$ values increase monotonically with t, we have

$$\sum_{v \in e} \phi_v = C \cdot \sum_{v \in e} \phi_v^{(|E|)} \geq C \cdot \sum_{v \in e} \phi_v^{(t-1)} \geq f(e:S) = \lambda_e. \qquad \square$$

It remains to relate the value of the solution M to the cost of this dual. We first prove an auxiliary relationship that will be useful:

Lemma 7.4.2. *The* b*-matching* M *output by Algorithm 14 satisfies*

$$f(M) \geq \frac{1}{2} \cdot \sum_{e \in S} \sum_{v \in e} b_v \cdot w_{ev}.$$

Proof. We first note that for any edge $e = e^{(t)}$ and $v \in e$, since $\phi_v^{(t-1)} = \sum_{e' \ni v, e' < e} w_e$, we have that

$$f(e:S) = b_v \cdot w_{ev} + \sum_{u \in e} \phi_u^{(t-1)} \geq b_v \cdot w_{ev} + \phi_v^{(t-1)} = b_v \cdot w_{ev} + \sum_{\substack{e' \ni v \\ e' < e}} w_{e'v}.$$

Combined with submodularity of f, the above yields the following lower bound on $f(M)$,

$$f(M) = \sum_{e \in M} f(e:M) \geq \sum_{e \in M} f(e:S) \geq \sum_{e \in M} \sum_{v \in e} \Big(b_v \cdot w_{ev} + \sum_{\substack{e' \ni v \\ e' < e}} w_{e'v} \Big).$$

On the other hand, the greedy manner in which we construct M implies that any edge $e' \in S \setminus M$ must have at least one endpoint v with b_v edges $e > e'$ in M. Consequently, the term $w_{e'v}$ for such e and v is summed b_v times in the above lower bound for $f(M)$. On the other hand, $b_v \cdot w_{ev} = b_u \cdot w_{eu}$ for $e = (u, v)$, by definition. From the above we obtain our desired inequality.

$$f(M) \geq \sum_{e \in M} \sum_{v \in e} b_v \cdot w_{ev} + \frac{1}{2} \cdot \sum_{e \in S \setminus M} \sum_{v \in e} b_v \cdot w_{ev} \geq \frac{1}{2} \cdot \sum_{e \in S} \sum_{v \in e} b_v \cdot w_{ev}. \qquad \square$$

We can now bound the two terms in the dual objective separately with respect to the primal, using the following two corollaries of Lemma 7.4.2.

Lemma 7.4.3. *The* b*-matching* M *output by Algorithm 14 satisfies* $f(M) \geq \frac{1}{2C} \sum_{v \in V} b_v \cdot \phi_v$.

Proof. Since $\phi_v = C \cdot \phi_v^{(|E|)}$, and $w_{ev} = \phi_v^{(t)} - \phi_v^{(t-1)}$ for all $v \in e = e^{(t)}$, Lemma 7.4.2 implies that

$$f(M) \geq \frac{1}{2} \cdot \sum_{e \in S} \sum_{v \in e} b_v \cdot w_{ev} = \frac{1}{2} \cdot \sum_{v \in V} \sum_{t=1}^{|E|} b_v \cdot \left(\phi_v^{(t)} - \phi_v^{(t-1)}\right) = \frac{1}{2} \cdot \sum_{v \in V} b_v \cdot \phi_v^{(|E|)} = \frac{1}{2C} \cdot \sum_{v \in V} b_v \cdot \phi_v.$$

□

Lemma 7.4.4. *The b-matching M output by Algorithm 14 satisfies $f(M) \geq \left(1 - \frac{1}{C}\right)\mu$.*

Proof. We note that $w_e > 0$ for an edge $e = e^{(t)}$ if and only if $f(e : S) \geq C \cdot \sum_{v \in e} \phi_v^{(t-1)}$. Hence,

$$b_v \cdot w_{ev} = f(e : S) - \sum_{v \in e} \phi_v^{(t-1)} \geq \left(1 - \frac{1}{C}\right) \cdot f(e : S).$$

Combining the above with Lemma 7.4.2, and again recalling that for $e = (u, v)$, we have that $b_v \cdot w_{ev} = b_u \cdot w_{eu}$, by definition, we obtain the desired inequality.

$$f(M) \geq \frac{1}{2} \cdot \sum_{e \in S} \sum_{v \in e} b_v \cdot w_{ev} \geq \left(1 - \frac{1}{C}\right) \sum_{e \in S} f(e : S) = \left(1 - \frac{1}{C}\right) f(S).$$

□

Combining the above two corollaries and Lemma 7.4.1 with LP duality, we can now analyze the algorithm's approximation ratio.

Theorem 7.4.5. *Algorithm 14 run with $q = 1$ and C on a monotone* MSBM *instance outputs a b-matching M of value*

$$\left(2C + \frac{C}{C-1}\right) \cdot f(M) \geq f(\textsc{Opt}).$$

This is optimized by taking $C = 1 + \frac{1}{\sqrt{2}}$, which yields a $3 + 2\sqrt{2} \approx 5.828$ approximation.

Proof. By weak LP duality and Lemma 7.4.1, together with monotonicity of f, we have that

$$C \cdot \sum_v b_v \cdot \phi_v + \mu \geq \max_T g^S(T) = \max_T f(S \cup T) \geq f(S \cup \textsc{Opt}) \geq f(\textsc{Opt}).$$

Combining Lemma 7.4.3 and Lemma 7.4.4 and rearranging, we get the desired inequality,

$$\left(2C + \frac{C}{C-1}\right) \cdot f(M) \geq C \cdot \sum_v b_v \cdot \phi_v + \mu \geq f(\textsc{Opt}).$$

□

In Section 7.10 we show that our analysis of Algorithm 14 is tight.

We note that our analysis of this section required monotonicity, as we lower bounded $f(M)$ by (a multiple of) $f(S \cup OPT) \geq f(OPT)$, where the last step crucially relies on monotonicity. In the next section, we show how the use of randomness (namely, setting $q \neq 1$) allows us to obtain new results for *non-monotone* MSM.

7.5 Non-Monotone MSM

In this section we consider MSM (so, $b_v = 1$ for all v in this section), for *non-monotone* functions.

To extend our results to non-monotone MSM, we make use of the freedom to choose $q \notin \{0, 1\}$, resulting in a randomized algorithm. This will allow us to lower bound $\mathsf{E}_S[f(S \cup OPT)]$ in terms of $f(OPT)$. But first, we show that for appropriately chosen q, the output matching M has high value compared to $\mathsf{E}_S[f(S \cup OPT)]$. The analysis of this fact will follow the same outline of Section 7.4, relying on LP duality, but with a twist.

For our dual fitting, we use the same dual solution as in Section 7.4. However, this time this dual solution will only be feasible *in expectation*, in the following sense. Since we now have $q \notin \{0, 1\}$, Algorithm 14 is now a randomized algorithm, S is a random set, g^S is a random submodular function, and thus (D) is a random LP. Let $\mathbb{E}[(D)]$ denote this LP, which is (D) with the submodular function $g(T) := \mathbb{E}_S[g^S(T)]$. We now show that our dual solution's expectation is feasible for $\mathbb{E}[(D)]$.

Lemma 7.5.1. *For $q \in [1/(2C+1), 1/2]$, the expected dual solution $(\mathbb{E}[\mu], \mathbb{E}[\vec{\phi}], \mathbb{E}[\vec{\lambda}])$ is feasible for the expected LP $\mathbb{E}[(D)]$.*

Proof. The first set of constraints is satisfied for any realization of the randomness. Indeed, as in the proof of Lemma 7.4.1, for any realization of S, by submodularity of f, we have

$$\sum_{e \in T} \lambda_e = \sum_{e \in T \setminus S} f(e : S) \geq \sum_{e \in T \setminus S} f_S(e) \geq f_S(T \setminus S) = f(S \cup T) - f(S) = g^S(T) - \mu.$$

Consequently, taking expectation over S, we have that indeed, $\mathsf{E}_S[\mu] + \sum_{e \in T} \mathsf{E}_S[\lambda_e] \geq \mathsf{E}_S[g^S(T)]$. We now tun to proving the second set of constraints, which will only hold in expectation.

Fix an edge $e = e^{(t)}$, and define the event $A_e := [f(e : S) \leq C \cdot \sum_{v \in V} \phi_v^{(t-1)}]$. Then, by definition of A_e and monotonicity of $\phi_v^{(t)}$ in t, we have that

$$\mathbb{E}\left[\sum_{v \in e} \phi_v \,\middle|\, A_e\right] \geq \mathbb{E}\left[C \cdot \sum_{v \in e} \phi_v^{(t-1)} \,\middle|\, A_e\right] \geq \mathbb{E}[f(e : S) \mid A_e] = \mathbb{E}[\lambda_e \mid A_e]. \tag{7.5.1}$$

We now prove the same inequality holds when conditioning on the complement, $\overline{A_e}$.

Fix a realization of the randomness R for which $\overline{A_e}$ holds. Then, $e = e^{(t)}$ fails the test in Line 6, and so with probability q, we have $\sum_{v \in e} \phi_v^{(t)} = \sum_{v \in e}(\phi_v^{(t-1)} + w_e) = 2 \cdot f(e : S) - \sum_{v \in e} \phi_v^{(t-1)}$, and with probability $(1 - q)$, we have $\sum_{v \in e} \phi_v^{(t)} = \sum_{v \in e} \phi_v^{(t-1)}$. Hence, in this case, as $q \leq \frac{1}{2}$, we have

$$\mathbb{E}\left[\sum_{v \in e} \phi_v^{(t)} \,\middle|\, R\right] = 2q \cdot f(e : S) + (1 - 2q) \cdot \sum_{v \in e} \phi_v^{(t-1)} \geq 2q \cdot f(e : S).$$

Now, since $\phi_v \geq C \cdot \phi^{(t)}$, and $q \geq 1/(2C+1)$ and since λ_e is set to $f(e : S)$ if e is not added to S (with probability $1 - q$) and set to zero otherwise, the above implies that

$$\mathbb{E}\left[\sum_{v \in e} \phi_v \,\middle|\, R\right] \geq 2qC \cdot f(e : S) \geq (1 - q) \cdot f(e : S) = \mathbb{E}[\lambda_e \mid R].$$

By the law of total expectation, taken over all $R \subseteq \overline{A_e}$, we have

$$\mathbb{E}\left[\sum_{v\in e} \phi_v \,\middle|\, \overline{A_e}\right] \geq \mathbb{E}\left[\lambda_e \mid \overline{A_e}\right]. \tag{7.5.2}$$

Combining inequalities (7.5.1) and (7.5.2) with the law of total expectation gives the desired inequality,

$$\mathbb{E}\left[\sum_{v\in e} \phi_v\right] \geq \mathbb{E}[\lambda_e]. \qquad \square$$

To bound the performance of this section's randomized variant of Algorithm 14, we can reuse corollaries 7.4.3 and 7.4.4, since these follow from Lemma 7.4.1, which holds for every realization of the random choices of the algorithm. We now use these corollaries, LP duality and Lemma 7.5.1, together with Lemma 7.2.1, to analyze this algorithm.

Theorem 7.5.2. *Algorithm 14 run with $q = 1/(2C+1)$ and C on a non-monotone* MSM *instance outputs a matching M of value*

$$\left(\frac{4C^2-1}{2C-2}\right)\cdot f(M) \geq f(\textsc{Opt}).$$

This is optimized by taking $C = 1+\frac{\sqrt{3}}{2}$, resulting in an approximation ratio of $4+2\sqrt{3} \approx 7.464$. *Moreover, the same algorithm is $2C + C/(C-1)$ approximate for* monotone MSM.

Proof. First, by Lemma 7.4.3 and Lemma 7.4.4, for every realization of the algorithm, we have

$$\left(2C + \frac{C}{C-1}\right)\cdot f(M) \geq \sum_v \phi_v + \mu,$$

and thus this relationship holds in expectation as well.

$$\left(2C + \frac{C}{C-1}\right)\cdot \mathsf{E}[f(M)] \geq \mathbb{E}\left[\sum_v \phi_v + \mu\right]. \tag{7.5.3}$$

On the other hand, by Lemma 7.5.1, the expected dual LP solution is feasible for $\mathbb{E}[(D)]$. Therefore, by weak LP duality, we have

$$\mathbb{E}\left[\sum_v \phi_v + \mu\right] \geq \max_T \mathbb{E}[g(T)] = \max_T \mathbb{E}[f(S\cup T)] \geq \mathbb{E}[f(S\cup \textsc{Opt})]. \tag{7.5.4}$$

The result for monotone MSM follows from equations (7.5.3) and (7.5.4), together with monotonicity implying $\mathbb{E}[f(S\cup\textsc{Opt})] \geq f(\textsc{Opt})$.

For non-monotone MSM, let us define the additional auxiliary function $h : 2^E \to \mathsf{R}^+$, with $h(T) := f(\textsc{Opt}\cup T)$. Now note that by our sampling procedure, S is a random subset of E containing every edge with probability at most q. Hence, by Lemma 7.2.1, we have

$$\mathbb{E}[f(S\cup\textsc{Opt})] = \mathbb{E}[h(S)] \geq (1-q)\cdot h(\emptyset) = (1-q)\cdot f(\textsc{Opt}). \tag{7.5.5}$$

Combining equations (7.5.3), (7.5.4) and (7.5.5), together with our choice of $q = 1/(2C+1)$, the desired inequality follows by rearranging terms. $\square$

Having explored the use of Algorithm 14 for submodular matchings, we now turn to analyzing this algorithm in the context of streaming *linear* objectives.

7.6 Linear Objectives

In this section we address the use of Algorithm 14 to matching and b-matching with linear objectives, i.e., MWM and MWBM, using a deterministic variant, with $q = 1$.

