University of Padova

Department of Mathematics "Tullio Levi-Civita"
Master's Degree in Computer Science

Advances in Submodularity: Optimization and Probabilistic Models

Internal Supervisor
Prof. Luigi De Giovanni
University of Padova

Candidate
Alberto Schiabel
1236598

External Co-supervisors
Dr. Jakub Mareček, Ph.D.
Czech Technical University in Prague

Dr. Vyacheslav Kungurtsev, Ph.D.
Czech Technical University in Prague

December 2021

“Optimization is an elegant mathematical theory which provides fundamental tools for reasoning about and solving problems across a broad range of areas in the data sciences.”

— **Yaron Singer**
Professor of Computer Science at Harvard University

Abstract

Submodularity plays an important role in theoretical computer science for mathematical optimization. The structure underneath submodular functions is crucial for the design of efficient optimization algorithms with approximation guarantees. Submodular objectives can be optimized using either discrete algorithms or continuous extensions, and naturally emerge in combinatorial optimization and machine learning scenarios.

Even though research in submodularity is mostly tied to set functions, submodularity on bounded integer lattices (multisets) allows for more expressive and relevant problem settings, but it also presents more challenges, as existing algorithms for set submodular functions cannot be applied directly in this case.

In this thesis, we present a novel family of randomized algorithms for maximizing monotone submodular functions on the integer lattice subject to a cardinality constraint, surpassing the overall performance of the state of the art in many instances.

Going beyond optimization, we also sketch a novel idea for Markov chain Monte Carlo sampler for probabilistic submodular models that uses the Lovász extension.

Contents

List of Figures

1 Introduction

This thesis investigates efficient solutions for discrete problems involving the notion of *submodularity*, either explicitly or implicitly. Most of the literature on submodular functions considers algorithms to solve problems defined on set functions, which are functions defined on subsets $S \subseteq \mathcal{V}$, where $\mathcal{V}$ is a given ground set. In the past 50 years, set-submodularity has emerged as a powerful tool in combinatorial optimization, with many practical applications in operations research, economics, machine learning, and computer vision. However, submodularity can also be defined on other domains, like the integer lattice, where it becomes a property of functions defined on multisets, allowing for more expressive (but also generally more difficult to solve) mathematical models.

Similarly to convex functions, submodular functions can be minimized exactly in polynomial time, assuming the problem to solve is unconstrained. However, unlike convex functions, submodular functions can also be maximized with approximation guarantees in polynomial time, even when the target problem is subject to matroid constraints or backpack constraints. The monotonicity property of submodular functions usually allows for more efficient algorithms to be implemented. In this work, we present two main novel contributions.

First, we introduce a new approach to maximize monotone submodular functions defined on the (bounded) integer lattice subject to a cardinality constraint. Building upon a recent STOCHASTIC-GREEDY algorithm proposed in Mirzasoleiman et al. (2015), we show that our algorithms are more efficient than the previous state of the art for the target problem, while retaining the comparable approximation guarantees. We also validate our novel theoretical results with practical experiments on both synthetic and realistic datasets, focusing on standard objective functions such as the *optimal budget allocation* problem.

Besides optimization, submodularity has also very recently been considered and studied in probabilistic settings, where a set submodular function is used to model the distribution of a binary random variable $X \in \{0, 1\}$. In the second part of this thesis, we review the current state of the art for the emerging field of *probabilistic submodular models* and the most important aspects of Markov Chain Monte Carlo sampling in the context of Bayesian inference. We then introduce a novel MCMC sampling approach for inferring the posterior distribution of a probabilistic submodular model using the Lovász extension and the subgradient projection method.

1.1 Original Seminal Papers

The following seminal papers has been submitted while working on this thesis:

- Alberto Schiabel, Vyacheslav Kungurtsev, and Jakub Jakub Mareček (2022). "Randomized Algorithms for Monotone Submodular Function Maximization on the Integer Lattice". In: *Proceedings of the 23rd Conference on Integer Programming and Combinatorial Optimization* (IPCO-22).

We expect to receive a notification of acceptance by January 2022.

1.2 Research Questions of Interest

In this work, we mainly focus on the following two research questions:

1. Given a submodular function of interested defined over a multiset of elements, how can we efficiently maximize it if the problem is subject to a cardinality constraint? Can we beat the previous state-of-the-art results in terms of either runtime or approximation ratios?
2. Can we design a *Markov chain Monte Carlo* sampler for a probabilistic submodular model of interest (#P-Hard to evaluate exactly in general) that converges to the true distribution faster than the well-known Gibbs and Metropolis samplers?

1.3 Thesis Topics & Contributions

In this thesis, we investigate the role of submodularity in real-world optimization problems and in probablistic inference settings, constrained or otherwise, which results in the following contributions:

- We present a comprehensive summary of the state-of-the-art in the field of submodular optimization, from the more common notion of discrete set-submodularity to the more general concepts of continuous submodularity and submodularity on the integer lattice.
- We study representative problems that originated in the set-submodular literature and generalize them to a multiset domain, discussing the improvements that this generalization offers.
- We provide a comparison of several existing and novel algorithms for maximizing a submodular function on an integer lattice subject to cardinality constraints, indicating tradeoffs in regard to runtime and final objective value found w.r.t. the size of the ground set, and other quantitative and qualitative parameters of the problem. We provide approximation-ratio and some runtime guarantees, and we observe that some of our algorithms are faster than the state-of-the-art for the problem under consideration.
- We review the recent studies on probabilistic submodular models, and we experiment with a novel Markov Chain Monte Carlo sampler for Bayesian inference tasks on such models.

1.3.1 Thesis Structure

The chapters of the thesis are organized as follows.

Chapter 2 We present the notations, preliminary definitions, and some background on submodular functions.

Chapter 3 We review the most important contributions in set submodular optimization, and we formalize the most common constrained variants of submodular optimization problems.

Chapter 4 We first give a thorough characterization of integer-lattice submodular functions in the constrained maximization setting. We then present a novel family of algorithms and related experiments for the maximization of integer-lattice submodular functions subject to a cardinality constraint.

Chapter 5 We review probabilistic submodular models and their applications in the context of Bayesian inference.

Chapter 6 We sketch a novel Markov chain Monte Carlo sampler for higher-order probabilistic submodular models.

Chapter 7 We discuss potential future directions of the work, and we conclude the thesis.

1.4 External Supervisors

This Master's thesis is the result of a joint research work with Prof. Jakub Mareček and Dr. Vyacheslav Kungurtsev, Ph.D. from the Czech Technical University in Prague. I have virtually met Vyacheslav Kungurtsev in December 2020 at an internship proposal event organized by the Career Service and the Department of Mathematics of the University of Padova. He then introduced me to Prof. Jakub Mareček, who is also interested in the topic of submodularity.

We worked together from March 2021 to November 2021, with weekly virtual meetings, reviews, and thousands of emails. They have introduced me to the concept of submodular optimization, and they have incentivated me to review hundreds of existing papers and books on this and related topics. Together, we tried to solve submodular problems with many novel approaches. This thesis only contains the most relevant subset of the work that we have done together, since we have run into some dead ends in the first months of our research. I am grateful for their time, patience, and dedication.

1.5 Introduction to Approximation Algorithms

Given that most of the results in submodular optimization and in our novel contributions are expressed in terms of approximation algorithms, it is worth formalizing what approximation algorithms are before proceeding further.

Definition 1.1 (α-approximation maximization algorithm)**.** *An α-approximation algorithm for an optimization problem is an algorithm A that, for all instances I of a target problem $\mathcal{P}$, yields a solution whose value is within a factor α of $OPT(I)$, where $OPT(I)$ is the optimal value of I. α is* *known as* performance guarantee *of the algorithm, but it is also called as* approximation ratio *or* approximation factor *of the algorithm. We assume $\alpha \geq 1$ for minimization problems, and $\alpha \leq 1$ for maximization problems. For instance, a $(\frac{1}{2})$-approximation algorithm for a maximization problem $\mathcal{P}$ is an algorithm that always produces a solution whose value is at least half of the optimal value $OPT(I)$, for all instances I of $\mathcal{P}$.*

We highlight that studying the approximation ratio of an algorithm is useful for a number of reasons:

- We need algorithms to get solutions to optimization problems (most of which are NP-Hard): *sometimes, near-optimal solutions can be found in polynomial time*;
- Often, approximation algorithms devise a heuristic that *simplifies the model requirements and works well in practice*;
- Approximation algorithms provide a *mathematically rigorous metric to classify and compare heuristics*, which are usually only studied empirically.

We highlight a simple remark that will be useful in the experimental evaluation in the following chapters.

Remark. *When an approximation algorithm returns a solution, it implicitly discovers a bound on the exact optimal value for the instance of the problem it was run against. This is useful when computing the optimal value is too expensive.*

Some optimization problems allow for very good approximation algorithms that allow to finetune the approximation ratio. These problems have so-called *polymial-time approximation schemes.*

Definition 1.2 (PTAS)**.** *A* polynomial-time approximation scheme *(PTAS) is a family of algorithms $\{A_\varepsilon\}$ for which an algorithm exists for each $\varepsilon > 0$, such that A_ε is a $(1+\varepsilon)$-approximation algorithm (for minimization), or a $(1-\varepsilon)$-approximation algorithm (for maximization).*

Notice, however, that in a PTAS, the runtime of an algorithm A_ε is allowed to depend arbitrarily on the value of $\frac{1}{\varepsilon}$. For instance, this dependence could be exponential, or even worse. For practical reasons, researchers are more interested in algorithms with a good bound on the dependence of the runtime of A_ε on $\frac{1}{\varepsilon}$. This prompts the following definition.

Definition 1.3 (FPTAS)**.** *A* fully polynomial-time approximation scheme *(FPTAS) is PTAS such that the runtime of A_ε is polynomially-bounded by $\frac{1}{\varepsilon}$.*

Many combinatorial problems have polynomial-time approximation schemes, such as the Euclidean Traveling Salesman Problem (ETSP). However, there is also a class of interesting problems that have no polynomial-time approximation schemes, but are approximable nonetheless: MAX SNP. For such problems, the best one can hope for is at least an approximation algorithm with constant performance guarantee.

Definition 1.4 (MAX SNP)**.** MAX SNP *is the class of problems that have constant-factor approximation algorithms, but no polynomial-time approximation schemes unless P=NP.*

For instance, the MAXCUT problem and the k-clustering problem belong to MAX SNP. For further details on approximation algorithms, we refer to Atallah & Blanton (2009); Williamson & Shmoys (2011).

2 Submodularity and Submodular Functions

In this chapter, we study some fundamental properties and results about a special class of objects named *submodular functions*, and their relation with mathematical optimization. In particular, we focus on set submodular functions. Let us first provide some basic notation that will be used throughout the thesis.

Notation We consider a finite n-dimensional set $\mathcal{V} := \{1, 2, \ldots, n\}$, which we refer to as *ground set*, and its power set $2^{\mathcal{V}}$, composed of the 2^n subsets of $\mathcal{V}$. Occasionally, we may also use $\{0, 1\}^{\mathcal{V}}$ with the same meaning of $2^{\mathcal{V}}$. We usually refer to the elements of $\mathcal{V}$ as $e \in \mathcal{V}$, and to subsets of $\mathcal{V}$ as A, B, or S (occasionally with subscripts, such as S^j for some j).
We use the standard set notation extensively. Given two sets A and B, $A \cup B$ denotes their union, $A \cap B$ their disjunction, and $A \setminus B$ their difference. $A \subseteq B$ indicates that A is a subset of B, and that potentially A may be equal to B. $|A|$ denotes the cardinality of the set A.
We denote by $\mathbb{R}_+$ the set of non-negative real numbers, and by $\mathbb{Z}_+$ the set of non-negative integer numbers. We use bold face letters such as $\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}}$ and $\boldsymbol{x} \in \mathbb{R}^n$ interchangeably to denote n-dimensional vectors. Similarly, we let $\mathbf{0}$ and $\mathbf{1}$ indicate n-dimensional vectors whose values are all 0 or 1, respectively. Given a vector $\boldsymbol{x}$, we denote its e-th coordinate by x_e or $\boldsymbol{x}(e)$. Given a set $S \subseteq \mathcal{V}$, $\mathbf{1}_S \in \mathbb{Z}_+^{\mathcal{V}}$ is the *indicator vector* of S, i.e., $\mathbf{1}_S(e)$ is 1 if $e \in S$, and 0 otherwise, for all $e \in \mathcal{V}$. We let $\mathbf{1}_e \in \mathbb{Z}_+$ indicate the *characteristic vector*, defined such that $\mathbf{1}_e(e) := 1$ and $\mathbf{1}_e(e') := 0$ for all $e, e' \in \mathcal{V}$ such that $e \neq e'$.
We define the *support* of $\boldsymbol{x} \in \mathbb{Z}_+^n$ as $supp(\boldsymbol{x}) := \{e \in \mathcal{V} \mid x_e > 0\}$. For two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}_+^n$, $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$ means $x_e \leq y_e$ for every element $e \in \mathcal{V}$. Additionally, given $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_+^n$ we let $\boldsymbol{x} \wedge \boldsymbol{y}$ and $\boldsymbol{x} \vee \boldsymbol{y}$ denote the coordinate-wise minimum and maximum, respectively, i.e., $(\boldsymbol{x} \wedge \boldsymbol{y})_e := \min\{x_e, y_e\}$ and $(\boldsymbol{x} \vee \boldsymbol{y})_e := \max\{x_e, y_e\}$. We also adopt the standard l_q-norm notation. For $q \in [1, +\infty]$, we denote by $\|\boldsymbol{x}\|_q$ the l_q-norm of $\boldsymbol{x}$. We define $\|\boldsymbol{x}\|_q := (\sum_{e \in \mathcal{V}} |e_k|)$ for $q \in [1, +\infty)$, and $\|\boldsymbol{x}\|_\infty := max_{e \in \mathcal{V}} |x_e|$. $\|\cdot\|$ means the Euclidean norm (l_2) by default. By default, $f(\cdot)$ is used to denote a discrete function, and $F(\cdot)$ to represent a continuous function. For a differentiable function $F(\cdot)$, we let $\nabla F(\cdot)$ denote its gradient.

Optimization problems with submodular objectives have piqued the interest of the optimization community, as submodular functions find many applications in real-world problems. In set submodular problems, the goal is to pick a subset of a ground set $\mathcal{V}$ that either maximizes or minimizes a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ defined on the powerset of $\mathcal{V}$. A set-valued function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is said to be *submodular* if

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

for all $A, B \subseteq \mathcal{V}$. One can interpret $f(S)$ as the value the function assumes when considering a subset $S \subseteq \mathcal{V}$, with additional items adding less and less value to f, as the size of the subset S increases.

Several operations research problems have an underlying submodular structure, for example optimal budget allocation (Alon et al., 2012), facility location (Cornuejols et al., 1977), budgeted maximum coverage (Khuller et al., 1999), and sensor placement (Frieze, 1974; Krause et al., 2006). Other notable examples in the machine learning area include experiment design (Agrawal et al., 2019; Sahin et al., 2020b), variable selection (Krause & Guestrin, 2005), dictionary learning (Krause & Cevher, 2010; Das & Kempe, 2011), and sparsity inducing regularizers (Bach, 2010). Submodular objectives are also motivated by economics where problems with *diminishing returns* and *decreasing marginal values* are frequently considered (Vondrák, 2007).

While notions such as diversity, coverage, or representativeness are naturally captured using submodular function objectives, supermodular functions (the dual of submodular functions) are used to model cooperation, smoothness, or regularity. From a mathematical optimization perspective, the problem of maximizing a submodular function is NP-hard, as it generalizes notoriously hard problems like Max Cut. Perhaps surprisingly, however, minimizing a submodular function (without any constraints) is known to be solvable in polynomial time. Before we present a brief summary of notable submodular optimization algorithms, we first formalize some definitions that will serve for the rest of the thesis. For an in-depth discussion of the history of submodular functions and submodularity theory, we refer to Fujishige, 2005; Narayanan, 1997; Vondrák, 2007.

2.1 Preliminaries on Submodularity

We consider *set functions* as functions of the form $f : 2^{\mathcal{V}} \to \mathbb{R}$ that map each subset of the ground set $\mathcal{V}$ to a real value. We will often restrict ourselves to set functions that obey to all or some of the following self-explanatory properties:

- f is *non-negative* if $f(S) \geq 0$ for every subset $S \subseteq \mathcal{V}$.
- f is *normalized* if $f(\emptyset) = 0$.
- f is *monotone* if $f(A) \leq f(B)$ for every two subsets $A \subseteq B \subseteq \mathcal{V}$.
- f is *symmetric* if $f(S) = f(\mathcal{V} \setminus S)$ for every subset $S \subseteq \mathcal{V}$.

Adding an element $e \in \mathcal{V}$ to a set $S \subseteq \mathcal{V}$ causes a difference in the value of the function f that is usually referred to as *marginal gain*.

Definition 2.1 (Marginal gain). *For any $e \in \mathcal{V}$, and $S \subseteq \mathcal{V}$, the marginal gain obtained by adding e to S is defined as*

$$f(e \,|\, S) := f(S \cup \{e\}) - f(S). \tag{2.1}$$

Set Submodularity is usually defined in two ways. The most intuitive definition expresses the *diminishing returns* (DR) property: adding one element u to a large set provides less benefit than adding u to a small set. Objectives that we often wish to optimize, such as coverage, diversity, or entropy, exhibit this property.

Definition 2.2 (Set DR-Submodularity). *A set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is submodular if, for any $A \subseteq B \subseteq \mathcal{V}$ and for any $e \in \mathcal{V} \setminus B$, it holds that*

$$f(A \cup \{e\}) - f(A) \geq f(B \cup \{e\}) - f(B) \tag{2.2}$$

An analogous, more general definition of submodularity is the following.

Definition 2.3 (Set Submodularity). *A set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is submodular if, for any $A, B \subseteq \mathcal{V}$, it holds that*

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B) \tag{2.3}$$

Remark. *In the context of set functions, set submodularity is equivalent to DR-submodularity.*

Set Supermodularity, a dual property of submodularity that finds applications especially in algorithmic game theory and abstract interpretation, is defined simply by reversing the sign of the above inequalities.

Definition 2.4 (Supermodularity). *A function f is supermodular if and only if $-f$ is submodular.*

A function that is both submodular and supermodular at the same time, i.e., in which the inequality in Definition 2.3 occurs as equality, is known as a *modular* function.

Definition 2.5 (Modularity). *A function $m : 2^{\mathcal{V}} \to \mathbb{R}$ is called modular if, for any $A, B \subseteq \mathcal{V}$, it holds that*

$$f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$$

Furthermore, modular functions can be decomposed as

$$f(S) = c + \sum_{i \in S} m_i,$$

where $c \in \mathbb{R}$ is a constant, and $m_i \in \mathbb{R}$, for all $i \in \mathcal{V}$.

Submodular functions have a number of desirable properties, which we briefly survey as follows.

Proposition 2.6. *If f is a submodular function and $A \subseteq \mathcal{V}$, then the following are also submodular:*

- ***Complement**: $g(A) := f(\mathcal{V} \setminus A)$.*
- ***Truncation**: $g(A) := \min\{f(A), c\}$, where c is a constant and f is monotone.*
- ***Minimum of two submodular functions**: $h(A) := \min\{f(A), g(A)\}$, if $m(A) := f(A) - g(A)$ is monotone submodular.*
- ***Scalar multiplication**: $g(A) := c \cdot f(A)$, where c is a non-negative constant.*
- ***Subset minimum**: $g(A) := \min_{B \subseteq A} f(B)$.*
- ***Superset minimum**: $g(A) := \min_{B \supseteq A} f(B)$.*
- ***Fixed set union**: $g(A) := f(A \cup B)$, given a fixed set $B \subseteq \mathcal{V}$.*

- ***Fixed set intersection**: $g(A) := f(A \cap B)$, given a fixed set $B \subseteq \mathcal{V}$.*

Moreover, if $f_1, f_2, \dots, f_m$ are m submodular functions, the *mixture function* defined as

$$g(S) = \sum_{i=1}^{m} \lambda_i \cdot f_i(S),$$

where $\lambda_i \geq 0$ for all $i \in [m]$, is submodular.

Theorem 2.7 (Cardinality-based set function (Section 6.1 of Bach, 2013))**.** *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set function. Let $c : \mathbb{Z}_+ \to \mathbb{R}$ be a defined such that*

$$f(S) := c(|S|),$$

where $|\cdot|$ is the cardinality of $S \subseteq \mathcal{V}$. Then, f is submodular if and only if c is a concave *function.*

Theorem 2.8. *Given two set submodular functions f_1 and f_2, their point-wise maximum*

$$f_{max}(S) := \max\{f_1(S), f_2(S)\}, \forall S \subseteq \mathcal{V}$$

is not necessarily submodular (Section 1.2 of Krause & Golovin, 2014).

Proof. We provide a brief proof by counterexample. Let f_1 and f_2 be both symmetric set submodular functions. From Definition 2.7, we know that f_1 and f_2 can be written as some concave functions w.r.t. the cardinality of the argument set. It immediately follows that the minimum of two concave functions is *not* concave in general. □

Theorem 2.9. *Given two set submodular functions f_1 and f_2, their point-wise minimum*

$$f_{min}(S) := \min\{f_1(S), f_2(S)\}, \forall S \subseteq \mathcal{V}$$

is not necessarily submodular (Section 1.2 of Krause & Golovin, 2014).