For MWM, this algorithm with $C = 1+\epsilon$ is essentially the algorithm of [PS19], and so it retrieves the state-of-the-art $(2+\epsilon)$-approximation for this problem, previously analyzed in [PS19, GW19]. We therefore focus on MWBM, for which a simple modification of Algorithm 14 yields a $3+\epsilon$ approximation, improving upon the previous best $4+\epsilon$ approximation due to [CS14].

The modification to Algorithm 14 which we consider is a natural one: instead of computing M greedily, we simply compute an optimal MWBM M in the subgraph induced by S, using a polytime linear-space offline algorithm (e.g., [Ans87, Gab18]). Trivially, the b-matching M has weight at least

$$w(M) \geq w(OPT \cap S). \tag{7.6.1}$$

Moreover, this b-matching has weight no lower than the greedily-constructed b-matching of lines 15-20. We use LP duality to show that this modified algorithm with $C = 1+\epsilon$ outputs a b-matching M of weight at least $w(M) \geq \frac{1}{2+\epsilon} \cdot w(OPT \setminus S)$.

Lemma 7.6.1. *Let M be a* MWBM *in the stack S obtained by running Algorithm 14 with $C = 1+\epsilon/2$ and $q = 1$ until Line 15. Then, we have $w(M) \geq \frac{1}{2+\epsilon} \cdot w(OPT \setminus S)$.*

Proof. Consider the matching M' obtained by greedily unwinding the stack, as in Algorithm 14. Clearly, $w(M) \geq w(M')$. So, by Lemma 7.4.3, we have $w(M) \geq \frac{1}{2+\epsilon} \cdot \sum_{v \in V} \phi_v$, for $\phi_v = C \cdot \phi_v^{(|E|)}$. To relate $\sum_{v\in V} \phi_v$ to $w(OPT)$, we show that the dual solution $(0, \vec{\phi}, \vec{w})$ is dual feasible for the LP (D) with function w.

The first set of constraints are trivially satisfied, due to linearity of w, as $0 + \sum_{e\in T} w_e = w(T)$.

For the second set of constraints, note that an edge $e = e^{(t)}$ is not added to the stack if and only if the check at Line 6 fails. Therefore, since $\phi_v^{(t)}$ values increase monotonically with t, we have

$$\sum_{v\in e} \phi_v = C \cdot \sum_{v \in e} \phi_v^{(|E|)} \geq C \cdot \sum_{v\in e} \phi_v^{(t-1)} \geq f(e:S) = w_e.$$

Therefore, by weak LP duality, we have $w(M) \geq 0 + \frac{1}{2+\epsilon} \cdot \sum_{v\in V} \phi_v \geq \frac{1}{2+\epsilon} \cdot w(OPT)$. □

We are now ready to analyze the approximation ratio of this MWBM algorithm.

Theorem 7.6.2. *For any $\epsilon \geq 0$, Algorithm 14 run with $C = 1+\epsilon/2$ and $q = 1$ until Line 15, followed by a linear-space offline* MWBM *algorithm run on S to compute a solution M is a $(3+\epsilon)$-approximate streaming* MWBM *algorithm.*

Proof. To see that this is a streaming algorithm, we recall that $|S| = \tilde{O}(\sum_v b_v)$, by Lemma 7.3.1. Since we compute M by running an offline linear-space algorithm on the subgraph induced by S, therefore using $O(|S|)$ space for this last step, the desired space bound follows.

To analyze the algorithm's approximation ratio, let $\alpha \in [0,1]$ be the weighted fraction of OPT in S. That is, $w(OPT \cap S) = \alpha \cdot w(OPT)$, and by linearity, $w(OPT \setminus S) = (1-\alpha)\cdot w(OPT)$. Therefore, by Equation (7.6.1) and Lemma 7.6.1 we have the following.

$$w(M) \geq w(OPT \cap S) = \alpha \cdot w(OPT).$$

$$w(M) \geq \frac{1}{2+\epsilon} \cdot w(OPT \setminus S) = \frac{1-\alpha}{2+\epsilon} \cdot w(OPT).$$

We thus find that the approximation ratio of this algorithm is at most $1/\min\{\alpha, \frac{1-\alpha}{2+\epsilon}\} \leq 3+\epsilon$. $\square$

Remark. We note that this approach—dual covering constraints for elements outside of the algorithm's memory S, and solving the problem optimally for S—is rather general. In particular, it applies to matching under any sub-additive (not just submodular) set function f, for which $f(OPT) \leq f(OPT \setminus S) + f(OPT \cap S)$. Moreover, this approach extends beyond matchings, to any downward-closed constraints, for which $OPT \setminus S$ and $OPT \cap S$ are both feasible solutions. So, it seems like this approach could find applications to streaming algorithms for other objectives and constraints, provided dual feasibility can be guaranteed using a dual solution of value bounded by that of the output solution.

7.7 Lower Bound for MSM

Previous work shows that beating a $\frac{e}{e-1} \approx 1.582$ approximation for MSM in the streaming model is impossible for quasilinear space bounded algorithms [Kap13], or polytime bounded algorithms [Fei98, DS14, Man20]. In this section, we show that assuming the exponential time hypothesis (ETH), whereby $\mathsf{NP} \not\subseteq \mathsf{TIME}(2^{o(n)})$ [IP01], beating 1.914 is impossible for any algorithm that is both space and time bounded. In particular, we will rely on seminal hardness of approximation results SetCover from [DS14]. Recall:

Definition 7.7.1. *A* SetCover *instance consists of a set system $(\mathcal{U}, \mathcal{S})$, with $\mathcal{S} \subseteq 2^{\mathcal{U}}$. The goal is to pick the smallest number k of sets $S_1, \ldots, S_k \in \mathcal{S}$ such that $\left|\bigcup_{i \in [k]} S_i\right| = |\mathcal{U}|$. We use K to denote the size of the minimal cover for the instance $(\mathcal{U}, \mathcal{S})$, and $N = |\mathcal{U}| + |\mathcal{S}|$ to denote the description size.*

Lemma 7.7.2 (Extension of Corollary 1.6 of [DS14])**.** *Assuming ETH, every algorithm achieving an approximation ratio $(1-\alpha)\ln|\mathcal{U}|$ for* Set Cover *runs in time strictly greater than $2^{N^{\gamma \cdot \alpha}}$ for some $\gamma > 0$. Furthermore, this holds even under the assumptions that $|\mathcal{S}| \leq K^{1/(\gamma\alpha)}$ and $|\mathcal{U}| \leq |\mathcal{S}|^{1/(\gamma\alpha)}$.*

See Section 7.12 for a proof that the hardness holds even under the extra assumptions. To describe the instance, we will also use some extremal graph theory results from [GKK12].

Definition 7.7.3. *An α-Ruzsa-Szemerédi graph (α-RS graph) is a bipartite graph $G = (P, Q, E)$ with $|P| = |Q| = n$ that is a union of induced matchings of size exactly αn.*

Theorem 7.7.4 (Lemma 53 of [GKK12])**.** *For any constant $\epsilon > 0$, there exists a family of balanced bipartite $(1/2 - \epsilon)$-RS graphs with $n^{1+\Omega(1/\log\log n)}$ edges.*

In what follows we will show a randomized reduction from SetCover to streaming MSM. Specifically, we will show that if there is a polytime streaming algorithm for MSM achieving ratio better than 1.914, then there is an algorithm for SetCover violating Lemma 7.7.2. We proceed to describing our reduction.

The Reduction. The input is a SetCover instance $(\mathcal{U}, \mathcal{S})$ for which the minimal cover contains K sets. Fix $n = 2^{k^{1/d}}$ for a degree d to be determined later.

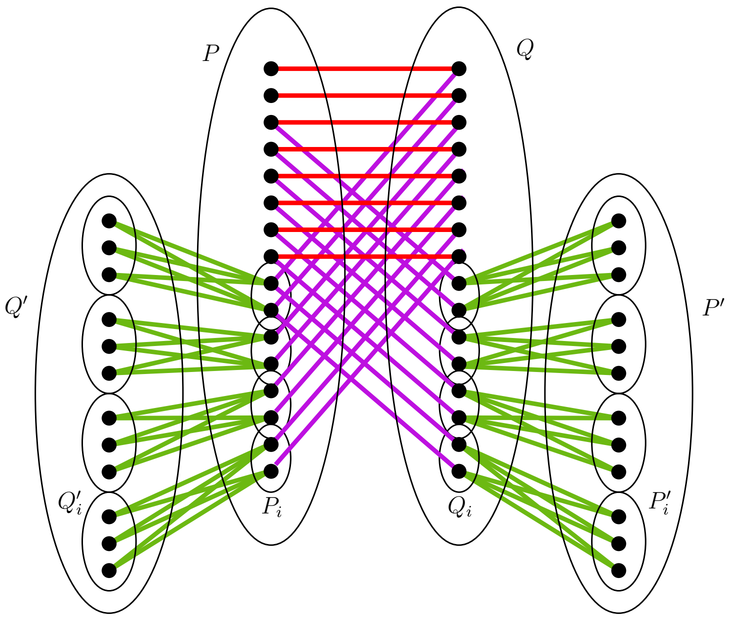


Figure 7.1: Illustration of lower bound instance. Red edges represent the edges of the distinguished matching M_r in E_1, purple edges represent other edges in E_1, green edges represent edges of E_2. The red and purple edges together form the $(1/2 - \epsilon)$-RS graph G_0, subsampled.

We create an underlying bipartite graph $G = (L, R, E)$ with $n \operatorname{poly}\log n$ vertices as follows. The left/right vertex sets are partitioned into $L = P \sqcup P'$, $R = Q \sqcup Q'$. We let $|P| = |Q| = 2n$, and we let $|P'| = |Q'| = n \cdot |\mathcal{S}|/K$.

The edge set E arrives in two phases. In phase 1, all the edge of a set E_1 arrive, in phase 2 the edges of E_2 arrive. To define E_1, let G_0 be a fixed $(1/2-\epsilon)$-RS graph with $m = \Omega\left(n^{1+1/\log\log n}\right)$ edges between P and Q, and let this graph be the union of the matchings $M_1, \ldots, M_t$. Let M'_i be a random subset of M_i of size $(1/2 - \delta)n$ for a parameter $\epsilon < \delta < 1/4$ and let $E_1 = M'_1 \cup \ldots \cup M'_t$. Choose one index $r \in [k]$ uniformly at random, and call M'_r the *distinguished matching*. Note that the index r is unknown to the algorithm.

Define E_2 as follows. Let $P_1 \sqcup P_2 \sqcup \ldots \sqcup P_{n/K}$ and $Q_1 \sqcup Q_2 \sqcup \ldots \sqcup Q_{n/K}$ be partitions of the the vertices of P and Q respectively not matched by M_r into subsets of size K. Similarly, let $P'_1 \sqcup P'_2 \sqcup \ldots \sqcup P'_{n/K}$ and $Q'_1 \sqcup Q'_2 \sqcup \ldots \sqcup Q'_{n/K}$ be partitions of P' and Q' into subsets of size $|\mathcal{S}|$. Let F_i be the edges of the complete bipartite graph between P_i and Q'_i, and let G_i be the edges of the complete bipartite graph between Q_i and P'_i. Finally, set $E_2 = \bigcup_i F_i \cup G_i$. See Figure 7.1.

It remains to describe the submodular function f. First, define the set function $f_1(E) = |E \cap E_1|$. Next, we define the function f_2 which is parametrized by the SETCOVER instance. We identify each set of vertices P'_i and Q'_i with a disjoint copy of $\mathcal{S}$. For every edge $e \in E_2$, let $\phi(e)$ denote the set with which the endpoint of e in $P' \cup Q'$ is associated. Now, for some parameter $\eta > 0$ to be determined later, we define

$$f_2(E) := \frac{\eta K}{|\mathcal{U}|} \cdot \left| \bigcup_{e \in E} \phi(e) \right|.$$

Finally, set $f := f_1 + f_2$. Note that f is submodular since it the sum of a scaled coverage function and a linear function. On a technical note, since we assume that $|\mathcal{U}|$ is polynomially bounded in $|\mathcal{S}|$, we can represent the values of this function with $\operatorname{poly}\log n$ bits.

Some intuition. Intuitively, we can imagine that all edges of E_1 are worth 1. We imagine that each edge of E_2 is a set in one of the copies of the instance $(\mathcal{S}, \mathcal{U})$, and we let the value of all edges selected in the second phase be the coverage of all the associated sets (scaled by $\eta K/|\mathcal{U}|$). First we we will argue that the algorithm can output almost none of the edges of M'_r, since it after phase 1 it has no information as to which matching is the distinguished one. Hence the majority of the edges it uses from phase 1 must be from $E_1 \backslash M'_r$. However, each edge the algorithm chooses from $E_1 \backslash M'_r$ precludes it from taking between 1 and 2 edges of E_2. Furthermore, maximizing the value of edges of E_2 amounts to solving a hard MAX K-COVERAGE instance. The coverage is scaled by the parameter η, and as a result, the algorithm is incentivized to take some $k := cK$ edges from each of the bipartite graphs (P_i, Q'_i) (and (Q_i, P'_i)) of E_2 and the remaining edges from $E_1 \backslash M'_r$. Meanwhile, OPT can take the distinguished matching edges M'_r as well as the edges of E_2 maximizing the coverage instance. Our bound will follow by setting η to maximize the ratio between these.

To start, we show that no streaming algorithm can "remember" more than a $o(1)$ fraction of the edges of the distinguished matching M_r. Since phase 1 of our construction is identical to the one in Appendix H of [GKK12] which shows a 3/2 semi-streaming lower bound for max *weight* matching., we can reuse their result here.

Lemma 7.7.5 (Appendix H.1 of [GKK12])**.** *For any constants $\gamma, \delta \in (0, 1/4)$, let $\mathcal{A}$ be an algorithm that at the end of phase 1, with constant probability, outputs at least γn of the the edges of M'_r. Then $\mathcal{A}$ uses $\Omega(E_1) \geq n^{1+\Omega(1/\log\log n)}$ bits of space.*

We reproduce a version of the proof in Section 7.12 for completeness. With this, we are finally ready to prove the main theorem of the section.