Proof. Considering a 2-dimensional ground set $\mathcal{V} = \{1, 2\}$, we consider the following counterexample:

$$\begin{aligned} &f_1(\emptyset) := 0; \quad f_1(\{1\}) := 1; \quad f_1(\{2\}) := 0; \quad f_1(\{1,2\}) := 0.5; \\ &f_2(\emptyset) := 0; \quad f_2(\{1\}) := 0; \quad f_2(\{2\}) := 1; \quad f_2(\{1,2\}) := 0.5. \end{aligned}$$

$f_{min} := \min\{f_1, f_2\}$ is then:

$$f_{min}(\emptyset) := 0; \quad f_{min}(\{1\}) := 0; \quad f_{min}(\{2\}) := 0; \quad f_{min}(\{1,2\}) := 0.5.$$

It is easy to see that both f_1 and f_2 are submodular, however their point-wise minimum f_{min} is *not* submodular. □

2.2 Examples of Submodular Functions

Submodular functions arise in several applications of practical utility. Here are some notable examples.

Example 2.10 (Modular functions (Section 3.1.1 of Krause & Golovin, 2014))**.** Modular *functions are the simplest example of submodular functions, in which for all $A, B \subseteq \mathcal{V}$, it holds that*

$$f(A) + f(B) = f(A \cup B) + f(A \cap B).$$

Modular set functions could be compared to linear functions, as their marginal gains (discrete derivatives) are constant: $f(e \mid A) = f(e \mid B)$ for all $A, B \subseteq \mathcal{V}$ and $e \ni nA \cup B$. A normalized modular function f can always be expressed in the form

$$f(S) = \sum_{e \in S} w(e) \tag{2.4}$$

for some weight function $w : \mathcal{V} \to \mathbb{R}$ For instance, $f(S) = |S|$ is monotone modular.

Example 2.11 (Weighted coverage functions (Section 3.1.1 of Krause & Golovin, 2014, Section 6.3 of Bach, 2013))**.** *A more proper example of a submodular function is the* weighted coverage *of a collection of sets. Let $g : 2^{\mathcal{V}} \to \mathbb{R}_+$ be a non-negative set modular function, and consider a collection U of subsets of $\mathcal{V}$. Then, for a subset $S \subseteq \mathcal{V}$, the function*

$$f(S) := g\left(\bigcup_{v \in S} v\right) := \sum_{x \in \bigcup_{v \in S} v} w(x), \tag{2.5}$$

is monotone submodular. Here, $w : \mathcal{V} \to \mathbb{R}_+$ is the weight function that represents g.
Notice that the well-known MaxCover *problem fits in this framework: we just need to define $g(A) := |A|$ (which is modular, as we have seen in Definition 2.10) and maximize $f(S)$, for $S \subseteq \mathcal{V}$. Noticeably, the weighted coverage function f is submodular even in the case of g being an arbitrary* submodular *function. Moreover, f is monotone if and only if g is monotone.*

Example 2.12 (Facility location (Section 3.1.1 of Krause & Golovin, 2014))**.** *Consider a ground set of locations $\mathcal{V} = \{1, 2, \dots, n\}$. In the facility location problem, one wishes to select some locations from $\mathcal{V}$ to open a new facility that serves a collection of m customers. Opening a new facility at location $j \in \mathcal{V}$ quantifies a service of value M_{ij} to customer i, with $M \in \mathbb{R}^{m \times n}$. It is assumed that customers choose the facility that provides the highest service value to themselves. With this assumption, we can model the total service value provided to all customers with the set function*

$$f(S) := \sum_{i=1}^{m} \max_{j \in S} M_{ij}. \tag{2.6}$$

Clearly, f should be a normalized function, so we set $f(\emptyset) := 0$. $f(S)$ is monotone submodular when $M_{ij} \geq 0$ for all i, j (Frieze, 1974).

This model can also be generalized to other applications. For instance, in the sensor placement *problem, M_{ij} may refer to the expected reduction in detection time obtained by placing sensor j in a scenario i (Xu et al., 2013).*

Example 2.13 (Cut functions (Section 3.1.1 of Krause & Golovin, 2014, Section 6.2 of Bach, 2013))**.** *Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph with $\mathcal{V}$ and $\mathcal{E}$ as vertex set and edge set, respectively, and with* *non-negative edge weights $w : \mathcal{V} \times \mathcal{V} \to \mathbb{R}_+$. Given $S \subseteq \mathcal{V}$, let ΔS be the* boundary *of S, defined as*

$$\Delta S := \{(u, v) \in \mathcal{E} \mid |S \cap \{u, v\}| = 1\}.$$

Then, the function

$$f(S) := \sum_{(u,v)\in\Delta S} w(u, v) \tag{2.7}$$

is submodular (Schrijver, 2003). The same results hold for directed graphs, in which case the boundary of S is rewritten as

$$\Delta S := \{(u, v) \in \mathcal{E} \mid u \in S, v \notin S\}.$$

It is easy to see that a cut function is, in general, non-monotone. In fact, $f(\emptyset) = f(\mathcal{V}) = 0$.

The MaxCut *(Goemans & Williamson, 1995) and* MaxDirectedCut *(Halperin & Zwick, 2001) problems are two special instances of submodular cut functions.*

2.3 Beyond Set Submodularity: Integer Lattice and Continuous Domains

So far, we have only discussed submodularity with a strong emphasis on set functions. However, the notion of submodularity extends not only beyond sets, but also it is not restricted to the discrete domain. We will now proceed to adapt some of the previous definitions to extend them to the general case. Keep in mind, however, that we will primarily deal with discrete functions in set and integer lattice domains.

Submodular functions are defined on subsets of $\mathbb{R}$: $\mathcal{X} := \prod_{i=1}^{n} \mathcal{X}_i$, where each $\mathcal{X}_i$ is a compact subset of $\mathbb{R}$ (Topkis, 1978; Bach, 2019).

Definition 2.14 (Submodularity). *A function $f : \mathcal{X} \to \mathbb{R}$ is submodular if, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, it holds that*

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \vee \boldsymbol{y}) + f(\boldsymbol{x} \wedge \boldsymbol{y}). \tag{2.8}$$

In particular, $\mathcal{X}_i$ could be a finite set, such as $\{0, 1\}$, in which case $f(\cdot)$ is referred to as a *set function*, or $\mathcal{X}_i$ could be $\{0, 1, \dots, k_i - 1\}$, for some integer k_i, in which case $f(\cdot)$ is called *integer-lattice function*. Moreover, $\mathcal{X}_i$ can be an interval, in which case it is called a *continuous domain*. In particular, $f : [0, 1]^{\mathcal{V}} \to \mathbb{R}$ is a said to be an *interval continuous function*.

Remark. *Notice how Eq. (2.8) resembles Eq. (2.3): the subsets $A, B \subseteq \mathcal{V}$ become the compact subsets (i.e., vectors in the general case) $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, and the set union $\cup$ (respectively, set intersection $\cap$) is replaced with the coordinate-wise maximum $\vee$ (respectively, coordinate-wise minimum $\wedge$).*

We also need to extend the previous definition of monotonicity.

Definition 2.15 (Monotonicity). *A function $f : \mathcal{X} \to \mathbb{R}$ is said to be monotone if, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$, it holds that*

$$\boldsymbol{x} \preccurlyeq \boldsymbol{y} \quad \Rightarrow \quad F(\boldsymbol{x}) \leq F(\boldsymbol{y}), \tag{2.9}$$

where $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$ refers to the vector partial order, in which $x_i \leq y_i$ for all $i \in \{1, 2, \dots, n\}$.

Remark. *Notice that, for set functions, Definition 2.15 is equivalent to the familiar*

$$A \subseteq B \quad \Rightarrow f(A) \leq f(B)$$

for all $A, B \in \mathcal{V}$.

We have already noted that, in the discrete case, set submodularity and diminishing-returns submodularity are two equivalent properties. However, in the general case, DR-submodularity is a subclass of submodular functions.

Definition 2.16 (DR-Submodularity). *A function $f : \mathcal{X} \to \mathbb{R}$ satisfies the* diminishing returns *(DR) property if, for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{X}$ such that $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$, for all $e \in \mathcal{V}$ and all $k \in \mathbb{R}_+$ such that $(\boldsymbol{x} + k\mathbf{1}_e)$ and $(\boldsymbol{y} + k\mathbf{1}_e)$ are still in $\mathcal{X}$, it holds that*

$$f(\boldsymbol{x} + k\mathbf{1}_e) - f(\boldsymbol{x}) \geq f(\boldsymbol{y} + k\mathbf{1}_e) - f(\boldsymbol{y}). \tag{2.10}$$

$f(\cdot)$ is said to be a DR-submodular *function.*

2.4 Continuous Extensions

Notice that we could represent each subset $S \subseteq \mathcal{V}$ as a binary vector $\mathbf{1}_S \in \{0,1\}^{\mathcal{V}}$:

$$\mathbf{1}_S(e) := \begin{cases} 1 & \text{if } e \in S \\ 0 & \text{if } e \notin S \end{cases} \qquad \text{for all } e \in \mathcal{V}$$

Such a binary representation, called *indicator vector*, links the powerset of the ground set $\mathcal{V}$ to every vertex of the n-dimensional hypercube.

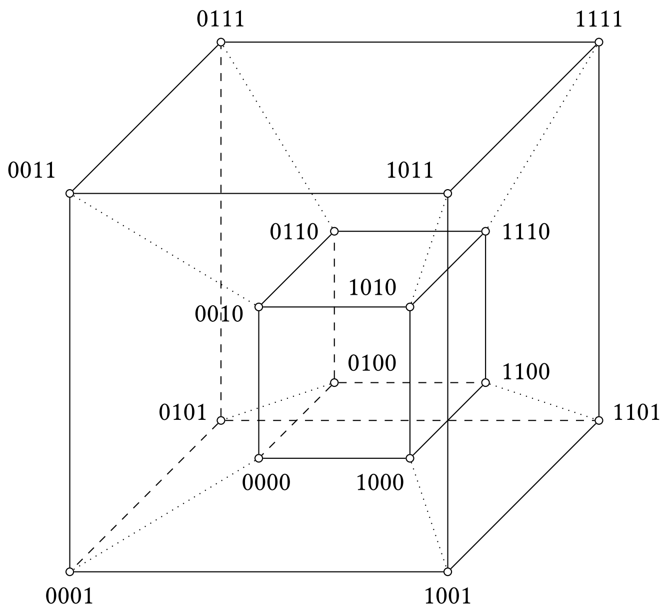
A particular feature of set functions defined on $\{0,1\}^{\mathcal{V}}$ is that they have some natural extensions to the unit $[0,1]^n$ hypercube domain. In fact, two natural extensions for submodular functions are the *convex closure* and *concave closure*. The convex closure of a submodular set function f is the point-wise largest convex function $f^- : [0,1]^n \to \mathbb{R}$ that lower bounds f. Analogously, the concave closure of f is the point-wise smallest concave function $f^+ : [0,1]^n \to \mathbb{R}$ that upper bounds f. We will see that *we can reduce the problem of minimizing a set function to the minimization of its convex closure.* Similarly, the *concave closure has strong ties with set submodular maximization, but it cannot be used directly.*

Before proceeding forward, let us recall some useful definitions about convexity and concavity.

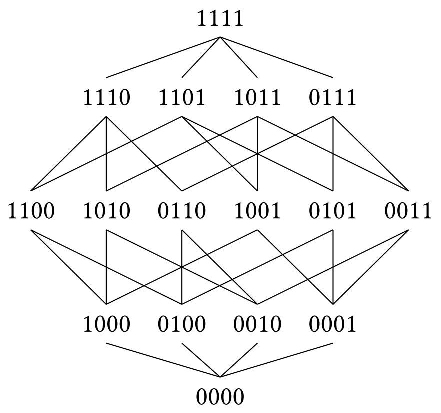
Definition 2.17 (Convex Set). *A set $C \subseteq \mathbb{R}^n$ is called* convex set *if $\lambda x + (1-\lambda)y$ belongs to C for all $x, y \in \mathbb{R}^n$ and for all $\lambda \in [0,1]$.*

Definition 2.18 (Support Function of a Convex Set). *The* support function *of a convex set C is the function obtained by maximizing linear functions $w^T s$ over $s \in C$, which is convex in w.*

Definition 2.19 (Convex Hull). *Given a set $C \subseteq \mathbb{R}^n$, its* convex hull *is the smallest convex set containing C. We denote the convex hull of C as $conv(C)$. Notice that $conv(C)$ is the unique convex set in $\mathbb{R}^n$ such that $C \subseteq conv(C) \subseteq C'$ for every convex set C' containing C.*



(a) $[0, 1]^n$ hypercube, with $n = 4$.



(b) Hasse diagram for the $(\mathcal{V}, \subseteq)$ partially ordered set, with $\mathcal{V} = \{1, 2, 3, 4\}$.

Figure 2.1: Comparison between the continuous $[0, 1]^{\mathcal{V}}$ hypercube at the top, and the discrete $\{0, 1\}^{\mathcal{V}}$ partially ordered set. The binary vectors $\boldsymbol{x} = [v_1, v_2, v_3, v_4]$ are compactly represented for visual purposes. For instance, the vector $\boldsymbol{x} = [0, 1, 1, 0]$, identifying the set $\{v_2, v_3\}$, is compactly represented as 0110.

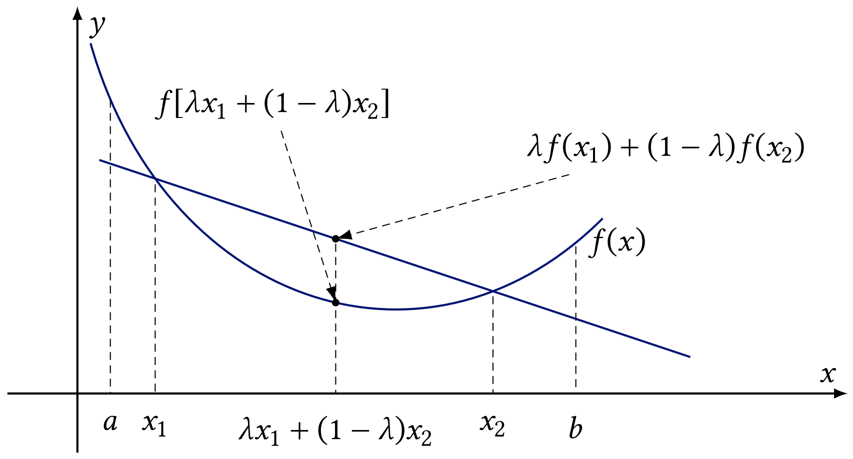

y
$f[\lambda x_1 + (1-\lambda)x_2]$
$\lambda f(x_1) + (1-\lambda)f(x_2)$
$f(x)$
x
a
x_1
$\lambda x_1 + (1-\lambda)x_2$
x_2
b

Figure 2.2: Example of a convex function. The line segment joining any two points x_1 and x_2 cannot lie below the functional graph.

Remark. *The unit hypercube* $[0, 1]^n$ *is the convex hull of all the points whose n Cartesian coordinates are each either 0 or 1. Thus,* $[0, 1]^n$ *is a convex set.*

Definition 2.20 (Convex Function)**.** *A function* $f : \mathbb{R}^n \to \mathbb{R}$ *is* convex *if its domain* $\mathbb{R}^n$ *is a convex set and, for all* $x, y \in \mathbb{R}^n$ *and for all lambda* $\in [0, 1]$, *it holds that*

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

Intuitively, if f *is convex, no line segment joining any two points* x, y *of the function* f *lies below the functional graph.*

Definition 2.21 (Concave Function)**.** *A function* f *is concave if its negation* $-f$ *is convex. Intuitively, if* f *is concave, no line segment joining any two points* x, y *of the function* f *lies above the functional graph.*

Definition 2.22 (Extension of a Set Function)**.** *An* extension *of a set function* $f : 2^{\mathcal{V}} \to \mathbb{R}$ *is some function* $f' : [0, 1]^n \to \mathbb{R}$ *that agrees with the values of* f *on the vertices of the* $\{0, 1\}^n$ *hypercube.*

2.4.1 Convex Closure

The convex closure $f^- : [0, 1]^{\mathcal{V}} \to \mathbb{R}$ exists for any set function $f : 2^{\mathcal{V}} \to \mathbb{R}$, regardless of the submodularity of f. Formally, f^- is defined as follows.

Definition 2.23 (Convex Closure)**.** *Given a set function* $f : 2^{\mathcal{V}} \to \mathbb{R}$, *its convex closure* $f^- : [0, 1]^{\mathcal{V}} \to \mathbb{R}$ *is the point-wise highest convex function in the hypercube domain that always lower-bounds* f.

An alternative (but mathematically equivalent) formulation defines the convex closure in terms of distributions of subsets of the ground set $\mathcal{V}$.

Definition 2.24 (Convex Closure (probabilistic formulation))**.** *Let* $f : 2^{\mathcal{V}} \to \mathbb{R}$ *be a set function. For every* $\boldsymbol{x} \in [0, 1]^{\mathcal{V}}$, *let* D_f^- *indicate a distribution over* $2^{\mathcal{V}}$, *with marginals* $\boldsymbol{x}$, *minimizing* $\mathbb{E}_{S \sim D_f^-(\boldsymbol{x})}[f(S)]$, *breaking ties arbitrarily. Then*

$$f^- := \mathbb{E}_{S \sim D_f^-(\boldsymbol{x})}[f(S)],$$

i.e., the convex closure $f^-(\boldsymbol{x})$ is the expected value of $f(S)$ over draws S from the distribution $D_f^-(\boldsymbol{x})$.

We now state some basic properties about the convex closure of f.

Proposition 2.25. *Consider the convex closure $f^- : [0,1]^{\mathcal{V}} \to \mathbb{R}$ of a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$. The following properties hold:*

- *The convex closure f^- is an extension, as it agrees with f on every integer point in $[0,1]^{\mathcal{V}}$ (i.e., it agrees on every vertex of the hypercube);*
- *The minimum of f^- is attained at an integer point, since f^- only assumes values that correspond to distributions of $2^{\mathcal{V}}$.*

Proposition 2.26. $\min\{f\} = \min\{f^-\}$. *If S is a minimizer for $f(S)$, then $\mathbf{1}_S$ is a minimizer for $f^-(S)$. Furthermore, if $\boldsymbol{x}$ is a minimizer of f^-, then every set in the support of $D_f^-(\boldsymbol{x})$ is a minimizer of f^-.*

2.4.2 From the Convex Closure to the Lovász Extension

We have seen that the convex closure f^- of a set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$ exists, it is well-defined, and it has desirable nice properties. However, it is not clear how to compute it efficiently for any $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$. We thus proceed to describe an extension $\hat{f} : [0,1]^{\mathcal{V}} \to \mathbb{R}$, introduced in Lovász (1983), which is equivalent to the convex closure when f is a set submodular function.

Definition 2.27 (Lovász Extension (Lovász, 1983))**.** *Consider a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ and a vector $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$. Let $\mathcal{V}' = \{v_1, v_2, \dots, v_n\}$ be a permutation of $\mathcal{V}$ such that $x_{v_1} \geq x_{v_2} \geq \cdots \geq x_{v_n}$. We denote $S_i := \{v_1, \dots, v_i\}$ for $0 \leq i \leq n$. Let $\{\lambda_i\}_{i=0}^{0}$ be the unique non-negative coefficients with $\sum_i \lambda_i = 1$ such that*

$$\boldsymbol{x} = \sum_{i=0}^{n} \lambda_i \mathbf{1}_{S_i}.$$

Clearly, $\lambda_n = x_{v_n}$, and, for $0 \leq i < n$ it holds that $\lambda_i = x_{v_i} - x_{v_{i+1}}$.
The Lovász extension $\hat{f} : [0,1]^{\mathcal{V}} \to \mathbb{R}$ is defined as

$$\hat{f}(\boldsymbol{x}) := \sum_i \lambda_i f(S_i).$$

The Lovász extension has a probabilistic interpretation as well. Given a set of marginal probabilities $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$, one can define a particular distribution $\hat{D}_f$ on $2^{\mathcal{V}}$ satisfying these marginals. Intuitively, this distribution (subject to obeying the marginals) attributes as much probability mass as possible to the large subsets of $\mathcal{V}'$. Hence, the largest set $S_n := \mathcal{V}'$ gets the highest possible probability mass (subject to the smallest marginal x_{v_n}), then the next largest set gets the second highest probability mass (subject to the marginal $x_{v_{n-1}}$), and so on. One can then realize that $\hat{f}(\boldsymbol{x})$ is just the expected value of f on draws sampled from $\hat{D}_f(\boldsymbol{x})$.

Remark. *The distributions $\hat{D}_f(\boldsymbol{x})$ implied by the Lovász extension are* oblivious, *as they do not depend on the particular set function f.*

Theorem 2.28. *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set submodular function. For any $x \in [0,1]^{\mathcal{V}}$, $D_f^-(\boldsymbol{x}) = \hat{D}_f(\boldsymbol{x})$, so $f^-(\boldsymbol{x}) = \hat{f}(\boldsymbol{x})$. Therefore, the convex closure and the Lovász extension are equivalent.*

We now introduce two particular polyhedra in $\mathbb{R}^n$ associated with submodular functions which are crucial for submodular minimization algorithms, and we show how the same procedure that allows us to easily compute the Lovász extension is closely tied to submodular minimization.