Theorem 7.7.6. *Assuming ETH, there exists a distribution over* MSM *instances such that any deterministic algorithm achieving an* 1.914 *approximation must use either $n^{1+\Omega(1/\log\log n)}$ space or $\Omega(2^{(\log n)^{10}})$ time.*

Proof. Our proof is a randomized polytime reduction from SETCOVER to streaming MSM. We will show that if there is a randomized streaming algorithm achieving ratio better than 1.914 for MSM, then there is an algorithm for SETCOVER achieving approximation ratio $(1-\alpha)\ln(|\mathcal{U}|)$ for constant $\alpha > 0$ that only requires polynomial extra overheard. We then argue that Lemma 7.7.2 implies that the streaming MSM algorithm must run in super polynomial time, assuming ETH.

Fix a deterministic algorithm A for streaming MSM. Now, given an instance of SETCOVER $(\mathcal{U}, \mathcal{S})$ with minimum cover size K and description size $N = |\mathcal{U}| + |\mathcal{S}|$, create a random instance of streaming MSM according to the reduction described in this section. For each bipartite graph (P_i, Q'_i) (or (Q_i, P'_i)), if the algorithm A chooses cK edges from this graph, it can select at most $2(1-c)K$ edges from E_1 that are adjacent to P_i (or Q_i). Suppose WLOG that it can always achieve the $2(1-c)k$ bound. In this case we can also assume WLOG the algorithm chooses the same number $c \cdot K$ of edges from each such graph, and furthermore that it selects the same sets in the set system $(\mathcal{S}, \mathcal{U})$. Otherwise it can locally improve its solution by copying the solution for the best index i.

Suppose this solution achieves coverage of $(1 - e^{-c} + \gamma) \cdot |\mathcal{U}|$. Since the matchings $M_1, \ldots, M_t$ are induced, and by Lemma 7.7.5 w.h.p. the algorithm can only output $o(n)$ edges of M'_r after phase 1, the algorithm can only select $o(n)$ edges *not* incident to some P_i or Q_i. Thus the total value achieved by the algorithm is at most:

$$\begin{aligned}&\left[2(1-e^{-c}+\gamma)\cdot\eta\cdot K+2(1-c)\cdot K\right]\cdot\frac{n}{K}+o(n)\\&\leq 2(1-e^{-c}+\gamma)\cdot\eta n+2(1-c)\cdot n+o(n)\\&\leq (2\eta-2\ln(\eta)+2\gamma)\cdot n+o(n),\end{aligned}$$

where the last step follows since the expression is maximized at $c=\ln\eta$. On the other hand, the optimal solution can select the distinguished matching edge M'_r, as well as K edges adjacent to each set P_i corresponding to the minimum SETCOVER solution. Thus the total value of OPT is at least:

$$(1-\delta+2\eta)\cdot n.$$

Thus the ratio between the maximum value achievable by the algorithm and the optimal value is bounded by:

$$\frac{1+2\eta-\delta}{2\eta-2\ln(\eta)+2\gamma+o(1)}.$$

Finally, we set $\eta=2.09$ and let $\delta\to 0$. If this ratio converges to a value strictly below 1.914, then we can conclude that $\gamma=\Omega(1)$ and $\gamma>0$.

We have shown that A can be used to pick cK sets with coverage $(1-e^{-c}+\gamma)\cdot|\mathcal{U}|$. To finish the proof, we now show that this can be used to recover an approximation algorithm B for SETCOVER. For convenience, set constant γ' such that $(1-e^{-c-\gamma'}):=(1-e^{-c}+\gamma)$. Then, guess K, and repeat algorithm A recursively $\lceil\ln|\mathcal{U}|/(c+\gamma')\rceil\leq\ln|\mathcal{U}|/(c+\gamma')+1$ times, each time on the residual uncovered set system. Each call to A covers $(1-e^{-c-\gamma'})$ of the elements remaining, so after this number of iterations, the fraction of uncovered elements is less than $1/|\mathcal{U}|$, i.e. all elements are covered. Since each iteration costs $c\cdot K$, the total number of sets picked here is at most

$$c\left(\frac{\ln|\mathcal{U}|}{c+\gamma'}+1\right)\cdot K=\left(\frac{c}{c+\gamma'}+\frac{c}{\ln|\mathcal{U}|}\right)\cdot\ln|\mathcal{U}|\cdot K.$$

Defining the constant $\alpha=\gamma'/(c+\gamma')$, this is a $(1-\alpha-o(1))\ln|\mathcal{U}|$ approximation to the SETCOVER instance. Furthermore, if A runs in time T then B runs in time $\mathrm{poly}(N)\cdot T$ (where $N=|\mathcal{U}|+|\mathcal{S}|$).

To conclude, if $T<2^{N^{\Delta}}$ for a constant $\Delta<\gamma\cdot\alpha$, then B runs faster than $2^{N^{\gamma\alpha}}$, contradicting Lemma 7.7.2. Thus the algorithm A must run in time at least $2^{N^{\gamma\cdot\alpha}}\geq 2^{|\mathcal{S}|^{\gamma\cdot\alpha}}\geq 2^{(\log n)^{d\cdot\gamma\cdot\alpha}}$. Setting $d=10/(\gamma\cdot\alpha)$, this running time is $2^{(\log n)^{10}}$, which is superpolynomial in n. □

Theorem 7.1.3 therefore follows from Theorem 7.7.6 and Yao's minimax principle [Yao77].

7.8 Conclusion

In this chapter we present a number of improved bounds for streaming submodular (b-)matching. Our work suggests a number of natural follow-up directions.

The first is to tighten (and ideally close) the gap between upper and lower bounds for the problems studied in this chapter. More ideas may be needed in order to resolve the optimal approximation ratio for these problems.

On the techniques side, it would be interesting to see whether the primal-dual method can be used for further applications in streaming submodular algorithms, or to further unify the analysis of prior work. More broadly, it would be interesting to apply our extension of the randomized primal dual method of [DJK13] to other problems.

Finally, we point to a possible connection between our streaming matching algorithms, and *dynamic* matching algorithms. Algorithm 14 when applied to monotone b-matching problems stores in its stack a bounded-degree subgraph with $\tilde{O}(M_{\max}) = \tilde{O}(|OPT|)$ edges. Such size-optimal constant-approximate matching sparsifiers have proven particularly useful in the dynamic matching literature. See [Waj20] for a discussion, and [ACC^{+}18, GP13, PS16, BHI18, BS15, BS16] for more applications of such dynamic matching sparsifiers. Is there a fast dynamic algorithm which maintains a sparisifier similar to that of Algorithm 14? This would result in improved dynamic weighted matching algorithms, and possibly would result in the first dynamic *submodular* matching algorithm.

7.9 Explaining Prior Work using LP Duality

In this section we further demonstrate the generality of our (randomized) primal-dual analysis, showing that it provides fairly simple alternative analyses of the algorithms of [CK15, FKK18], giving one unified analysis for these algorithms and ours. To keep things simple, we focus only on MSM, though [CK15, FKK18] show that their algorithms also work more broadly for k-matchoid and k-set system constraints.

In [CK15], Chakrabarti and Kale presented a reduction from MSM to MWM, by showing how to use a subclass of MWM algorithms to solve MSM. We now introduce the algorithm of [CK15] instantiated with the MWM algorithm of [McG05]. The algorithm is a natural and elegant one: when an edge e arrives, we consider its marginal gain with respect to the current matching. If this marginal gain is higher than some slack parameter C times the marginal gains of the currently blocking edges $e' \in N(e) \cap M$, we preempt those edges and add e to the matching.

In anticipation of our analysis of the algorithm of [FKK18] in Section 7.9.2, we generalize the algorithm's description and allow the algorithm to preempt with some probability $q \in [0, 1]$. The full pseudo-code is given in Algorithm 15.

This algorithm only ever adds an edge e to M upon its arrival. After adding an edge to M, this edge can be *preempted*, i.e., removed from M, after which it is never added back to M. Thus we note that this algorithm is not only a streaming algorithm, but also a so-called *preemptive* algorithm: it only stores a single matching in memory and therefore trivially requires $\tilde{O}(n)$ space.

For convenience, we let $M^{(t)}$ denote the matching M at time t, and let $S := \bigcup_t M^{(t)}$ denote the set of edges ever added to M. For an edge e let $B^{(t)}(e) := \sum_{e' \in N(e) \cap M^{(t)}} f(e' : M^{(t)})$. We will also denote by $P := S \setminus M$ the set of *preempted* edges.

7.9.1 The Framework of [CK15], Applied to the Algorithm of [McG05]

In this section we analyze the deterministic algorithm obtained by applying the framework of [CK15] to the MWM algorithm of [McG05], corresponding to Algorithm 15 run with $q = 1$.

Algorithm 15 The MSM Algorithm of [CK15] and [FKK18]

```
    Initialization
1:  $M \leftarrow \emptyset$
    In Stream
2:  for $t \in \{1, \ldots, |E|\}$ do
3:      $e \leftarrow e^{(t)}$, $t^{th}$ edge in stream.
4:      $B(e) \leftarrow \sum_{e' \in N(e) \cap M} f(e' : M)$.
5:      if $f(e : M) \leq C \cdot B(e)$ then
6:          Skip edge $e$ and continue.
7:      else
8:          with probability $q$ do
9:              $M \leftarrow (M \setminus N(e)) \cup \{e\}$.
10: return $M$
```

To argue about the approximation ratio, we will again fit a dual solution to this algorithm. Define the auxiliary submodular functions $g^S : 2^E \to \mathsf{R}^+$ to be $g^S(T) := f(S \cup T)$. Similarly to our analysis of Algorithm 14, we define the following dual.

$$\begin{aligned}
&\mu := f(S) = g^S(\emptyset), \\
&\phi_v := C \cdot \max\{f(e : M^{(t)}) \mid t \in [|E|],\ v \in e \in M^{(t)}\}, \\
&\lambda_e := \begin{cases} f(e : S) & e \notin S \\ 0 & e \in S. \end{cases}
\end{aligned}$$

Note the difference here in the setting of ϕ_v from the algorithms of Sections 7.4 and 7.5. We start by showing that this is a dual feasible solution to the LP (D) for the function g^S.

Lemma 7.9.1. *The dual solution $(\vec{\lambda}, \vec{\phi}, \mu)$ is feasible for the LP (D) with function g^S.*

Proof. To see that the first set of constraints are satisfied, note that by submodularity of f,

$$\sum_{e \in T} \lambda_e = \sum_{e \in T \setminus S} f(e : S) \geq \sum_{e \in T \setminus S} f_S(e) \geq f_S(T \setminus S) = f(S \cup T) - f(S) = g^S(T) - \mu.$$

For the second set of constraints, we note that if $e = e^{(t)} \notin S$, then by the test in Line 5 and submodularity, we have that

$$\lambda_e = f(e : S) \leq f(e : M^{(t-1)}) \leq C \cdot B^{(t-1)}(e) \leq \sum_{v \in e} \phi_v. \qquad \square$$

It remains to relate the value of the solution M to the cost of this dual. For this, we introduce the following useful notation. For any edge $e \in S$, we define the weight of e to be

$$w_e := \begin{cases} f(e : M) & e \in M \\ f(e : M^{(t)}) & e \in M^{(t-1)} \setminus M^{(t)} \subseteq P. \end{cases}$$

In words, the weight of an edge in the matching is $f(e : M)$, and the weight of a preempted edge is frozen to its last value before the edge was preempted. One simple consequence of the definition of the weights w_e is the following relationship to $f(M)$.

Observation 7.9.2. $f(M) = \sum_{e\in M} f(e : M) = \sum_{e\in M} w_e = w(M)$.

We now show that the preempted edges' weight is bounded in terms of the weight of M.

Lemma 7.9.3. *The weights of P and M satisfy $w(P) \leq w(M) \cdot \frac{1}{C-1}$.*

Proof. For any edge e, we define the following set of preempted edges which are preempted in favor of$e$ or in favor of an edge (recursively) preempted due to e.[5] First, for an edge $e = e^{(t)}$, we let the set $P^1(e) := N(e) \cap M^{(t-1)}$ denote the edges preempted when e is added to M. For any $i > 1$, we let $P^i(e) := P^1(P^{i-1}(e))$ be the set of edges preempted by an edge with a trail of preemptions of length $i-1$ from e. By Line 6, we have that any edge $e \in S$ has weight at least $w_e \geq C \cdot P^1(e)$. By induction, this implies that $w_e \geq C \cdot w(P^{(i-1)}(e)) \geq C^i \cdot w(P^i(e))$. Now, since each preempted edge $e' \in P$ belongs to precisely one set $P^i(e)$ for some $i \geq 1$ and $e \in M$, we find that indeed,

$$w(P) = \sum_{e\in M}\sum_{i\geq 1} w(P^i(e)) \leq \sum_{e\in M}\sum_{i\geq 1} \frac{1}{C^i} \cdot w_e = w(M) \cdot \left(\frac{1}{C} + \frac{1}{C^2} + \dots\right) = w(M) \cdot \frac{1}{C-1}.$$

□

Using Lemma 7.9.3, we can now relate the value of the primal solution M to the cost of the our dual solution, $\mu + \sum_v \phi_v$. We start by bounding μ in terms of $f(M)$.

Lemma 7.9.4. *The matching M output by Algorithm 15 satisfies $f(M) \geq \left(1 - \frac{1}{C}\right) \cdot \mu$.*

Proof. By submodularity of f, Lemma 7.9.3, and Observation 7.9.2 we obtain the desired inequality,

$$\mu = f(S) = f(M \cup P) \leq f(M) + \sum_{e\in P} f(e : M) = w(M) + w(P) \leq \left(1 + \frac{1}{C-1}\right) \cdot f(M).$$ □

We next bound $\sum_v \phi_v$ in terms of $f(M)$.