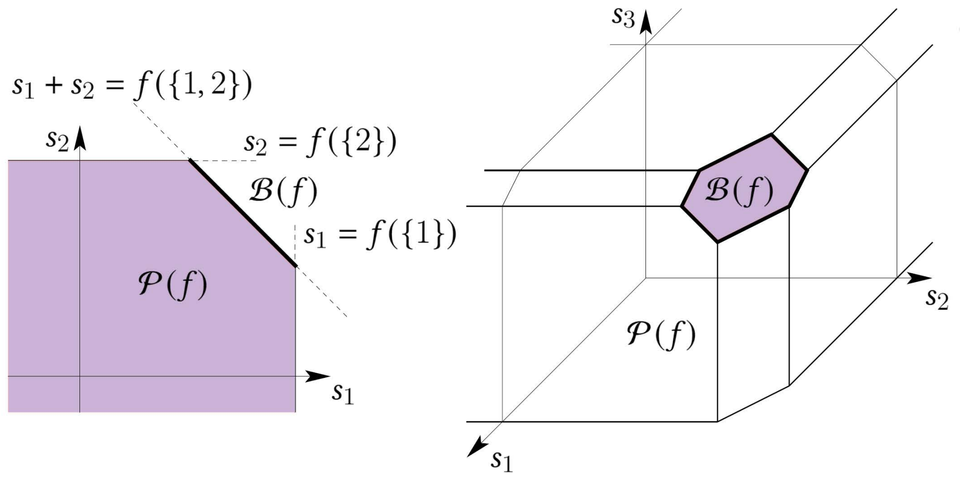


Figure 2.3: Example of a submodular polyhedron and base polyhedron for $n = 2$ (left) and $n = 3$ (right), for a monotone normalized submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$ for which $\mathcal{B}(f) \subseteq \mathbb{R}^n_+$. This figure is adapted from Bach (2013).

Definition 2.29 (Submodular polyhedra (Fujishige, 2005)). *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a* normalized[1] *set submodular function. Let $\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}}$ be a vector, for which we define its canonical modular function: $\boldsymbol{x}(S) := \sum_{e\in S} x_e$ for every $S \subseteq \mathcal{V}$. The* submodular polyhedron *and the* submodular base polyhedron *of f, noted as $\mathcal{P}(f)$ and $\mathcal{B}(f)$ respectively, are defined as:*

$$\mathcal{P}(f) := \{\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}} \mid \boldsymbol{x}(S) \leq f(S), \forall S \subseteq \mathcal{V}\} \tag{2.11}$$

$$\begin{aligned}\mathcal{B}(f) &:= \{\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}} \mid \boldsymbol{x}(\mathcal{V}) = f(\mathcal{V}), \boldsymbol{x}(S) \leq f(S), \forall S \subseteq \mathcal{V}\} & (2.12)\\ &= \mathcal{P}(f) \cap \{\boldsymbol{x}(\mathcal{V}) = f(\mathcal{V})\}. & (2.13)\end{aligned}$$

Notice that the base polyhedron $\mathcal{B}(f)$ is the face of $\mathcal{P}(f)$ satisfying $\boldsymbol{x}(\mathcal{V}) = f(\mathcal{V})$, i.e., it is defined as the intersection of the hyperplanes $\{\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}} \mid \boldsymbol{x}(A) \leq f(A)\} := \{\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}} \mid \boldsymbol{x}^\top \mathbf{1}_S \leq f(S)\}$, whose normals are indicator vectors $\mathbf{1}_A$ of $A \subseteq \mathcal{V}$. See Fig. 2.3 for some canonical examples with $n = 2$ and $n = 3$ due to Bach (2013).

A natural problem to consider over these polyhedra is the maximization of the linear objective $\boldsymbol{w}^\top \boldsymbol{x}$, given some weights $\boldsymbol{w} \in \mathbb{R}^{\mathcal{V}}$. Since $\boldsymbol{y} \preccurlyeq \boldsymbol{x} \in \mathcal{P}(f)$ implies $\boldsymbol{y} \in \mathcal{P}(f)$, it is easy to see that when $w_e < 0$ for some element $e \in \mathcal{V}$, the objective value $\boldsymbol{w}^\top \boldsymbol{x}$ is unbounded on $\mathcal{P}(f)$. However, if we consider a non-negative weight vector $\boldsymbol{w} \in \mathbb{R}^{\mathcal{V}}_+$, we show that an optimal solution x^* must lie in $\mathcal{B}(f)$. We can thus formalize the following linear program:

[1]If $f(\emptyset) \neq 0$, one can re-define $f(S)$ to be $f(S) - f(\emptyset)$. Notice that this does not influence submodularity or submodular function minimization.

$$\begin{aligned} \max \quad & \boldsymbol{w}^\top \boldsymbol{x} & \\ \text{s.t.} \quad & \boldsymbol{x}(S) \leq f(S) & \text{for all } S \subset \mathcal{V} \\ & \boldsymbol{x}(\mathcal{V}) = f(\mathcal{V}) & \\ & x_e \text{ free} & \text{for all } e \in \mathcal{V} \end{aligned} \tag{2.14}$$

It is known that Eq. (2.14) is a linear programming problem with a finite optimum solution (Fujishige, 2005; Bach, 2013). From duality theory, this primal problem then admits a dual minimization problem which shares the same solution. The dual of Eq. (2.14) is:

$$\begin{aligned} \min \quad & \sum_{S \subseteq \mathcal{V}} f(S) \pi_S & \\ \text{s.t.} \quad & \sum_{e \in S} \pi_S = w_e & \text{for all } e \in \mathcal{V} \\ & \pi_S \geq 0 & \text{for all } S \subset \mathcal{V} \\ & \pi_e \text{ free} & \text{for all } e \in \mathcal{V} \end{aligned} \tag{2.15}$$

Notice that Eq. (2.15) is a linear problem with an exponential number of variables (since the number of constraints in the primal problem is exponential as well), which is not feasible for a linear programming solver. However, one remarkable property of submodularity is that both these problems can be solved with a celebrated *greedy algorithm* due to Edmonds (1971). Let $\mathcal{V}' = \{v_1, v_2, \dots, v_n\}$ be a permutation of $\mathcal{V}$ such that $v_1 \geq v_2 \geq \cdots \geq v_n$. We denote $S_i := \{v_1, \dots, v_i\}$ for $0 \leq i \leq n$. Then, an optimal primal solution of the problem presented above is given by

$$x_{v_i} = f(S_i) - f(S_{i-1}) \quad \text{for some } 0 \leq i \leq n,$$

whereas the optimal dual solution is given by

$$\pi_S = \begin{cases} w(v_i) - w(v_{i+1}) & \text{if } S = \{v_1, \dots, v_i\} \text{ for some } 0 \leq i \leq n \\ 0 & \text{otherwise.} \end{cases}$$

Notice that this greedy algorithm is essentially the same procedure used to define the Lovász extension in Definition 2.27.

Theorem 2.30 ((Lovász, 1983)). *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set submodular function. Then, the subset $S \subseteq \mathcal{V}$ minimizing f can be found in polynomial time. In fact, rather then minimizing f over the subsets $S \subseteq \mathcal{V}$, it is sufficient to minimize its Lovász extension $\hat{f}$ over the hypercube $[0,1]^{\mathcal{V}}$. This is possible thanks to two properties:*

1. *$\hat{f}(\boldsymbol{x})$ can be evaluated in polymial time for any $x \in [0,1]^{\mathcal{V}}$;*

2. *If f is submodular, then $\hat{f}$ is convex (since the Lovász extension is equivalent to the convex closure), and therefore any standard algorithm for convex optimization can be used to find the minimum (for instance, one can use the algorithm due to Yudin & Nemirovskii (1976)).*

2.4.3 Concave Closure

The concave closure $f^+ : [0,1]^{\mathcal{V}} \to \mathbb{R}$ is defined analogously to the convex closure. It exists for any set function $f : 2^{\mathcal{V}} \to \mathbb{R}$, regardless of the submodularity of f. However, the concave closure is NP-Hard to evaluate f^+, even in the simple case of f being a graph cut function. Formally, f^+ is defined as follows.

Definition 2.31 (Concave Closure). *Given a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$, its concave closure $f^+ : [0,1]^{\mathcal{V}} \to \mathbb{R}$ is the point-wise lowest concave function in the hypercube domain that always upper-bounds f.*

An alternative (but mathematically equivalent) formulation defines the concave closure in terms of distributions of subsets of the ground set $\mathcal{V}$.

Definition 2.32 (Concave Closure (probabilistic formulation)). *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set function. For every $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$, let D_f^+ indicate a distribution over $2^{\mathcal{V}}$, with marginals $\boldsymbol{x}$, maximizing $\mathbb{E}_{S\sim D_f^+(\boldsymbol{x})}[f(S)]$, breaking ties arbitrarily. Then*

$$f^+ := \mathbb{E}_{S\sim D_f^+(\boldsymbol{x})}[f(S)],$$

i.e., the concave closure $f^+(\boldsymbol{x})$ is the expected value of $f(S)$ over draws S from the distribution $D_f^+(\boldsymbol{x})$.

We now state some basic properties about the concave closure of f.

Proposition 2.33. *Consider the convex closure $f^+ : [0,1]^{\mathcal{V}} \to \mathbb{R}$ of a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$. The following properties hold:*

- *The concave closure f^+ is an extension, as it agrees with f on every integer point in $[0,1]^{\mathcal{V}}$ (i.e., it agrees on every vertex of the hypercube);*
- *It is NP-Hard to evaluate $f^+(\boldsymbol{x})$ for any $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$ (Calinescu et al., 2007; Vondrák, 2007).*

Moreover, stronger hardness results have been investigated in the literature. Even in the simple case of f being a monotone coverage function and k an integer, the convex optimization problem $\max\{f^+(\boldsymbol{x}) \mid \mathbf{1}^\top \boldsymbol{x} \leq k\}$ is APX-Hard (Vondrák, 2007). Furthermore, maximizing a general submodular function in the value oracle model with an approximation ratio better than $\frac{1}{2}$ is hard, regardless of whether $P \neq NP$ (Feige et al., 2007). There is thus no way of devising exact polynomial time algorithms for maximizing submodular functions in real-world scenarios using f^+ or f. Another problem of the concave closure is that any concave extension has a non-trivial integrality gap relative to the polytopes that arise with the most commonly used constraints, included the hypercube $[0,1]^{\mathcal{V}}$.

2.4.4 From the Concave Closure to the Multilinear Extension

To overcome the limitations of the concave closure, Calinescu et al. introduced the multilinear extension F of a set function f (Calinescu et al., 2007). A function $F : [0,1]^{\mathcal{V}} \to \mathbb{R}$ is called *multilinear* whenever F is linear in each coordinate x_i while the other variables $\{x_i\}_{j\neq i}$ are fixed. Furthermore, using induction on the dimension n, one could see that any multilinear function is uniquely determined by its values on the vertices of the hypercube $[0,1]^{\mathcal{V}}$.

The multilinear extension F of f is the unique multilinear function that agrees with f on every integer point of the hypercube $[0,1]^{\mathcal{V}}$. Formally, it is defined as follows.