Lemma 7.9.5. *The matching M output by Algorithm 15 run with $C > 1$ satisfies*

$$f(M) \geq \frac{1}{2C + C/(C-1)} \cdot \sum_{v\in V} \phi_v.$$

Proof. Fix a vertex v and edge $e \in M^{(t-1)} \setminus M^{(t)} \subseteq P$ preempted at time t in favor of edge $e' = e^{(t)} \ni v$. For this edge e, by monotonicty in t' of $f(e' : M^{(t')})$, the test of Line 5, non-negativity of $f(e'' : M^{(t-1)})$ for any edge $e'' \in M^{(t-1)}$ and $C > 1$ we have that

$$w_{e'} \geq f(e' : M^{(t-1)}) \geq C \cdot B^{(t-1)}(e') \geq C \cdot f(e : M^{(t-1)}) = C \cdot w_e > w_e.$$

Consequently, again relying on monotonicity in t' of $f(e' : M^{(t')})$, we have that for any edge $e \in P$, there is at most one vertex $v \in e$ such that $w_e = f(e : M^{(t-1)})$ is equal to $\phi_v = \max\{f(e' : M^{(t')}) \mid v \in e' \in M^{(t')}\}$. Edges $e \in M$, on the other hand, clearly have $w_e = f(e : M)$ equal to $\phi_v = \max\{f(e' : M^{(t-1)}) \mid v \in e' \in M^{(t-1)} \setminus M^{(t)}\}$ for at most two vertices $v \in e$. Combined with Lemma 7.9.3 and Observation 7.9.2, this yields the desired inequality,

$$\sum_{v\in V} \phi_v \leq C \cdot \left(2 \sum_{e\in M} w_e + \sum_{e\in P} w_e\right) \leq \left(2C + \frac{C}{C-1}\right) \cdot w(M) = \left(2C + \frac{C}{C-1}\right) \cdot f(M).$$ □

[5] In [FKM+05, McG05], these sets are referred to by the somewhat morbid term "trail of the dead".

Equipped with the above lemmas, we can now analyze Algorithm 15's approximation ratio.

Theorem 7.9.6. *Algorithm 15 run with $C > 1$ and $q = 1$ on a monotone* MSM *instance outputs a matching M of value*

$$\left(2C + \frac{2C}{C-1}\right) \cdot f(M) \geq f(\textsc{Opt}).$$

This is optimized by taking $C = 2$, resulting in an approximation ratio of 8.

Proof. By weak LP duality and Lemma 7.9.1, together with monotonicity of f, we have that

$$C \cdot \sum_v \phi_v + \mu \geq \max_T g^S(T) = \max_T f(S \cup T) \geq f(S \cup \textsc{Opt}) \geq f(\textsc{Opt}).$$

Combining Lemma 7.9.5 and $f(M) = \mu$ by definition and rearranging, we get the desired inequality,

$$\left(2C + \frac{2C}{C-1}\right) \cdot f(M) \geq C \cdot \sum_v \phi_v + \mu \geq f(\textsc{Opt}). \qquad \square$$

As with our algorithm of Section 7.4, our analysis of Algorithm 15 relied on monotonicity, crucially using $f(S \cup OPT) \geq f(OPT)$. To extend this algorithm to non-monotone MSM, we again appeal to Lemma 7.2.1, setting $q = \frac{1}{2C+1}$. This is precisely the algorithm of [FKK18], which we analyze in the following section.

7.9.2 The Algorithm of [FKK18]

In [FKK18], Feldman et al. showed how to generalize the algorithm of [CK15] to non-monotone function maximization. Here we show an analysis of their algorithm in our primal dual framework. Our proof is an extension of the one in Section 7.9.1 in a way that is analogous to how Section 7.5 extends Section 7.4.

We reuse the same dual from Section 7.9.1, only this time, both our dual object and the function g^S are random variables. The proof of expected dual feasibility for this variant of the algorithm of Section 7.9.1 is analogous to that of Lemma 7.5.1, so we only outline the differences here.

We start with expected feasibility.

Lemma 7.9.7. *The dual solution $(\mathbb{E}[\vec{\lambda}], \mathbb{E}[\vec{\phi}], \mathbb{E}[\mu])$ is feasible for the expected LP $\mathbb{E}[(D)]$.*

Proof (Sketch). The first set of constraints is satisfied for any random realization. Indeed, as in the proof of Lemma 7.9.1, for any realization of S, by submodularity of f, we have

$$\sum_{e \in T} \lambda_e = \sum_{e \in T \setminus S} f(e : S) \geq \sum_{e \in T \setminus S} f_S(e) \geq f_S(T \setminus S) = f(S \cup T) - f(S) = g^S(T) - \mu.$$

Consequently, taking expectation over S, we have that indeed, $\mathsf{E}_S[\mu] + \sum_{e \in T} \mathsf{E}_S[\lambda_e] \geq \mathsf{E}_S[g^S(T)]$.

For the second set of constraints, the proof is nearly identical to that of Lemma 7.5.1, where we show that

$$\mathbb{E}\left[\sum_{v \in e} \phi_v\right] \geq \mathbb{E}[\lambda_e].$$

This is proved by taking total probability over the event $A_e := [f(e : S) \leq C \cdot \sum_{v \in V} \phi_v^{(t-1)}]$ and its complement. The key inequality to prove here is that for any realization of randomness R for which $\overline{A_e}$ holds, we have that

$$\mathbb{E}\left[\sum_{v \in e} \phi_v^{(t)} \;\middle|\; R\right] = 2q \cdot f(e : S) + (1 - 2q) \cdot \sum_{v \in e} \phi_v^{(t-1)} \geq 2q \cdot f(e : S).$$

And indeed, conditioned on R, the edge $e = e^{(t)}$ fails the test in Line 5, and so with probability q, we have $\sum_{v \in e} \phi_v^{(t)} = 2 \cdot f(e : S)$. To see this, note that if e is added to the matching, then for both $v \in e$, by definition $\phi_v^{(t)}$ must be at least $f(e : S)$. Hence, in this case

$$\mathbb{E}\left[\sum_{v \in e} \phi_v^{(t)} \;\middle|\; R\right] \geq 2q \cdot f(e : S).$$

The proof then proceeds as that of Lemma 7.5.1. □

To relate the value of the solution M to the cost of the dual, we can define weights as in Section 7.9.1 and reuse lemmas 7.9.3, 7.9.4, and 7.9.5, which hold for every realization of the random choices of the algorithm. From here, following our template, we can use these along with LP duality, Lemma 7.9.7 and Lemma 7.2.1, to analyze this algorithm.

Theorem 7.9.8. *Algorithm 15 run with $q = 1/(2C+1)$ and C on a non-monotone* MSM *instance outputs a matching M of value*

$$\left(\frac{2C^2 + C}{C - 1}\right) \cdot f(M) \geq f(\textsc{Opt}).$$

This is optimized by taking $C = 1 + \frac{\sqrt{3}}{2}$, resulting in an approximation ratio of $5 + 2\sqrt{6} \approx 9.899$. Moreover, the same algorithm is $2C + 2C/(C - 1)$ approximate for monotone MSM, *yielding an approximation ratio of 8 for $C = 2$.*

7.10 Tight instance for Algorithm 14

In this section we show that there exists a family of instances of MSM instances parametrized by C for which Algorithm 14 with parameter $C > 1$ yields an approximation factor of $2C + C/(C - 1)$.

Lemma 7.10.1. *The approximation ratio of Algorithm 14 with $C > 1$ and $q = 1$ for monotone MSM is at least $2C + \frac{C}{C-1}$.*

Proof. Define the graph G as follows. The vertex set $V(G)$ consists of $\{x_i, y_i\}_{i \in [0,n]}$. For convenience, for every $i \in [1, n]$ we define the edges $d_i = (x_0, x_i)$ and $e_i = (x_i, y_i)$. Then the edge set $E(G)$ consists of the edges $\{d_i\}_{i=1}^n \cup \{e_i\}_{i=0}^n$.

To define the (monotone) submodular function, we first define an auxiliary weight function $w : E(G) \to \mathsf{R}_{\geq 0}$. The weights are:

$$w(d_i) = C^{i-1} \qquad (n \geq i \geq 1)$$
$$w(e_1) = 1 + C - \epsilon$$

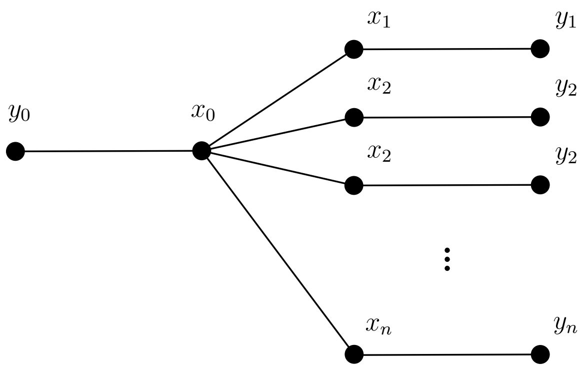


Figure 7.2: Tight Example for Algorithm 14

$$w(e_i) = C^i - \epsilon \qquad (n \geq i \geq 2)$$
$$w(e_0) = C^n - \epsilon$$

Now the submodular function is:

$$f(T) := w(T \cap \{e_0\}) + \sum_{i=0}^{n} \min(w(T \cap \{d_i, e_i\}), w(e_i))$$

Since weights are non-negative, this function is monotone. Submodularity follows from preseration of subdmodularity under linear combinations (and in particular sums), and $\min\{w(S), X\}$ being submodular for any linear function w.

The stream reveals the edges $d_1, \ldots, d_n$ in order, and subsequently reveals $e_0, e_1, \ldots, e_n$ in order. For a run of Algorithm 14 with this choice of C and $q = 1$, several claims hold inductively:

(a) On the arrival of edge d_i, we have $\phi_{x_0} = C^{i-2}$ (except for the arrival of d_1, at which point $\phi_0 = 0$) and $\phi_{x_i} = 0$.

(b) The algorithm takes every edge d_i into the stack.

(c) After d_i is taken into the stack, we have $\phi_{x_0} = C^{i-1}$ and $\phi_{x_i} = C^{i-1} + C^{i-2}$ (except for ϕ_{x_1} which is set to 1).

(d) The algorithm does not take e_i into the stack.

Let Λ_t be the statement that these claims holds for time t. By inspection Λ_1 holds, now consider some time $i > 1$. Claim (a) follows directly from claim (c) of Λ_{i-1}. Claim (b) follows from (a) since $f_S(d_i) = C^{i-1} = C \cdot \phi_0$ when d_i arrives. Claim (c) is a consequence of how the algorithm increases the potentials ϕ when taking edges into the stack. Claim (d) holds since $f_S(e_i) = w(e_i) - w(d_i) = C^i - C^{i-1} - \epsilon < C \cdot \phi_{x_i}$.

From the above, we find that Algorithm 14 with parameter C as above and $q = 1$ will have all edges $d_1, \ldots, d_n$ in its stack by the end, resulting in it outputting the matching consisting of the single edge d_n. The value of this edge (and hence this matching) is C^{n-1}, while on the other hand OPT can take the edges $\{e_i\}_{i=0}^n$, which have value

$$\sum_{i=0}^{n} w(e_i) = C^n + \sum_{i=0}^{n} C^i - \epsilon(n+1) \to C^{n-1}\left(2C + \frac{C}{C-1}\right). \qquad \text{(as } n \to \infty \text{ and } \epsilon \to 0\text{)}$$

so long as $C > 1$. Hence $c(\textsc{Opt})/c(\textsc{Alg}) \to 2C + C/(C-1)$. The lemma follows. □

7.11 Space Bound of Algorithm 14

In this section we bound the space and time complexities of Algorithm 14. We start by bounding its space usage, as restated in the following lemma.

Lemma 7.3.1. *For any constant $\epsilon > 0$, Algorithm 14 run with $C = 1 + \epsilon$ uses $\tilde{O}(M_{\max})$ space.*

First, we prove the following simpler lemma.

Lemma 7.11.1. *Each vertex $v \in V$ has at most $\tilde{O}(b_v)$ edges in the stack S.*

Proof. If an edge $e \ni v$ is added to S at time t, then by the test in Line 6, $f(e : S) \geq (1+\epsilon) \cdot \sum_{u \in e} \phi_u^{(t-1)}$. Consequently, and since ϕ values are easily seen to always be positive, we have

$$\begin{aligned}\phi_v^{(t)} - \phi_v^{(t-1)} &= \frac{f(e:S) - \sum_{u\in e} \phi_u^{(t-1)}}{b_v} \\ &\geq \frac{\epsilon \cdot \sum_{u \in e} \phi_u^{(t-1)}}{b_v} \geq \frac{\epsilon \cdot \phi_v^{(t-1)}}{b_v}.\end{aligned}$$

Thus, adding this edge $e \ni v$ to S results in $\phi_v^{(t)} \geq \phi_v^{(t-1)} \cdot (1 + \epsilon/b_v)$. Moreover, if e is the first edge of v added to S, then we have

$$\begin{aligned}\phi_v^{(t)} &= \frac{f(e:S) - \sum_{u\in e} \phi_u^{(t-1)}}{b_v} \geq \frac{\epsilon}{1+\epsilon} \cdot \frac{f(e:S)}{b_v} \\ &\geq \frac{\epsilon}{1+\epsilon} \cdot \frac{f_{\min}}{b_v}.\end{aligned}$$

Hence, if v had k edges added to the stack by time t,

$$\phi_v^{(t)} \geq (\epsilon/(1+\epsilon)) \cdot (f_{min}/b_v) \cdot (1+\epsilon/b_v)^{k-1}. \tag{7.11.1}$$

On the other hand, since f is polynomially bounded, we have that for some constant d

$$\phi_v^{(t)} \leq \sum_{e \ni v} f_{S_e}(e)/b_v \leq n^d \cdot (f_{min}/b_v). \tag{7.11.2}$$

Combining equations (7.11.1) and (7.11.2) and simplifying, we find that $(1+\epsilon/b_v)^{k-1} \leq n^d \cdot (1+\epsilon)/\epsilon$. Taking out logarithms and simplifying further, we find that

$$k \leq 1 + \frac{d \log n + \log(1+\epsilon) + \log(1/\epsilon)}{\log(1+\epsilon/b_v)} = O((b_v/\epsilon) \cdot (\log n + \log(1/\epsilon))) = \tilde{O}(b_v).$$

That is, the number of edges of v in the stack is at most $\tilde{O}(b_v)$. □

Proof of Lemma 7.3.1. Consider a maximum cardinality b-matching M (i.e., $|M| = M_{\max}$). Denote by d_v the degree of a vertex v in G. Then, the set U of saturated vertices $v \in V$ in M, i.e., those with $\min\{b_v, d_v\}$ edges in M, is a vertex cover. To see this, we note that no edge $e = (u, v)$ can have both endpoints not saturated by M, as the converse would imply that $M \cup \{e\}$ is a feasible b-matching, contradicting maximality of M. Now, since each edge of M has at most two saturated endpoints, we have that $\sum_{u \in U} \min\{b_u, d_u\} \leq 2|M|$. On the other hand, since U is a vertex cover, the number of edges in S is upper bounded by the number of edges of vertices in U

in S. But each vertex $u \in U$ trivially has at most d_u edges in S. Combined with Lemma 7.11.1, we find that the number of edges in S is at most

$$|S| = \tilde{O}\left(\sum_{u\in U} \min\{b_u, d_u\}\right) = \tilde{O}(|M|). \qquad \square$$

Finally, since each edge added to S causes at most two nodes to have non-zero ϕ value, and since each such value can be specified using $\tilde{O}(1)$ bits, by the assumption of f being poly-bounded, the space to store all non-zero ϕ values is at most $\tilde{O}(|S|) = \tilde{O}(|M|) = \tilde{O}(M_{\max})$.

Finally, we briefly analyze the algorithm's running time.

Lemma 7.3.2. *A randomized (deterministic) implementation of Algorithm 14 requires $O(1)$ ($O(\log n)$) operations and $O(1)$ function evaluations per arrival, followed by $\tilde{O}(M_{\max})$ time post-processing. Using $\tilde{O}(n)$ space, the deterministic time per arrival can be decreased to $O(1)$, by storing all ϕ_v.*

Proof. The algorithm clearly requires a constant number of evaluations of ϕ_v and f for each edge arrival. The difference between deterministic and randomized implementations is due to the data structures used to maintain the mapping from v to ϕ_v (if non zero). Finally, the preprocessing time follows directly from our upper bound on $|S|$ and the post-processing stage clearly taking time linear in $|S|$. $\square$

7.12 Deferred Proofs of Section 7.7

Lemma 7.7.2 (Extension of Corollary 1.6 of [DS14]). *Assuming ETH, every algorithm achieving an approximation ratio $(1-\alpha)\ln|\mathcal{U}|$ for* Set Cover *runs in time strictly greater than $2^{N^{\gamma\cdot\alpha}}$ for some $\gamma > 0$. Furthermore, this holds even under the assumptions that $|\mathcal{S}| \leq K^{1/(\gamma\alpha)}$ and $|\mathcal{U}| \leq |\mathcal{S}|^{1/(\gamma\alpha)}$.*

Proof. The first statement is precisely Corollary 1.6 of [DS14].

For the extra assumptions, if $K < |\mathcal{S}|^{\gamma\alpha}$ then the brute force algorithm that checks all subsets of size K runs in time $|\mathcal{S}|^K < 2^{|\mathcal{S}|^{\gamma\alpha}\log|\mathcal{S}|} \leq 2^{N^{\gamma\alpha}}$. If $|\mathcal{S}| < |\mathcal{U}|^{\gamma\alpha}$, then one can brute force over all sub collections of S in time $2^{|\mathcal{S}|} \leq 2^{|\mathcal{U}|^{\gamma\alpha}} \leq 2^{N^{\gamma\alpha}}$. Both running times contradict Lemma 7.7.2. $\square$

Lemma 7.7.5 (Appendix H.1 of [GKK12]). *For any constants $\gamma, \delta \in (0, 1/4)$, let $\mathcal{A}$ be an algorithm that at the end of phase 1, with constant probability, outputs at least γn of the the edges of M'_r. Then $\mathcal{A}$ uses $\Omega(E_1) \geq n^{1+\Omega(1/\log\log n)}$ bits of space.*

Proof. Let $\mathcal{A}$ be an algorithm that outputs γn of the edges of M'_r at the end of phase 1 that uses fewer than $s = n \operatorname{poly}\log n$ bits. We will show that $\gamma = o(1)$.

Let $\mathcal{G}$ be the set of possible first phase graphs. Then

$$|\mathcal{G}| = \binom{n/2}{\delta n}^t = 2^{\gamma m}$$

for some $\gamma > 0$. Let $\phi : \mathcal{G} \to \{0,1\}^s$ be the function that takes an input graph G to the state of the algorithm $\mathcal{A}$ after running $\mathcal{A}$ on G. Let $\Gamma(G) = \{H \mid \phi(G) = \phi(H)\}$, that is the set of graphs inducing the same internal state for $\mathcal{A}$ at the end of phase 1.

Define $\Psi(G) = \bigcap_{H \in \Gamma(G)} E(H)$. Note that for any input graph G, the algorithm $\mathcal{A}$ can output an edge e if and only if $e \in \Psi(G)$. Also, for any G let t' be the number of matchings in the RS graph G_0 for which $\Psi(G)$ contains at least γn edges. Since algorithm $\mathcal{A}$ outputs γn edges of M'_r, the number of graphs in $\Gamma(G)$ is bounded by

$$\binom{(1/2-\gamma)n}{\delta n}^{t'} \binom{n/2}{\delta n}^{t-t'} = \left(2^{-\Omega(\gamma n)} \binom{n/2}{\delta n}\right)^{t'} \binom{n/2}{\delta n}^{t-t'} = 2^{-\Omega(t'\gamma n)} 2^{\gamma m}. \qquad (*)$$

On the other hand, since the first phase graph G is chosen uniformly at random, by a counting argument, with probability at least $1 - o(1)$ we have that $|\Gamma(G)| \geq 2^{(\gamma - o(1))m}$. Conditioning on this happening, we also know that $t' \geq \Omega(t)$ since the input graph is uniformly chosen within $\Gamma(G)$, and the algorithm succeeds with constant probability. These two facts together with $(*)$ imply that $\gamma = o(1)$. □

Chapter 8

Conclusion and Open Questions

In thesis, we give algorithms for several important of instances of submodular optimization under uncertainty. Our results generalize, refine, and improve a number previous methods. We hope the reader is convinced that the intersection of submodular optimization and algorithms under uncertainty is a rich and still burgeoning area. In particular, we believe that it serves as an excellent sandbox in which to explore fundamental algorithmic ideas.

Many interesting questions in the area remain. Below, we collect some of the outstanding open problems from this thesis.

Online Algorithms. Is possible to extend the LEARNORCOVER technique to give an $O(\log mn)$ competitive algorithm for covering IPs *with box constraints*? Beyond that, can one improve the adversarial order bounds for the more general SUBMODULARCOVER problem? Another question is whether the result for NONMETRICFACILITYLOCATION extends to GROUPSTEINERTREE. In particular, can one get an $O(\log^3 n)$ competitive algorithm when the terminals are revealed in random order? Finally, we show an $O(\log(mn)/\alpha)$ competitive ratio for α-SAMPLESETCOVER, but what is the right dependence on α? We conjecture that $O(\log(mn)\log(1/\alpha))$ is possible.

Can our $O(\log^2 k)$ competitive ratio for block-caching in the eviction cost model be improved to $O(\log k)$ to match the lower bound from classical paging? Note that even for the special case of generalized caching, the first result of [BBN12b] achieved competitive ratio $O(\log^2 k)$. The subsequent improvement due to [ACER19] required non-trivial ideas which seem difficult to adapt to the more general block-aware caching problem. For the fetching cost model, is there an algorithm matching the lower bound of $\beta + \log k$, or can the lower bound be strengthened to match the trivial $\beta \log k$ upper bound? Finally, are there constant-approximation *offline* algorithms for block-caching in either the fetching or eviction cost models?

Dynamic Algorithms. Is it possible to remove the $\log(c_{\max}/c_{\min})$ in the recourse bound of Theorem 6.3.1? Are there instances where this cost is incurred, or this just an analysis artifact? Another interesting orthogonal question is whether there is a fully-dynamic algorithms for SUBMODULARCOVER with fast *update time* (as opposed to just recourse). The naive implementation of our local search requires polynomial time, since it must greedily search for local moves, as well as preform bubble sort between any two such moves. Since known algorithms for dynamic SETCOVER with update time bounds ([GKKP17]) explicitly maintain an assignment of elements to sets, it is conceivable that one may need more than black box access to a value oracle. This question is interesting even for special cases of SUBMODULARCOVER such as partial cover, or the SIMULTANEOUSSOURCELOCATION problem we discuss in Chapter 1.

Streaming Algorithms. Can one tighten (and ideally close) the gap between upper and lower bounds for Streaming MSM? On the techniques side, it would be interesting to see whether the primal-dual method can be used for further applications in streaming submodular algorithms, or to further unify the analysis of prior work. More broadly, it would be interesting to apply our extension of the randomized primal dual method of [DJK13] to other problems. Is there a fast dynamic algorithm which maintains a sparisifier similar to that of Algorithm 14? This would result in improved dynamic weighted matching algorithms, and possibly would result in the first dynamic *submodular* matching algorithm.

Bibliography

[AAA$^+$06] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. A general approach to online network optimization problems. *ACM Trans. Algorithms*, 2(4):640–660, 2006. 2.5, 2

[AAA$^+$09] Noga Alon, Baruch Awerbuch, Yossi Azar, Niv Buchbinder, and Joseph Naor. The online set cover problem. *SIAM J. Comput.*, 39(2):361–370, 2009. 2.5, 1, 3.1, 3.1.1, 3.5, 4.1.1, 4.7.1, 4.12, 5.1

[AAG$^+$19] Amir Abboud, Raghavendra Addanki, Fabrizio Grandoni, Debmalya Panigrahi, and Barna Saha. Dynamic set cover: improved algorithms and lower bounds. In Moses Charikar and Edith Cohen, editors, *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 114–125. ACM, 2019. 6.1.3

[AAK99] Susanne Albers, Sanjeev Arora, and Sanjeev Khanna. Page replacement for general caching problems. In Robert Endre Tarjan and Tandy J. Warnow, editors, *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland, USA*, pages 31–40. ACM/SIAM, 1999. 5.1

[AAK19] Arpit Agarwal, Sepehr Assadi, and Sanjeev Khanna. Stochastic submodular cover with limited adaptivity. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 323–342. SIAM, 2019. 3.1.2

[ACC$^+$18] Moab Arar, Shiri Chechik, Sarel Cohen, Cliff Stein, and David Wajc. Dynamic matching: Reducing integral algorithms to approximately-maximal fractional algorithms. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 7.8

[ACER19] Anna Adamaszek, Artur Czumaj, Matthias Englert, and Harald Räcke. An $O(\log k)$-competitive algorithm for generalized caching. *ACM Trans. Algorithms*, 15(1):6:1–6:18, 2019. 5.1, 5.1.1, 5.5, 8

[ACN00] Dimitris Achlioptas, Marek Chrobak, and John Noga. Competitive analysis of randomized paging algorithms. *Theor. Comput. Sci.*, 234(1-2):203–218, 2000. 5.1

[AD15] Shipra Agrawal and Nikhil R. Devanur. Fast algorithms for online stochastic convex programming. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA,* *January 4-6, 2015*, pages 1405–1424. SIAM, 2015. 4.1.3

[ADF17] Faez Ahmed, John P. Dickerson, and Mark D. Fuge. Diverse weighted bipartite b-matching. In Carles Sierra, editor, *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 35–41. ijcai.org, 2017. 1, 7.1

[AEF$^+$20] Naor Alaluf, Alina Ene, Moran Feldman, Huy L. Nguyen, and Andrew Suh. Optimal streaming algorithms for submodular maximization with cardinality constraints. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 7.1

[AGG$^+$09] Konstantin Andreev, Charles Garrod, Daniel Golovin, Bruce M. Maggs, and Adam Meyerson. Simultaneous source location. *ACM Trans. Algorithms*, 6(1):16:1–16:17, 2009. 1, 1.1.1, 3.1

[AGHI09] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In Ricardo Baeza-Yates, Paolo Boldi, Berthier A. Ribeiro-Neto, and Berkant Barla Cambazoglu, editors, *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, pages 5–14. ACM, 2009. 1, 7.1

[AGTG21] C. J. Argue, Anupam Gupta, Ziye Tang, and Guru Guruganesh. Chasing convex bodies with linear competitive ratio. *J. ACM*, 68(5):32:1–32:10, 2021. 6.1.3

[AGZ99] Matthew Andrews, Michel X. Goemans, and Lisa Zhang. Improved bounds for on-line load balancing. *Algorithmica*, 23(4):278–301, 1999. 6.1.3

[AHK12] Sanjeev Arora, Elad Hazan, and Satyen Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.*, 8(1):121–164, 2012. 4.1.3

[AJ21a] Susanne Albers and Maximilian Janke. Scheduling in the random-order model. *Algorithmica*, 83(9):2803–2832, 2021. 4.1.3

[AJ21b] Susanne Albers and Maximilian Janke. Scheduling in the secretary model. In Mikolaj Bojanczyk and Chandra Chekuri, editors, *41st IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2021, December 15-17, 2021, Virtual Conference*, volume 213 of *LIPIcs*, pages 6:1–6:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. 4.1.3

[AKL21] Susanne Albers, Arindam Khan, and Leon Ladewig. Improved online algorithms for knapsack and GAP in the random order model. *Algorithmica*, 83(6):1750–1785, 2021. 4.1.3