Definition 2.34 (Multilinear Extension (Calinescu et al., 2007)). *Consider a set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$ and a vector $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$. The* multilinear extension $F : [0,1]^n \to \mathbb{R}$ *of f is defined as*

$$F(\boldsymbol{x}) := \sum_{S \subseteq \mathcal{V}} f(S) \cdot \prod_{i \in S} x_i \cdot \prod_{i \in \mathcal{V} \setminus S} (1 - x_i) \tag{2.16}$$

Similarly to the Lovász extension, we observe that the multilinear extension has a probabilistic interpretation as well. $F(\boldsymbol{x})$ corresponds to a distribution with marginals at $\boldsymbol{x}$. Let D^i_f be the distribution over $2^{\mathcal{V}}$ that selects every element $e \in \mathcal{V}$ i.i.d. with probability x_e. Then, $F(\boldsymbol{x})$ is the expected value of $f(S)$ over draws S from the distribution $D^i_f(\boldsymbol{x})$. We can then give the following equivalent definition for the multilinear extension F.

Definition 2.35 (Multilinear Extension (probabilistic formulation) (Calinescu et al., 2007)). *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set function. For every $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$, let D^i_f indicate the distribution over $2^{\mathcal{V}}$ that selects every element $e \in \mathcal{V}$ independently with probability x_e. Then, the value of the multilinear extension $F : [0,1]^{\mathcal{V}} \to \mathbb{R}$ of f is the expected value of f over draws from D^i_f:*

$$F(\boldsymbol{x}) := \mathbb{E}_{S \sim D^i_f(\boldsymbol{x})} f(S) = \sum_{S \subseteq \mathcal{V}} f(S) \cdot \prod_{i \in S} x_i \cdot \prod_{i \notin S} (1 - x_i).$$

Remark. *Analogously to the Lovász extension, the multilinear extension is defined on an* oblivious *distribution, i.e., the distribution underlying $F(\boldsymbol{x})$ does not depend on the set function f.*

The following lemma describes the relation between the multilinear and the Lovász extensions.

Lemma 2.36 (Lemma A.4 in Vondrák, 2009). *Let F and $\hat{f}$ be the multilinear and Lovász extension, respectively, of a set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$. Then, $F(\boldsymbol{x}) \geq \hat{x}$ for each $\boldsymbol{x} \in [0,1]^n$.*

Notice that the multilinear extension has some concave-like properties, even though it is not fully concave.

Proposition 2.37. *Let $F : [0,1]^n \to \mathbb{R}$ be the multilinear extension of a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$. The following properties hold (Calinescu et al., 2007; Vondrak, 2008):*

1. *If f is monotone, then F is monotone along any direction, i.e., $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$ implies $F(\boldsymbol{x}) \leq F(\boldsymbol{y})$;*
2. *If f is submodular, then F is up-concave, i.e., for each $\boldsymbol{x} \in [0,1]^n$, it holds that $\frac{\partial^2 F}{\partial x_i \partial x_j}(\boldsymbol{x}) \leq 0$, for every $i \neq j$;*
3. *If f is a set submodular function, then F is cross-convex, i.e., for any $i \neq j$, the function $F^x_{i,j}(\varepsilon) := F(x + \varepsilon(e_i - e_j))$ is convex, as a function of $\varepsilon \in \mathbb{R}$.*

Remark. *Notice that cross-convexity does not contradict set submodularity and the independent distribution interpretation of the multilinear extension. In fact, the probability that both $i, j \in \mathcal{V}$ are drawn by an independent uniform distribution is a* concave *function of ε. Since this co-occurrence corresponds to the DR property, or to a* decrease *of the expected value of f, this implies that the expectation of f is* convex *in ε.*

Before moving on with the relationship between the multilinear extension and submodular maximization, consider a vector $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$ and any fractional x_i, x_j values.
The technical issue with the multilinear extension F of a set submodular function f is that evaluating it requires exactly 2^n evaluations of f, which is clearly infeasible. However, we can use a nice approximation result with important ties to submodular maximization.

Theorem 2.38 ((Vondrák, 2007)). *Consider a set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$ and its multilinear extension $F : [0,1]^n \to \mathbb{R}$. Let $R_1, R_2, \dots, R_k$ be independent samples of random sets for some integer k, where the element $i \in \mathcal{V}$ is drawn independently with probability x_i. Then, it holds that*

$$\left| \frac{1}{k} \sum_{i=1}^{k} f(R_i) - F(\boldsymbol{x}) \right| \leq \varepsilon |\max f(S)|$$

with probability at least $1 - e^{-k\frac{\varepsilon^2}{4}}$.

Proof. We can rewrite $F(\boldsymbol{x})$ as $\mathbb{E}[f(R)]$, where R is a random collection of sample sets as defined in the theorem. Let $M = \max |f(S)|$. $f(R)$ is then a random variable that can assume values in the range $[-M, +M]$. Let $Y_i := \frac{1}{M} f(R_i)$, where R_i is the i-th random sample. Then, Y_i is a random variable in the $[-1, 1]$ range, and the sum of the expected values of Y_i for $1 \leq ileqk$ is $\frac{k}{M}F(\boldsymbol{x})$. Using a Chernoff bound argument, we have that:

$$Pr\left(\left| \sum_{i=1}^{k} Y_i - \frac{k}{M}F(\boldsymbol{x}) \right| > k\varepsilon \right) < e^{-k^2\frac{\varepsilon^2}{4k}} = e^{-k\frac{\varepsilon^2}{4}}$$

□

2.5 Representation of a Submodular Function

The problems studied in this thesis can be formalized using one or more submodular functions, but so far we have only presented the submodularity in an abstract, mathematical fashion. However, the explicit representation of submodular functions (and the possible constraints of the optimization problem) might be exponential in the size of the ground set $\mathcal{V}$. This raises the concern of how to represent a submodular function, in order to write and practical experiments on a computing machine.

The ground set $\mathcal{V}$ can be represented by a set of integers $\{1, 2, \dots, n\}$, without loss of generality, assuming that n integers can fit in memory. However, the first issue we face is that, if we chose to list the result values for all possible subsets $S \subseteq \mathcal{V}$, we would need a space exponentially large in n. This is not desirable at all. The second issue is more structural. Some problems allow for a **compact representation** of the submodular function, using some data structures that require a space that is polynomial in n, such as a list of sets in *maximum coverage* functions, or a bipartite graph in *optimal budgete allocation* problems. A third issue is that there are applications where evaluating $f(\cdot)$ per se may be difficult (Kempe et al., 2003), and in these cases we may only be able to obtain an approximate value up to some multiplicative relative error ε. Luckily, most results in submodular optimization are robust against such errors (see Goundan & Schulz (2007); Calinescu et al. (2007); Streeter & Golovin (2008); Golovin & Krause (2011)).

2.5.1 Value Oracle Model

Typically, and in the rest of the thesis, we will assume the **value oracle** model, in which the submodular function f is given in terms of a black box that computes $f(\cdot)$ on any valid input. This value oracle is particularly useful to abstract away the details of the specific problem in a generic algorithm that solves a submodular optimization problem (with or without additional constraints). As a consequence, we will consider the time complexity of all algorithms in this thesis in terms of the number of value-oracle queries.

2.6 Computational Models for Submodular Optimization

Classical approaches in submodular optimization require a *centralized* access to a dataset, which implies a large enough memory and storage. However, several large-scale machine learning problems, such as exemplar-based (Dueck & Frey, 2007) clustering and text summarization (Lin & Bilmes, 2011), require extracting a representative subset of a manageable size from a massive dataset that is impractical to fit in the memory of a single machine (Mirzasoleiman et al., 2013). Such instances are often modeled as a set submodular maximization problem with cardinality constraints (Gomes & Krause, 2010; Krause & Golovin, 2014).

Data volumes are expanding much faster than the ability of a single computer to process them. Thus, to keep up with today's massive datasets, we need to use several machines at once that process a portion of the data and coordinate with each other to solve the target problem with the best possible approximation ratio. Unfortunately, the weak point of greedy and exact algorithms - that work well for centralized submodular maximization and minimization - is that they are inherently *sequential*. They are therefore unsuitable for truly *distributed* or *decentralized* architectures. A novel class of algorithms that exploit the submodularity property has thus emerged to solve decentralized optimization tasks, in which the goal is to maximize or minimize a global objective function through local computations distributed over a cluster network and communication among computing nodes. For more details, we refer to Mirzasoleiman et al. (2013); Mokhtari et al. (2018); Bateni et al. (2018); Testa et al. (2020).

Another promising computational model involves a single computing unit (similarly to the centralized case), but it assumes the dataset is consumed "on the fly", without storing it all at once, and that the results may be computed even before having seen the entire ground set once. Performance of streaming algorithms is usually measured by the maximum *amount of memory* required at any time, the *number of passes* the algorithm needs to make over the data stream, as well as the usual number of oracle queries and the approximation ratio. This approach is particularly attractive for submodular maximization problems subject to cardinality constraints. We mention the following recent papers on streaming submodular maximization: Badanidiyuru et al. (2014); Kazemi et al. (2019); Buschjäger et al. (2021).

The novel algorithms presented in these thesis assume the standard centralized computational model, as the combinatorial optimization problems we studied are not expensive space-wise. However, we present some parallel experiments on the *budgeted maximum coverage* problem in the Appendix.

2.7 Concluding Remarks

In this chapter, we have introduced the notion of submodularity and the diminishing returns (DR) property, the value oracle model, and the computational models for submodular optimization, focusing on the *centralized* setting. Furthermore, we have explored some of the connections between submodular functions, convexity, and concavity. Definition 2.2 seems to suggest that submodularity is related to concavity. In fact, if we consider a scalar function $g : \mathbb{R} \to \mathbb{R}$ and a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ defined by $f(S) = g(|S|)$ for all $S \subseteq \mathcal{V}$, it is easy to see that f is submodular if and only if g is concave. The results from Lovász (1983) and the Lovász extension itself tend to suggest that submodularity is related to convexity, instead. As it turns out, the link between convexity and submodularity is stronger than the “concave view” of the submodular functions. In particular, minimizing a submodular function is “easy”, while maximizing it optimally is “hard”.

Indeed, Murota (1998; 2003) studied and developed a theory of discrete convexity based on submodularity, in which many class convexity results are matched to their submodular counterpart. For a more in-depth study of submodular functions and their applications, we refer the reader to Lovász, 1983; Fujishige, 2005; Vondrák, 2007; Bach, 2013; Krause & Golovin, 2014, and Section III.3 of Nemhauser & Wolsey, 1988.

3 Previous Work on Set Submodular Optimization

After presenting the preliminaries to submodularity and its extensions, we are now ready to present the major results on submodular optimization in the literature. We focus our attention on the *centralized* setting and on set submodular optimization. We first present some historical and state of the art results on submodular function minimization, in particular on the unconstrained case, which is solvable in polynomial time. Afterwards, we discuss the evolution of approximation algorithms for submodular function maximization, highlighting the differences between the *monotone* and *non-monotone* scenarios. We will also present various constrained scenarios, which are of particular interest for submodular maximization, and which prepare the reader for the next chapter. As we have seen in Section 2.5.1, we assume access to a value oracle for f which returns the value of $f(S)$ for any subset $S \subseteq \mathcal{V}$.