[Ans87] Richard P. Anstee. A polynomial algorithm for b-matchings: An alternative approach. *Inf. Process. Lett.*, 24(3):153–157, 1987. 7.6

[AWY14] Shipra Agrawal, Zizhuo Wang, and Yinyu Ye. A dynamic near-optimal algorithm for online linear programming. *Oper. Res.*, 62(4):876–890, 2014. 4.1.3

[Bac13] Francis R. Bach. Learning with submodular functions: A convex optimization perspective. *Found. Trends Mach. Learn.*, 6(2-3):145–373, 2013. 6.1.3, 6.4.1, 2

[BBF$^+$01] Amotz Bar-Noy, Reuven Bar-Yehuda, Ari Freund, Joseph Naor, and Baruch Schieber. A unified approach to approximating resource allocation and scheduling. *J. ACM*, 48(5):1069–1090, 2001. 7.1.2

[BBK99] Avrim Blum, Carl Burch, and Adam Kalai. Finely-competitive paging. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 450–458. IEEE Computer Society, 1999. 5.1

[BBN12a] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. A primal-dual randomized algorithm for weighted paging. *J. ACM*, 59(4):19:1–19:24, 2012. 5.1, 5.1.1

[BBN12b] Nikhil Bansal, Niv Buchbinder, and Joseph Naor. Randomized competitive algorithms for generalized caching. *SIAM J. Comput.*, 41(2):391–414, 2012. 1.1.3, 5.1, 5.1, 5.1.1, 5.5, 8

[BCH17] Sayan Bhattacharya, Deeparnab Chakrabarty, and Monika Henzinger. Deterministic fully dynamic approximate vertex cover and fractional matching in O(1) amortized update time. In Friedrich Eisenbrand and Jochen Könemann, editors, *Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings*, volume 10328 of *Lecture Notes in Computer Science*, pages 86–98. Springer, 2017. 6.1.3

[BCL$^+$18] Sébastien Bubeck, Michael B. Cohen, Yin Tat Lee, James R. Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 3–16. ACM, 2018. 5.1

[BCN14a] Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive algorithms for restricted caching and matroid caching. In Andreas S. Schulz and Dorothea Wagner, editors, *Algorithms - ESA 2014 - 22th Annual European Symposium, Wroclaw, Poland, September 8-10, 2014. Proceedings*, volume 8737 of *Lecture Notes in Computer Science*, pages 209–221. Springer, 2014. 3.1.2

[BCN14b] Niv Buchbinder, Shahar Chen, and Joseph Naor. Competitive analysis via regularization. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 436–444. SIAM, 2014. 5.1

[BF18] Niv Buchbinder and Moran Feldman. Submodular functions maximization problems. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics, Second Edition, Volume 1: Methologies and Traditional Applications*, pages 753–788. Chapman and Hall/CRC, 2018. 2.4

[BF19] Niv Buchbinder and Moran Feldman. Constrained submodular maximization via a nonsymmetric technique. *Math. Oper. Res.*, 44(3):988–1005, 2019. 7.1

[BFNS14] Niv Buchbinder, Moran Feldman, Joseph Naor, and Roy Schwartz. Submodular maximization with cardinality constraints. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 1433–1452. SIAM, 2014. 7.1, 7.1.2, 7.2.2, 7.2.1

[BFT96] Avrim Blum, Merrick L Furst, and Andrew Tomkins. What to do with your free time: Algorithms for infrequent requests and randomized weighted caching. 1996. 5.1

[BGHM20] Nathan Beckmann, Phillip B. Gibbons, Bernhard Haeupler, and Charles McGuffey. Writeback-aware caching. In Bruce M. Maggs, editor, *1st Symposium on Algorithmic Principles of Computer Systems, APOCS 2020, Salt Lake City, UT, USA, January 8, 2020*, pages 1–15. SIAM, 2020. 5.1, 5.1

[BGM21] Nathan Beckmann, Phillip B. Gibbons, and Charles McGuffey. Block-granularity-aware caching. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 414–416. ACM, 2021. 5.1, 1, 5.1, 5.1.1, 5.1.1, 5.1.1, 3, 5.4, 5.4.2, 5.4.3, 5.5

[BGMN19] Niv Buchbinder, Anupam Gupta, Marco Molinaro, and Joseph (Seffi) Naor. k-servers with a smile: Online algorithms via projections. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 98–116. SIAM, 2019. 4.12

[BHI18] Sayan Bhattacharya, Monika Henzinger, and Giuseppe F. Italiano. Deterministic fully dynamic data structures for vertex cover and matching. *SIAM J. Comput.*, 47(3):859–887, 2018. 6.1.3, 7.8

[BHN19] Sayan Bhattacharya, Monika Henzinger, and Danupon Nanongkai. A new deterministic algorithm for dynamic set cover. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 406–423. IEEE Computer Society, 2019. 6.1.3

[BHNW21] Sayan Bhattacharya, Monika Henzinger, Danupon Nanongkai, and Xiaowei Wu. Dynamic set cover: Improved amortized and worst-case update time. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2537–2549. SIAM, 2021. 6.1.3

[BK19] Sayan Bhattacharya and Janardhan Kulkarni. Deterministically maintaining a $(2+\epsilon)$-approximate minimum vertex cover in $O(1/\epsilon^2)$ amortized update time. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1872–1885. SIAM, 2019. 6.1.3

[BLSZ14] Bartlomiej Bosek, Dariusz Leniowski, Piotr Sankowski, and Anna Zych. Online bipartite matching in offline time. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 384–393. IEEE Computer Society, 2014. 6.1.3

[BMKB13] Maximilian Beinhofer, Jörg Müller, Andreas Krause, and Wolfram Burgard. Robust landmark selection for mobile robot navigation. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pages 3637–2643. IEEE, 2013. 1.1.1

[BMKK14] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas

Krause. Streaming submodular maximization: massive data summarization on the fly. In Sofus A. Macskassy, Claudia Perlich, Jure Leskovec, Wei Wang, and Rayid Ghani, editors, *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 671–680. ACM, 2014. 7.1

[BN09a] Niv Buchbinder and Joseph Naor. The design of competitive online algorithms via a primal-dual approach. *Found. Trends Theor. Comput. Sci.*, 3(2-3):93–263, 2009. 5.1

[BN09b] Niv Buchbinder and Joseph Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009. 2.5, 1, 3.2.1, 3.2.3, 3.3.1, 3.3.2, 3.4, 3.4.2, 3.4.2, 4.1.1, 4.7.2, 4.1, 4.7.3, 4.7.2, 4.12, 4.12, 4.12

[BNT21] Nikhil Bansal, Joseph (Seffi) Naor, and Ohad Talmon. Efficient online weighted multi-level paging. In Kunal Agrawal and Yossi Azar, editors, *SPAA '21: 33rd ACM Symposium on Parallelism in Algorithms and Architectures, Virtual Event, USA, 6-8 July, 2021*, pages 94–104. ACM, 2021. 5.1, 5.1

[BR05] Reuven Bar-Yehuda and Dror Rawitz. On the equivalence between the primal-dual schema and the local ratio technique. *SIAM J. Discret. Math.*, 19(3):762–797, 2005. 3

[BS15] Aaron Bernstein and Cliff Stein. Fully dynamic matching in bipartite graphs. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2015. 7.8

[BS16] Aaron Bernstein and Cliff Stein. Faster fully dynamic matchings with small approximation ratios. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 692–711. SIAM, 2016. 7.8

[Bub15] Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 8(3-4):231–357, 2015. 4.12

[CCPV11] Gruia Călinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011. 1, 7.1

[CDKL09] Kamalika Chaudhuri, Constantinos Daskalakis, Robert D. Kleinberg, and Henry Lin. Online bipartite perfect matching with augmentations. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 1044–1052. IEEE, 2009. 6.1.3

[CGQ15] Chandra Chekuri, Shalmoli Gupta, and Kent Quanrud. Streaming algorithms for submodular function maximization. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, volume 9134 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2015. 7.1, 7.1.1, 7.2

[CHP$^+$19] Vincent Cohen-Addad, Niklas Hjuler, Nikos Parotsidis, David Saulpic, and Chris

Schwiegelshohn. Fully dynamic consistent facility location. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 3250–3260, 2019. 6.1.3

[Chv79] Vasek Chvátal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4(3):233–235, 1979. 2.5

[CK15] Amit Chakrabarti and Sagar Kale. Submodular maximization meets streaming: matchings, matroids, and more. *Math. Program.*, 154(1-2):225–247, 2015. 7.1, 7.1.1, 7.1.1, 7.1.1, 7.1.2, 7.9, 7.9.1, 15, 7.9.2

[CL91] Marek Chrobak and Lawrence L. Larmore. An optimal on-line algorithm for k-servers on trees. *SIAM J. Comput.*, 20(1):144–148, 1991. 5.1

[CLNT22] Christian Coester, Roie Levin, Joseph (Seffi) Naor, and Ohad Talmon. Competitive algorithms for block-aware caching. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 161–172. ACM, 2022. 1.1.3

[CM18] Micah Corah and Nathan Michael. Distributed submodular maximization on partition matroids for planning on large sensor networks. In *57th IEEE Conference on Decision and Control, CDC 2018, Miami, FL, USA, December 17-19, 2018*, pages 6792–6799. IEEE, 2018. 6.1.3

[CRT06] Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Stable signal recovery from incomplete and inaccurate measurements. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 59(8):1207–1223, 2006. 6.1.2

[CS14] Michael S. Crouch and Daniel M. Stubbs. Improved streaming algorithms for weighted matching, via unweighted matching. In Klaus Jansen, José D. P. Rolim, Nikhil R. Devanur, and Cristopher Moore, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, volume 28 of *LIPIcs*, pages 96–104. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. 7.1, 7.1.1, 7.6

[CVZ14] Chandra Chekuri, Jan Vondrák, and Rico Zenklusen. Submodular function maximization via the multilinear relaxation and contention resolution schemes. *SIAM J. Comput.*, 43(6):1831–1879, 2014. 4.9

[CWJ18] Yang Chen, Jie Wu, and Bo Ji. Virtual network function deployment in tree-structured networks. In *2018 IEEE 26th International Conference on Network Protocols, ICNP 2018, Cambridge, UK, September 25-27, 2018*, pages 132–142. IEEE Computer Society, 2018. 1, 1.1.1

[DEH$^+$18] Sina Dehghani, Soheil Ehsani, MohammadTaghi Hajiaghayi, Vahid Liaghat, and Saeed Seddighin. Greedy algorithms for online survivable network design. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages

152:1–152:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 4.1.3

[DHK16] Amol Deshpande, Lisa Hellerstein, and Devorah Kletenik. Approximation algorithms for stochastic submodular set cover with applications to boolean function evaluation and min-knapsack. *ACM Trans. Algorithms*, 12(3):42:1–42:28, 2016. 3.1.2

[DIMV14] Erik D. Demaine, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. On streaming and communication complexity of the set cover problem. In Fabian Kuhn, editor, *Distributed Computing - 28th International Symposium, DISC 2014, Austin, TX, USA, October 12-15, 2014. Proceedings*, volume 8784 of *Lecture Notes in Computer Science*, pages 484–498. Springer, 2014. 4.1.1

[DJK13] Nikhil R. Devanur, Kamal Jain, and Robert D. Kleinberg. Randomized primal-dual analysis of RANKING for online bipartite matching. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 101–107. SIAM, 2013. 7.1.2, 7.2.2, 7.8, 8

[DS14] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In David B. Shmoys, editor, *Symposium on Theory of Computing, STOC 2014, New York, NY, USA, May 31 - June 03, 2014*, pages 624–633. ACM, 2014. 2.4, 2.5, 7.1, 7.1.2, 7.7, 7.7.2, 7.7.2, 7.12

[DSSX19] John P. Dickerson, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu. Balancing relevance and diversity in online bipartite matching via submodularity. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 1877–1884. AAAI Press, 2019. 7.1

[EKM21] Hossein Esfandiari, Amin Karbasi, and Vahab S. Mirrokni. Adaptivity in adaptive submodularity. In Mikhail Belkin and Samory Kpotufe, editors, *Conference on Learning Theory, COLT 2021, 15-19 August 2021, Boulder, Colorado, USA*, volume 134 of *Proceedings of Machine Learning Research*, pages 1823–1846. PMLR, 2021. 3.1.2

[EL14] Leah Epstein and Asaf Levin. Robust algorithms for preemptive scheduling. *Algorithmica*, 69(1):26–57, 2014. 6.1.3

[ELSW18] Leah Epstein, Asaf Levin, Danny Segev, and Oren Weimann. Improved bounds for randomized preemptive online matching. *Inf. Comput.*, 259(1):31–40, 2018. 7.1

[EMR18] Guy Even, Moti Medina, and Dror Rawitz. Online generalized caching with varying weights and costs. In Christian Scheideler and Jeremy T. Fineman, editors, *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018*, pages 205–212. ACM, 2018. 5.1

[EN16] Alina Ene and Huy L. Nguyen. Constrained submodular maximization: Beyond 1/e. In Irit Dinur, editor, *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 248–257. IEEE Computer Society, 2016. 7.1

[ER16] Yuval Emek and Adi Rosén. Semi-streaming set cover. *ACM Trans. Algorithms*, 13(1):6:1–6:22, 2016. 4.1.1

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998. 2.5, 7.1, 7.1.1, 7.1.2, 7.7

[FH05] Stephan Foldes and Peter L. Hammer. Submodularity, supermodularity, and higher-order monotonicities of pseudo-boolean functions. *Math. Oper. Res.*, 30(2):453–461, 2005. 2.2, 2.2.3, 6.1.1, 6.1.2, 6.1.3

[FHTZ20] Matthew Fahrbach, Zhiyi Huang, Runzhou Tao, and Morteza Zadimoghaddam. Edge-weighted online bipartite matching. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 412–423. IEEE, 2020. 7.1.2, 7.2.2

[FKK18] Moran Feldman, Amin Karbasi, and Ehsan Kazemi. Do less, get more: Streaming submodular maximization with subsampling. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 730–740, 2018. 7.1, 7.1.1, 7.1.1, 7.1.2, 7.2, 7.9, 15, 7.9.1, 7.9.2

[FKL$^+$91] Amos Fiat, Richard M. Karp, Michael Luby, Lyle A. McGeoch, Daniel Dominic Sleator, and Neal E. Young. Competitive paging algorithms. *J. Algorithms*, 12(4):685–699, 1991. 5.1

[FKM$^+$05] Joan Feigenbaum, Sampath Kannan, Andrew McGregor, Siddharth Suri, and Jian Zhang. On graph problems in a semi-streaming model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005. 7.1, 5

[FNS11] Moran Feldman, Joseph Naor, and Roy Schwartz. A unified continuous greedy algorithm for submodular maximization. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 570–579. IEEE Computer Society, 2011. 7.1

[FNSW11] Moran Feldman, Joseph Naor, Roy Schwartz, and Justin Ward. Improved approximations for k-exchange systems - (extended abstract). In Camil Demetrescu and Magnús M. Halldórsson, editors, *Algorithms - ESA 2011 - 19th Annual European Symposium, Saarbrücken, Germany, September 5-9, 2011. Proceedings*, volume 6942 of *Lecture Notes in Computer Science*, pages 784–798. Springer, 2011. 7.1

[FNSZ20] Moran Feldman, Ashkan Norouzi-Fard, Ola Svensson, and Rico Zenklusen. The one-way communication complexity of submodular maximization with applications to streaming and robustness. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1363–1374. ACM, 2020. 7.1, 2

[FNW78] Marshall L. Fisher, George L. Nemhauser, and Laurence A. Wolsey. *An analysis of approximations for maximizing submodular set functions—II*, pages 73–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 1978. 2.4, 7.1

[Fuj99] Toshihiro Fujito. On approximation of the submodular set cover problem. *Oper.*

Res. Lett., 25(4):169–174, 1999. 2.4

[Gab18] Harold N. Gabow. Data structures for weighted matching and extensions to *b*-matching and *f*-factors. *ACM Trans. Algorithms*, 14(3):39:1–39:80, 2018. 7.6

[GB11] Andrew Guillory and Jeff A. Bilmes. Online submodular set cover, ranking, and repeated active learning. In John Shawe-Taylor, Richard S. Zemel, Peter L. Bartlett, Fernando C. N. Pereira, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*, pages 1107–1115, 2011. 3.1.2

[GBLV13] Amit Goyal, Francesco Bonchi, Laks V. S. Lakshmanan, and Suresh Venkatasubramanian. On minimizing budget and time in influence propagation over social networks. *Soc. Netw. Anal. Min.*, 3(2):179–192, 2013. 1, 1.1.1

[GGK16] Albert Gu, Anupam Gupta, and Amit Kumar. The power of deferral: Maintaining a constant-competitive steiner tree online. *SIAM J. Comput.*, 45(1):1–28, 2016. 6.1.3, 6.7.1

[GGL$^+$13] Fabrizio Grandoni, Anupam Gupta, Stefano Leonardi, Pauli Miettinen, Piotr Sankowski, and Mohit Singh. Set covering with our eyes closed. *SIAM J. Comput.*, 42(3):808–830, 2013. 4.1.3, 4.11

[GGN21] Rohan Ghuge, Anupam Gupta, and Viswanath Nagarajan. The power of adaptivity for stochastic submodular cover. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 3702–3712. PMLR, 2021. 3.1.2

[GHKL16] Nathaniel Grammel, Lisa Hellerstein, Devorah Kletenik, and Patrick Lin. Scenario submodular cover. In Klaus Jansen and Monaldo Mastrolilli, editors, *Approximation and Online Algorithms - 14th International Workshop, WAOA 2016, Aarhus, Denmark, August 25-26, 2016, Revised Selected Papers*, volume 10138 of *Lecture Notes in Computer Science*, pages 116–128. Springer, 2016. 3.1.2

[GK11] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artif. Intell. Res.*, 42:427–486, 2011. 3.1.2

[GK14] Anupam Gupta and Amit Kumar. Online steiner tree with deletions. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 455–467. SIAM, 2014. 6.1.3, 6.7.2, 6.7.2

[GKK12] Ashish Goel, Michael Kapralov, and Sanjeev Khanna. On the communication and streaming complexity of maximum bipartite matching. In Yuval Rabani, editor, *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2012, Kyoto, Japan, January 17-19, 2012*, pages 468–485. SIAM, 2012. 7.1.2, 7.7, 7.7.4, 7.7, 7.7.5, 7.7.5

[GKKP17] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Online and dynamic algorithms for set cover. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium* *on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 537–550. ACM, 2017. 6.1, 6.1.2, 6.1.3, 6.5, 6.6, 6.8, 7.1.2, 7.2.2, 8

[GKKP19] Anupam Gupta, Ravishankar Krishnaswamy, Amit Kumar, and Debmalya Panigrahi. Elastic caching. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 143–156. SIAM, 2019. 5.1

[GKKV95] Edward F. Grove, Ming-Yang Kao, P. Krishnan, and Jeffrey Scott Vitter. Online perfect matching and mobile computing. In Selim G. Akl, Frank K. H. A. Dehne, Jörg-Rüdiger Sack, and Nicola Santoro, editors, *Algorithms and Data Structures, 4th International Workshop, WADS '95, Kingston, Ontario, Canada, August 16-18, 1995, Proceedings*, volume 955 of *Lecture Notes in Computer Science*, pages 194–205. Springer, 1995. 6.1.3

[GKL21] Anupam Gupta, Gregory Kehne, and Roie Levin. Random order online set cover is as easy as offline. In *62nd IEEE Annual Symposium on Foundations of Computer Science, FOCS 2021, Denver, CO, USA, February 7-10, 2022*, pages 1253–1264. IEEE, 2021. 1.1.2

[GKLX20] Xiangyu Guo, Janardhan Kulkarni, Shi Li, and Jiayi Xian. On the facility location problem in online and dynamic models. In Jaroslaw Byrka and Raghu Meka, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2020, August 17-19, 2020, Virtual Conference*, volume 176 of *LIPIcs*, pages 42:1–42:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. 6.1.3

[GKP20] Anupam Gupta, Amit Kumar, and Debmalya Panigrahi. Caching with time windows. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1125–1138. ACM, 2020. 5.1

[GKS14] Anupam Gupta, Amit Kumar, and Cliff Stein. Maintaining assignments online: Matching, scheduling, and flows. In Chandra Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 468–479. SIAM, 2014. 6.1.3

[GL20a] Anupam Gupta and Roie Levin. Fully-dynamic submodular cover with bounded recourse. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1147–1157. IEEE, 2020. 1.1.4

[GL20b] Anupam Gupta and Roie Levin. The online submodular cover problem. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1525–1537. SIAM, 2020. 1.1.1

[GLS88] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. 2.2

[GM16] Anupam Gupta and Marco Molinaro. How the experts algorithm can help solve lps online. *Math. Oper. Res.*, 41(4):1404–1431, 2016. 4.1.3

[GN14] Anupam Gupta and Viswanath Nagarajan. Approximating sparse covering integer programs online. *Math. Oper. Res.*, 39(4):998–1011, 2014. 2.5

[GP13] Manoj Gupta and Richard Peng. Fully dynamic (1+ e)-approximate matchings. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 548–557. IEEE Computer Society, 2013. 7.8

[GRST10] Anupam Gupta, Aaron Roth, Grant Schoenebeck, and Kunal Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In Amin Saberi, editor, *Internet and Network Economics - 6th International Workshop, WINE 2010, Stanford, CA, USA, December 13-17, 2010. Proceedings*, volume 6484 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2010. 7.1

[GS20] Anupam Gupta and Sahil Singla. Random-order models. In Tim Roughgarden, editor, *Beyond the Worst-Case Analysis of Algorithms*, pages 234–258. Cambridge University Press, 2020. 4.1.3, 4.11

[GTW14] Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, volume 8572 of *Lecture Notes in Computer Science*, pages 563–575. Springer, 2014. 3.1.2

[GV11] Shayan Oveis Gharan and Jan Vondrák. Submodular maximization by simulated annealing. In Dana Randall, editor, *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 1098–1116. SIAM, 2011. 7.1

[GW19] Mohsen Ghaffari and David Wajc. Simplified and space-optimal semi-streaming (2+epsilon)-approximate matching. In Jeremy T. Fineman and Michael Mitzenmacher, editors, *2nd Symposium on Simplicity in Algorithms, SOSA 2019, January 8-9, 2019, San Diego, CA, USA*, volume 69 of *OASIcs*, pages 13:1–13:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. 7.1, 7.1.2, 7.6

[HGDS14] Syed Hasan, Sergey Gorinsky, Constantine Dovrolis, and Ramesh K. Sitaraman. Trade-offs in optimizing the cache deployments of cdns. In *2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, Canada, April 27 - May 2, 2014*, pages 460–468. IEEE, 2014. 5.1

[HIMV16] Sariel Har-Peled, Piotr Indyk, Sepideh Mahabadi, and Ali Vakilian. Towards tight bounds for the streaming set cover problem. In Tova Milo and Wang-Chiew Tan, editors, *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, pages 371–383. ACM, 2016. 4.1.1

[HK18] Lisa Hellerstein and Devorah Kletenik. Revisiting the approximation bound for stochastic submodular cover. *J. Artif. Intell. Res.*, 63:265–279, 2018. 3.1.2

[HKT$^+$18] Zhiyi Huang, Ning Kang, Zhihao Gavin Tang, Xiaowei Wu, Yuhao Zhang, and Xue Zhu. How to match when all vertices arrive online. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 17–29. ACM, 2018. 7.1.2, 7.2.2

[HNO08] Nicholas J. A. Harvey, Jelani Nelson, and Krzysztof Onak. Sketching and streaming entropy via approximation theory. In *49th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2008, October 25-28, 2008, Philadelphia, PA, USA*, pages 489–498. IEEE Computer Society, 2008. 6.1.2

[Hoc82] Dorit S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982. 2.4

[HPT$^+$19] Zhiyi Huang, Binghui Peng, Zhihao Gavin Tang, Runzhou Tao, Xiaowei Wu, and Yuhao Zhang. Tight competitive ratios of classic matching algorithms in the fully online model. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2875–2886. SIAM, 2019. 7.1.2, 7.2.2

[HTWZ19] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Online vertex-weighted bipartite matching: Beating 1-1/e with random arrivals. *ACM Trans. Algorithms*, 15(3):38:1–38:15, 2019. 7.1.2, 7.2.2

[HTWZ20] Zhiyi Huang, Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Fully online matching II: beating ranking and water-filling. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1380–1391. IEEE, 2020. 7.1.2, 7.2.2

[HZ20] Zhiyi Huang and Qiankun Zhang. Online primal dual meets online matching with stochastic rewards: configuration LP to the rescue. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1153–1164. ACM, 2020. 7.1.2, 7.2.2

[HZZ20] Zhiyi Huang, Qiankun Zhang, and Yuhao Zhang. Adwords in a panorama. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 1416–1426. IEEE, 2020. 7.1.2, 7.2.2

[IIOW10] Tomoko Izumi, Taisuke Izumi, Hirotaka Ono, and Koichi Wada. Approximability and inapproximability of the minimum certificate dispersal problem. *Theor. Comput. Sci.*, 411(31-33):2773–2783, 2010. 1, 1.1.1

[IKBA21] Rishabh K. Iyer, Ninad Khargoankar, Jeff A. Bilmes, and Himanshu Asanani. Submodular combinatorial information measures with applications in machine learning. In Vitaly Feldman, Katrina Ligett, and Sivan Sabato, editors, *Algorithmic Learning Theory, 16-19 March 2021, Virtual Conference, Worldwide*, volume 132 of *Proceedings of Machine Learning Research*, pages 722–754. PMLR, 2021. 2.3, 6.1.3

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001. 7.7

[Ira96] Sandy Irani. Competitive analysis of paging: A survey. In *Proc. of the Dagstuhl*

Seminar on Online Algorithms, 1996. 5.1

[Ira02a] Sandy Irani. Page replacement with multi-size pages and applications to web caching. *Algorithmica*, 33(3):384–409, 2002. 5.1

[Ira02b] Sandy Irani. Randomized weighted caching with two page weights. *Algorithmica*, 32(4):624–640, 2002. 5.1

[IW91] Makoto Imase and Bernard M. Waxman. Dynamic steiner tree problem. *SIAM J. Discret. Math.*, 4(3):369–384, 1991. 6.1.3, 6.7.2, 6.7.3

[Jak05] Aleks Jakulin. Machine learning based on attribute interactions. *PhD thesis, University of Ljubljana, Ljubljana, Slovenia*, 2005. 6.1.2

[JCMP17] Stefan Jorgensen, Robert H. Chen, Mark B. Milam, and Marco Pavone. The risk-sensitive coverage problem: Multi-robot routing under uncertainty with service level and survival constraints. In *56th IEEE Annual Conference on Decision and Control, CDC 2017, Melbourne, Australia, December 12-15, 2017*, pages 925–932. IEEE, 2017. 1.1.1

[Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974. 2.5

[Kap13] Michael Kapralov. Better bounds for matchings in the streaming model. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 1679–1697. SIAM, 2013. 7.1, 7.1.1, 7.7

[KMGG07] Andreas Krause, H. Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Selecting observations against adversarial objectives. In John C. Platt, Daphne Koller, Yoram Singer, and Sam T. Roweis, editors, *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 3-6, 2007*, pages 777–784. Curran Associates, Inc., 2007. 1.1.1

[KMZ18] Nitish Korula, Vahab S. Mirrokni, and Morteza Zadimoghaddam. Online submodular welfare maximization: Greedy beats 1/2 in random order. *SIAM J. Comput.*, 47(3):1056–1086, 2018. 7.1