3.1 Submodular Set Minimization

Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set submodular function. The Submodular Function Minimization (SFM) problem is the following:

$$\min_{S \subseteq \mathcal{V}} f(S). \tag{3.1}$$

In this section, we let EO denote the time needed for one oracle query for f. The most naive algorithm for SFM would compute the 2^n possible values of $f(S)$ and pick the smallest, but this would require exponential time and hence be impractical in practice. We will only cite algorithms that are polynomial in n. The runtime of some algorithms might also be dependent on f, as measured by some upper bound M on $\max_{S \subseteq \mathcal{V}} |f(S)|$. In such cases, since f could be scaled to arbitrarily shrink the value of M, it is common to assume that the submodular function f is integer-valued: $f : 2^{\mathcal{V}} \to \mathbb{Z}$. In SFM, an algorithm that is polynomially dependent on n and M is said to be *pseudo-polynomial*. If the dependency is polynomial in n and $\log M$, however, the algorithm is called *weakly polynomial*. Finally, if $f(\cdot)$ assumes values in $\mathbb{R}$, or if M is too large, then a *strongly polynomial* algorithm is preferable, i.e., an algorithm that is polynomial in n and independent of M.

The importance of SFM has been discussed since the early years of combinatorial optimization, when Edmonds (1970) established many of the fundamental results of submodularity. First approaches to SFM relied on the ellipsoid method Grötschel et al. (1981), thanks to the properties of the Lovász extension (Section 2.4.2). In the literature, ellipsoid-like methods are contrasted with *combinatorial* algorithms. Combinatorial algorithms that rely only on additions, subtractions, and comparisons are called *fully combinatorial* (Schrijver, 2000). The early results on SFM were quite unsatisfactory, as the ellipsoid method is not particularly practical, nor does it gives a good combinatorial insight into the problem. The following statement from Cunningham (1985) is emblematic:

> It is an outstanding open problem to find a practical combinatorial algorithm to minimize a general submodular function, which also runs in polynomial time.

Cunningham also set the ground to solve this open problem by proposing a MaxFlow framework for SFM that used a linear programming duality result due to Edmonds (1970). Schrijver (2000) and Iwata et al. (2001) developed the first strongly polynomial algorithms for SFM, all of which rely on Cunningham's framework. Orlin (2007) and Iwata & Orlin (2009) proposed two significantly faster combinatorial approaches that too use Cunningham's framework. Currently, the method in Orlin (2007) is the theoretically fastest fully combinatorial algorithm strongly polynomial for SFM, which runs in $\mathcal{O}(n^5 EO + n^6)$ time. Similarly, Iwata & Orlin (2009) provided the fastest fully combinatorial weakly polynomial algorithm, which requires $\mathcal{O}((n^4 EO + n^5) \log nM)$ time.

In practice, however, the combinatorial algorithms are still far from practical due to the high polynomial coefficients of their theoretical runtime. In a recent survey (Bach, 2013), other *practically fast* procedures have been explored (especially for machine learning applications), but most of those approaches either rely on a special class of submodular functions such as decomposable functions (Stobbe & Krause, 2010; Jegelka et al., 2011), or on particular SFM problem scenarios or constraints that we do not discuss in this thesis (Iyer et al., 2013b;c; Iyer & Bilmes, 2013; Jegelka et al., 2013).

3.1.1 The Minimum Euclidean Norm Point Problem

The issues of practicality bring us back to non-combinatorial methods. Wolfe (1976) developed a combinatorial method to compute the minimum Euclidean norm point in a polytope as long as linear functions can be minimized over it (notice that this procedure is distinct from the well-known Frank-Wolfe algorithm, which was proposed in Frank & Wolfe (1956)). Fujishige (1980) was the first to realise that Wolfe's procedure can be used for SFM thanks to Edmonds's greedy algorithm, the same simple greedy algorithm that was used a few years later to define the Lovász extension. Indeed, although the base submodular polytope $\mathcal{B}(f)$ has exponentially many constraints, the simple greedy algorithm that sorts the ground set $\mathcal{V}$ as $\mathcal{V}' = \{v_1, v_2, \dots, v_n\}$ such that $v_1 \geq v_2 \geq \cdots \geq v_n$ can minimize any linear function over $\mathcal{B}(f)$. This approach was formalized in Fujishige (1984), and since then this is known as the Fujishige-Wolfe algorithm.

Definition 3.1 (Minimum Euclidean Norm Point). *Let $\mathcal{P} \subset \mathbb{R}^n$ be a convex polytope. We assume $\mathcal{P}$ is presented as the convex hull of finitely many points $\boldsymbol{p_1}, \boldsymbol{p_2}, \dots, \boldsymbol{p_m}$. The minimum Euclidean norm point problem requires computing the point $x \in \mathcal{P}$ of minimum Euclidean norm, or minimum norm point for short. More formally, we wish to solve the following problem:*

$$\begin{aligned} \text{argmin} \quad & \|\boldsymbol{x}\|_2 \\ s.t. \quad & \boldsymbol{x} = \sum_{k=1}^{m} \lambda_k p_k \end{aligned}$$

$$\sum_{k=1}^{m} \lambda_k = 1$$

$$\lambda_k \geq 0 \qquad\qquad \textit{for all } k \in \{1, \dots, m\}$$

The minimum norm point problem is intimately related to the complexity of linear programming. In fact, any linear program can be polynomially reduced to the minimum norm point problem Fujishige et al. (2006): this means that this apparently simple geometric problem is not only useful for submodular minimization, but it is also relevant to the theory of algorithm complexity of linear optimization.

The Fujishige-Wolfe algorithm was brought back to attention when Fujishige et al. (2006) and Fujishige & Isotani (2011) discovered encouraging computational results regarding the minimum norm point algorithm, which significantly outperformed all known *provably* polynomial time methods. For a while, it wasn't known whether Fujishige-Wolfe allowed a sub-exponential bound on the theoretical runtime, both with general polytopes and in the base polytope case. However, a breakthrough result by Chakrabarty et al. (2014) finally gives a pseudo-polynomial bound on the runtime of Fujishige-Wolfe for SFM.

Theorem 3.2 (Theorem 1 of Chakrabarty et al. (2014))**.** *Let $f : 2^{\mathcal{V}} \to \mathbb{Z}$ be an integer-valued set submodular function. Let $F := \max_{e \in \{1,\dots,n\}}(|f(\{e\})|, |f(\{1,\dots,n\}) - f(\{1,\dots,n\} \setminus \{e\})|)$. The Fujishige-Wolfe algorithm returns the minimizer of f in $\mathcal{O}(F^2(n^5 EO + n^7))$ time.*

Even though this result seems to shows that Fujishige-Wolfe has a theoretically worse dependency on n than Iwata & Orlin (2009) algorithms do and that it is dependent on F, benchmarks from Chakrabarty et al. (2014) show that the runtime of Fujishige-Wolfe seems to be independent of F and to have a better dependence on n than the combinatorial algorithms for SFM. Moreover, an $\mathcal{O}(\frac{1}{n^2})$-approximate solution to the minimum norm point of $\mathcal{B}(f)$ implies *exact* submodular minimization (Chakrabarty et al., 2014).

3.1.2 Constrained Submnodular Function Minimization can be Hard

Even though there are a plethora of alternative algorithms for the SFM problem, many related problems with additional constraints are NP-Hard to solve. *Cardinality constrained* SFM is one such example, where we want to restrict ourselves to sets $S \subseteq \mathcal{V}$ with cardinality $|S| = k$.

3.2 Submodular Set Maximization

The problem of maximizing a set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$, i.e.,

$$\max_{S \subseteq \mathcal{V}} f(S), \tag{3.2}$$

is NP-Hard, as it generalizes the Max Cut problem. Eq. (3.2) is known as Unconstrained Submodular Maximization (USM). Notice that, if f is monotone, USM is solvable in constant time: the optimal value will surely be $f(\mathcal{V})$. If f is non monotone however, one can either resort to the naive enumeration of the 2^n possible values of $f(S)$ and pick the set that gives the highest value (in exponential time), or devise smarter heuristic algorithms that are guaranteed to work in polynomial time, at the risk of returning a result that is slightly distant from the optimum. Before proceeding further, we drop the *EO* term in the runtime complexities, as it is not a convention followed by the submodular maximization community. We thus present the theoretical runtimes $\mathcal{O}(\cdot)$ assuming that the actual runtime is the cost of accessing the value oracle for f multiplied by $\mathcal{O}(\cdot)$.

3.2.1 Independence Systems and Matroid Constraints

One of the most flexible and well-known type of constraints in submodular optimization is the family of *independence systems* and *matroid constraints*. Independence systems, in particular, generalize the notion of *linear independece* of vectors. We now present some definitions that will be useful later.

Definition 3.3 (Independence system)**.** *A (finite) independence system is a pair $(\mathcal{V}, \mathcal{I})$ where $\mathcal{V}$ is a (finite) ground set and $\mathcal{I}$ is a family of subsets of $\mathcal{V}$. The family $\mathcal{I}$ must satisfy the following properties:*

- *$\emptyset \in \mathcal{I}$ (independence of the empty set);*
- *$I_1 \subseteq I_2 \in \mathcal{I} \Rightarrow I_1 \in \mathcal{I}$ (the subset of each independent set is also independent).*

The sets in $\mathcal{I}$ are called independent sets*, whereas all other subsets of $\mathcal{V}$ are called* dependent sets*. A maximal independent set, i.e., a set which becomes dependent when any item in $\mathcal{V}$ is added, is called a* basis *(of $\mathcal{V}$).*

Definition 3.4 (p-system)**.** *A p-system is an independence system $(\mathcal{V}, \mathcal{I})$ where, if $A, B \in \mathcal{I}$ are two maximal sets, then $|A| \leq p|B|$.*

Matroids are independence systems where the family of subsets $\mathcal{I}$ has an additional "exchange" property: if there are two independent sets such that one has fewer elements than the other, an element of the biggest set can be moved to the smallest set to expand the latter. Matroids were introduced into combinatorics by Whitney (1935) in the paper that characterized the "abstract properties of linear independence". Between the 1960's and the 1970's, Edmonds and Fulkerson discovered important applications of matroids to combinatorial and submodular optimization (Edmonds & Fulkerson, 1965; Edmonds, 1965; 1970). Since then, research in this area has been very active, especially in the field of submodular maximization.

Definition 3.5 (Matroid)**.** *A matroid $\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is an independence system where $\mathcal{I}$ has the additional "exchange" property:*

- *If $A, B \in \mathcal{I}$ and $|A| < |B|$, there is an element $b \in B \setminus A$ such that $A \cup \{b\} \in \mathcal{I}$.*

In plain words, if we have two independent sets such that one has fewer elements than the other, then we can expand the smallest set using one element of the biggest set, preserving the independence of the smallest set.