[KMZ$^+$19] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3311–3320. PMLR, 2019. 7.1

[KN17] Guy Kortsarz and Zeev Nutov. Approximating source location and star survivable network problems. *Theor. Comput. Sci.*, 674:32–42, 2017. 1, 1.1.1

[KNR20] Haim Kaplan, David Naori, and Danny Raz. Competitive analysis with a sample and the secretary problem. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2082–2095. SIAM, 2020. 1

[KNR22] Haim Kaplan, David Naori, and Danny Raz. Online weighted matching with a sample. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 1247–1272. SIAM, 2022. 1

[Kor04] Simon Korman. On the use of randomization in the online set cover problem. *Master's thesis, Weizmann Institute of Science, Rehovot, Israel*, 2004. 1.1.4, 2.5, 1, 4.1.2, 4.1.3, 4.7.3, 4.11

[KRTV18] Thomas Kesselheim, Klaus Radke, Andreas Tönnis, and Berthold Vöcking. Primal beats dual on online packing lps in the random-order model. *SIAM J. Comput.*, 47(5):1939–1964, 2018. 4.1.3

[LB11] Hui Lin and Jeff A. Bilmes. Word alignment via submodular maximization over matroids. In *The 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference, 19-24 June, 2011, Portland, Oregon, USA - Short Papers*, pages 170–175. The Association for Computer Linguistics, 2011. 1, 7.1

[LCL$^+$16] Zhipeng Liu, Andrew Clark, Phillip Lee, Linda Bushnell, Daniel S. Kirschen, and Radha Poovendran. Mingen: Minimal generator set selection for small signal stability in power systems: A submodular framework. In *55th IEEE Conference on Decision and Control, CDC 2016, Las Vegas, NV, USA, December 12-14, 2016*, pages 4122–4129. IEEE, 2016. 1.1.1

[LG16] Grigorios Loukides and Robert Gwadera. Limiting the diffusion of information by a selective pagerank-preserving approach. In *2016 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2016, Montreal, QC, Canada, October 17-19, 2016*, pages 90–99. IEEE, 2016. 1, 1.1.1

[LLN06] Benny Lehmann, Daniel Lehmann, and Noam Nisan. Combinatorial auctions with decreasing marginal utilities. *Games Econ. Behav.*, 55(2):270–296, 2006. 1, 7.1

[LMNS09] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 323–332. ACM, 2009. 7.1

[LMNS10] Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discret. Math.*, 23(4):2053–2078, 2010. 7.1

[LOP$^+$15] Jakub Lacki, Jakub Ocwieja, Marcin Pilipczuk, Piotr Sankowski, and Anna Zych. The power of dynamic distance oracles: Efficient dynamic algorithms for the steiner tree. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 11–20. ACM, 2015. 6.1.3, 6.7.2

[Lov75] László Lovász. On the ratio of optimal integral and fractional covers. *Discret. Math.*, 13(4):383–390, 1975. 2.5

[LPS13] Seungjoon Lee, Manish Purohit, and Barna Saha. Firewall placement in cloud data centers. In Guy M. Lohman, editor, *ACM Symposium on Cloud Computing, SOCC*

'13, Santa Clara, CA, USA, October 1-3, 2013, pages 52:1–52:2. ACM, 2013. 1, 1.1.1

[LRS18] Tamás Lukovszki, Matthias Rost, and Stefan Schmid. Approximate and incremental network function placement. *J. Parallel Distributed Comput.*, 120:159–169, 2018. 1, 1.1.1

[LSV10] Jon Lee, Maxim Sviridenko, and Jan Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. Oper. Res.*, 35(4):795–806, 2010. 7.1

[LV21] Thodoris Lykouris and Sergei Vassilvitskii. Competitive caching with machine learned advice. *J. ACM*, 68(4):24:1–24:25, 2021. 5.1

[LW21] Roie Levin and David Wajc. Streaming submodular matching meets the primal-dual method. In Dániel Marx, editor, *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 1914–1933. SIAM, 2021. 1.1.5

[Man20] Pasin Manurangsi. Tight running time lower bounds for strong inapproximability of maximum k-coverage, unique set cover and related problems (via t-wise agreement testing theorem). In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 62–81. SIAM, 2020. 7.1, 7.7

[McG05] Andrew McGregor. Finding graph matchings in data streams. In Chandra Chekuri, Klaus Jansen, José D. P. Rolim, and Luca Trevisan, editors, *Approximation, Randomization and Combinatorial Optimization, Algorithms and Techniques, 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX 2005 and 9th InternationalWorkshop on Randomization and Computation, RANDOM 2005, Berkeley, CA, USA, August 22-24, 2005, Proceedings*, volume 3624 of *Lecture Notes in Computer Science*, pages 170–181. Springer, 2005. 7.1, 7.9, 7.9.1, 5

[Mey01] Adam Meyerson. Online facility location. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 426–431. IEEE Computer Society, 2001. 4.1.3

[MJK18] Baharan Mirzasoleiman, Stefanie Jegelka, and Andreas Krause. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 1379–1386. AAAI Press, 2018. 7.1

[MKSK13] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages

2049–2057, 2013. 1

[MMP01] Adam Meyerson, Kamesh Munagala, and Serge A. Plotkin. Designing networks incrementally. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 406–415. IEEE Computer Society, 2001. 4.1.3

[Mol17] Marco Molinaro. Online and random-order load balancing simultaneously. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1638–1650. SIAM, 2017. 4.1.3

[MR14] Marco Molinaro and R. Ravi. The geometry of online packing linear programs. *Math. Oper. Res.*, 39(1):46–59, 2014. 4.1.3

[MS91] Lyle A. McGeoch and Daniel Dominic Sleator. A strongly competitive randomized paging algorithm. *Algorithmica*, 6(6):816–825, 1991. 5.1

[MSM] Bertinoro workshop 2014, problem 63. `https://sublinear.info/index.php?title=Open_Problems:63`. Accessed: 2020-06-20. 7.1.1

[MW16] Armeline Dembo Mafuta and Tom Walingo. Spatial relay node placement in wireless sensor networks. In *IEEE 83rd Vehicular Technology Conference, VTC Spring 2016, Nanjing, China, May 15-18, 2016*, pages 1–5. IEEE, 2016. 1, 1.1.1

[NTM$^+$18] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. Beyond 1/2-approximation for submodular maximization on massive data streams. In Jennifer G. Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 3826–3835. PMLR, 2018. 7.1, 2

[NW78] George L. Nemhauser and Laurence A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978. 7.1, 7.1.1

[NWF78] George L. Nemhauser, Laurence A. Wolsey, and Marshall L. Fisher. An analysis of approximations for maximizing submodular set functions - I. *Math. Program.*, 14(1):265–294, 1978. 2.4, 7.1

[Pit85] Leonard Brian Pitt. *A simple probabilistic approximation algorithm for vertex cover*. Yale University, Department of Computer Science, 1985. 6.5

[PS16] David Peleg and Shay Solomon. Dynamic $(1+\epsilon)$-approximate matchings: A density-sensitive approach. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 712–729. SIAM, 2016. 7.8

[PS19] Ami Paz and Gregory Schwartzman. A $(2+\epsilon)$-approximation for maximum weight matching in the semi-streaming model. *ACM Trans. Algorithms*, 15(2):18:1–18:15, 2019. 7.1, 7.1.1, 7.1.2, 3, 7.6

[PW93] Steven J. Phillips and Jeffery R. Westbrook. Online load balancing and network flow. In S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors, *Proceedings* *of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, pages 402–411. ACM, 1993. 6.1.3

[RP15] Mohammad Amin Rahimian and Victor M. Preciado. Detection and isolation of failures in directed networks of LTI systems. *IEEE Trans. Control. Netw. Syst.*, 2(2):183–192, 2015. 1, 1.1.1

[RS97] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In Frank Thomson Leighton and Peter W. Shor, editors, *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 475–484. ACM, 1997. 4.7.3

[SBLL20] Zhenyu Song, Daniel S. Berger, Kai Li, and Wyatt Lloyd. Learning relaxed belady for content distribution network caching. In Ranjita Bhagwan and George Porter, editors, *17th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2020, Santa Clara, CA, USA, February 25-27, 2020*, pages 529–544. USENIX Association, 2020. 5.1

[Sel20] Mark Sellke. Chasing convex bodies optimally. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 1509–1518. SIAM, 2020. 6.1.3

[SG08] Matthew J. Streeter and Daniel Golovin. An online algorithm for maximizing submodular functions. In Daphne Koller, Dale Schuurmans, Yoshua Bengio, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1577–1584. Curran Associates, Inc., 2008. 3.1.2

[SG09] Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2009, April 30 - May 2, 2009, Sparks, Nevada, USA*, pages 697–708. SIAM, 2009. 4.1.1

[SS14] Yevgeny Seldin and Aleksandrs Slivkins. One practical algorithm for both stochastic and adversarial bandits. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 1287–1295. JMLR.org, 2014. 6.1.2

[SSS09] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009. 6.1.3

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28(2):202–208, 1985. 5.1

[SV10] Martin Skutella and José Verschae. A robust PTAS for machine covering and packing. In Mark de Berg and Ulrich Meyer, editors, *Algorithms - ESA 2010, 18th Annual European Symposium, Liverpool, UK, September 6-8, 2010. Proceedings, Part I*, volume 6346 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2010. 6.1.3

[Svi04] Maxim Sviridenko. A note on maximizing a submodular set function subject to a knapsack constraint. *Oper. Res. Lett.*, 32(1):41–43, 2004. 2.4

[TRPJ16] Vasileios Tzoumas, Mohammad Amin Rahimian, George J. Pappas, and Ali Jadbabaie. Minimal actuator placement with bounds on control effort. *IEEE Trans. Control. Netw. Syst.*, 3(1):67–78, 2016. 1.1.1

[Tsa88] Constantino Tsallis. Possible generalization of Boltzmann-Gibbs statistics. *Journal of statistical physics*, 52(1-2):479–487, 1988. 6.1.2

[TWPD17] Guangmo Tong, Weili Wu, Panos M. Pardalos, and Ding-Zhu Du. On positive-influence target-domination. *Optim. Lett.*, 11(2):419–427, 2017. 1, 1.1.1

[TWZ20] Zhihao Gavin Tang, Xiaowei Wu, and Yuhao Zhang. Towards a better understanding of randomized greedy matching. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 1097–1110. ACM, 2020. 7.1.2, 7.2.2

[Von07] Jan Vondrák. Submodularity in combinatorial optimization. *PhD thesis, Charles University, Prague, Czech Republic*, 2007. 2.2, 2.2, 3.1.1, 3.2.2, 3.2.2, 3.4, 3.4.1

[Von08] Jan Vondrák. Optimal approximation for the submodular welfare problem in the value oracle model. In Cynthia Dwork, editor, *Proceedings of the 40th Annual ACM Symposium on Theory of Computing, Victoria, British Columbia, Canada, May 17-20, 2008*, pages 67–74. ACM, 2008. 1, 7.1

[Von13] Jan Vondrák. Symmetry and approximability of submodular maximization problems. *SIAM J. Comput.*, 42(1):265–304, 2013. 7.1

[Waj20] David Wajc. Rounding dynamic matchings against an adaptive adversary. In Konstantin Makarychev, Yury Makarychev, Madhur Tulsiani, Gautam Kamath, and Julia Chuzhoy, editors, *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 194–207. ACM, 2020. 7.8

[WCZW15] Pan Wu, Guihai Chen, Xiaojun Zhu, and Xiaobing Wu. Minimizing receivers under link coverage model for device-free surveillance. *Comput. Commun.*, 63:53–64, 2015. 1, 1.1.1

[Wes00] Jeffery R. Westbrook. Load balancing for response time. *J. Algorithms*, 35(1):1–16, 2000. 6.1.3

[WMWP15] Zengfu Wang, William Moran, Xuezhi Wang, and Quan Pan. An accelerated continuous greedy algorithm for maximizing strong submodular functions. *J. Comb. Optim.*, 30(4):1107–1124, 2015. 6.1.3

[Wol82] Laurence A. Wolsey. An analysis of the greedy algorithm for the submodular set covering problem. *Comb.*, 2(4):385–393, 1982. 1, 2.4, 2.4.1, 2.4, 3.1.1, 6.2.2, 6.3.2

[WS11] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2011. 2.5

[WWLP13] Zengfu Wang, Xuezhi Wang, Yan Liang, and Quan Pan. Weapon target assignment leveraging strong submodularity. In *IEEE International Conference on Information and Automation, ICIA 2013, Yinchuan, China, August 26-28, 2013*, pages 74–79. IEEE, 2013. 6.1.3

[Yao77] Andrew Chi-Chih Yao. Probabilistic computations: Toward a unified measure of complexity (extended abstract). In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 222–227. IEEE Computer Society, 1977. 7.7

[YCDW15] Ying Yang, Li Chen, Wenxiang Dong, and Weidong Wang. Active base station set optimization for minimal energy consumption in green cellular networks. *IEEE Trans. Veh. Technol.*, 64(11):5340–5349, 2015. 1.1.1

[You91] Neal E. Young. On-line caching as cache size varies. In Alok Aggarwal, editor, *Proceedings of the Second Annual ACM/SIGACT-SIAM Symposium on Discrete Algorithms, 28-30 January 1991, San Francisco, California, USA*, pages 241–250. ACM/SIAM, 1991. 5.1

[You94] Neal E. Young. The k-server dual and loose competitiveness for paging. *Algorithmica*, 11(6):525–541, 1994. 5.1

[You02] Neal E. Young. On-line file caching. *Algorithmica*, 33(3):371–383, 2002. 5.1

[ZPW$^+$17] Zhenzhe Zheng, Yanqing Peng, Fan Wu, Shaojie Tang, and Guihai Chen. Trading data in the crowd: Profit-driven data acquisition for mobile crowdsensing. *IEEE J. Sel. Areas Commun.*, 35(2):486–501, 2017. 1, 1.1.1