Any matroid qualifies as a 1-independence system. The intersection of p-matroids (which is not necessarily a matroid) qualifies as a p-independence system.
There are several examples of matroids. The most commonly mentioned in the literature of set submodular optimization are the *uniform* and *partition* matroids.

Definition 3.6 (Uniform Matroid)**.** *A* uniform matroid *$\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is the family of all subsets of size at most k, for some $k \in \mathbb{Z}_+$.*

Definition 3.7 (Partition Matroid)**.** *A* partition matroid *$\mathcal{M} = (\mathcal{V}, \mathcal{I})$ is a collection of disjoint sets B_i and integers $0 \leq k_i \leq |B_i|$ where a set A is independent if, for every index i, it holds that $|A \cap B_i| \leq k$.*

Another fundamental definition is the *rank* of a matroid, which is an important subclass of set submodular functions, and that sometimes influences the running time of the algorithms (e.g., in Filmus & Ward (2012)).

Definition 3.8 (Rank of a Matroid)**.** *Given a matroid $\mathcal{M} := (\mathcal{V}, \mathcal{I})$, its* rank *is the maximum size of an independent set in $\mathcal{M}$.*

Remark. *If $\mathcal{M} := (\mathcal{V}, \mathcal{I})$ is a* uniform matroid*, then the rank of $\mathcal{M}$ is $\rho(S) := \min\{k, |S|\}$, where S is any subset of $\mathcal{V}$.*

The submodular maximization problem subject to an independence system constraint is defined as follows. Let $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ be a non-negative normalized set submodular function. Let $(\mathcal{V}, \mathcal{I})$ be and independence system where $\mathcal{V}$ is a (finite) ground set and $\mathcal{I}$ is a family of subsets of $\mathcal{V}$. We assume that $\mathcal{I}$ is specified via a membership oracle which can be queried polynomially many times. A membership oracle query returns whether a given set $S \subseteq \mathcal{V}$ is independent. The problem we consider is:

$$\max_{S \subseteq \mathcal{V}} \{f(S) \mid S \in \mathcal{I}\}.$$

Remark. *In the formulation above, $(\mathcal{V}, \mathcal{I})$ can be either a matroid or a p-independence system.*

There are several examples of instances in which matroids are used to model complex constraints. The most recurrent example of such problems is the *subset selection* problem, which involves a single uniform matroid constraint. Another practical application is the *submodular welfare* problem, where a set $\mathcal{R}$ of items should be partitioned among a ground set $\mathcal{V}$ of n agents, each of which is modeled by a submodular utility function $f_i : 2^{\mathcal{R}} \to \mathbb{R}_+$, and the goal is to find the partition of the set $\mathcal{R}$ that maximizes $\sum_i f_i$ for each agent i in the partition. This submodular welfare maximization problem is modeled using a partition constraint, and it was first addressed in Fisher et al. (1978). Interestingly, Fisher et al. (1978) proposed a $\frac{1}{2}$-approximation greedy algorithm for this problem that also holds in the online setting, i.e., when $\mathcal{R}$ is not known *a priori*, but it is gradually revealed throughout the execution of the algorithm. Vondrak (2008) later proposed a randomized algorithm with a $(1 - \frac{1}{e})$-approximation guarantee.

3.2.2 Cardinality Constraints

The submodular maximization problem subject to cardinality constraints is defined as follows. Let $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ be a non-negative normalized set submodular function. Let $k \in \mathbb{Z}_+$ be an integer such that $1 \leq k < n$. The problem we consider is

$$\max_{S \subseteq \mathcal{V}} \{f(S) \mid |S| \leq k\}.$$

Remark. *A cardinality constraint is equivalent to a uniform matroid constraint. However, in some cases, it may by "cardinality constraint", researchers mean that the goal is choosing* exactly *k elements from $\mathcal{V}$, i.e., $|S| = k$.*

Subset selection problems subject to cardinality constraints arise frequently in combinatorial optimization, and are typically modeled by maximizing a submodular function. Typical examples are the *Max-k-Coverage* (Feige, 1998; Khuller et al., 1999) and *Max-Bisection* (Austrin et al., 2016; Frieze & Jerrum, 1997) problems, but cardinality constraints also emerge in natural language processing (Lin & Bilmes, 2011), sensor placement (Frieze, 1974; Krause et al., 2006; Xu et al., 2013), and information retrieval (Krause & Guestrin, 2007). In the context of *document summarization* in Lin & Bilmes (2011), for instance, the goal is to retrieve a small sample of words and phrases from a large corpus to form a summary, whose length is bounded by a given cardinality constraint. Such a constraint is natural to consider whenever the size of the body of text is too massive to analyse by a human.

3.2.3 Knapsack Constraints

The submodular maximization problem subject to knapsack constraints is defined as follows. Let $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ be a non-negative normalized set submodular function. Let $\boldsymbol{w}^1, \boldsymbol{w}^2, \dots, \boldsymbol{w}^l$ be l weight vectors corresponding to knapsacks having capacities $c_1, c_2, \dots, c_l$, respectively. The problem we consider is:

$$\max_{S \subseteq \mathcal{V}} \{f(S) \mid \sum_{j \in S} w^i_j \leq c_i, \text{ for all } 1 \leq i \leq l\}.$$

In this setting, the following assumptions are commonly considered:

- $c_i = 1$ for all $i \in \{1, \dots, l\}$, by scaling each knapsack accordingly;
- all weights $\boldsymbol{w}^1, \boldsymbol{w}^2, \dots, \boldsymbol{w}^l$ are rational;
- every $i \in \mathcal{V}$ is feasible for the knapsacks (otherwise, such elements can be dropped from the ground set w.l.o.g.).

The literature on submodular maximization algorithms is impressively large, therefore we only mention some of the most relevant works below.

3.2.4 Constrained Monotone Submodular Maximization

Without a doubt, the most celebrated result pertaining to submodular function maximization is due to Nemhauser et al. (1978), who showed a discrete greedy algorithm yields a constant $(1 - \frac{1}{e})$-approximation to the problem of maximizing a monotone submodular function under a cardinality constraint. In the context of submodular maximization, this procedure is simply referred to as the Greedy algorithm. The Greedy algorithm starts with the empty set $S_0 = \emptyset$, iteratively adding one element at a time for k iterations. At each step, it picks the unselected element increasing the value of the current solution the most, i.e., the element with the largest marginal value w.r.t. the current solution. We show such an algorithm in Line 1. The approximation ratio of Greedy is tight, i.e., it is the best possible performance guarantee for the considered problem both in the value oracle model and independently of $P \neq NP$ (Feige, 1998). Other combinatorial approaches to solve submodular maximization subject to some specific independence systems and matroid constraints include Conforti & Cornuejols (1984); Hausmann & Korte (1978); Hausmann et al. (1980).

Minoux (1978) proposed an alternative version of the Greedy algorithm that leverages submodularity to speed up the algorithm in practice (without theoretical runtime guarantees), while retaining the same approximation ratio. This alternative algorithm is called Lazy Greedy, and instead of computing the marginal gain $f(e \mid S_{j-1})$ for each element $e \in \mathcal{V}$, it just keeps an upper bound ρ_e (initially set to $+\infty$) on the gain sorted in decreasing order. At each step, Lazy Greedy selects the element e at the top of the list and updates its upper bound such that $\rho_e \leftarrow f(e \mid S_{j-1})$. If, after this update, it holds that $\rho_e \geq \rho_{e'}$ for every $e' \neq e$, submodularity implies that e is the actual element with the largest marginal gain w.r.t. the current solution. Unfortunately, Lazy Greedy may be impractical to use as the ground set size increase, even for small values of k. To overcome this problem, Mirzasoleiman et al. (2015) devised the Stochastic Greedy algorithm, which achieves a $(1 - \frac{1}{e})$-approximation ratio (on average) in $\mathcal{O}(n \log(\frac{1}{\varepsilon}))$ (for a constant $\varepsilon > 0$) and claims to be "the first algorithm that truly scales to voluminous datasets". The algorithm is shown in Line 2.

The major drawback of combinatorial algorithms for submodular maximization is that they are usually specifically tailored for the structure of the explored problem, making them difficult to extend and use for other related problems. Continuous relaxations are a popular way of overcoming this obstacle. This approach is two-folded: in the first step, the continuos algorithm finds a factional solution $\boldsymbol{x} \in \mathbb{R}^{\mathcal{V}}$ for a relaxed version of the problem; in the second step, $\boldsymbol{x}$ is rounded to obtain an integral solution (i.e., an indicator vector $\mathbf{1}_S \in \mathbb{Z}_+^{\mathcal{V}}$ for some set $S \subseteq \mathcal{V}$), with a negligible loss in the objective value. Vondrak (2008) introduced the Continuous Greedy algorithm, a randomized procedure that, given a matroid $\mathcal{M} := (\mathcal{V}, \mathcal{I})$ and the multilinear extension $F : [0,1]^{\mathcal{V}} \to \mathbb{R}$ of a monotone set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}$, returns a convex combination of independent sets in the matroid polytope $\mathcal{P}(\mathcal{M})$ that (with high probability) is an $(1 - \frac{1}{e})$-approximation of the optimum for the submodular welfare problem. Continuous Greedy approximately solves $\max\{F(\boldsymbol{y}) \mid \boldsymbol{y} \in \mathcal{P}(\mathcal{M})\}$ by moving in the direction of a feasible vector $\boldsymbol{y} \in \mathcal{P}(\mathcal{M})$ with the largest local marginal gain. Using rounding methods like the *pipeage rounding* technique (Ageev & Sviridenko, 2004; Gandhi et al., 2006) allows to convert a fractional solution returned by Continuous Greedy to a discrete solution $S \subseteq \mathcal{V}$, preserving the approximation ratio of $(1 - \frac{1}{e})$. Continuous Greedy has later been used to solve general submodular maximization problems subject to matroid constraints with the same tight approximation guarantees in Calinescu et al. (2011), and to obtain a $(1 - \frac{1}{e})$-approximation ratio for a constant number of knapsack constraints in Kulik et al. (2009). Continuous Greedy's runtime is reported to be very high for a polynomial algorithm: $\Theta(n^8)$ Filmus & Ward (2012). Feldman et al. (2011) also built on Continuous Greedy to propose the Measured Continuous Greedy algorithm, which gives a unified continuous approach for both monotone and non-monotone set submodular functions and improves the approximation ratio for some particular constrained problems. Moreover, Kulik et al. (2009) studied the problem of maximizing a monotone submodular function subject to multiple linear constraints, and Filmus & Ward (2012) proposed a combinatorial $(1 - \frac{1}{e})$-approximation algorithm for submodular maximization subject to a single matroid with $\mathcal{O}(n\rho^4\varepsilon^{-3})$ running time, where ρ is the rank of the matroid (which yields a performance similar to the Continuous Greedy algorithm in practice).

Furthermore, Badanidiyuru & Vondrák (2014) proposed the Decreasing Threshold Greedy framework, a discrete algorithm that claims to be *a smooth interpolation between the Continuous Greedy and Greedy algorithms*, which also presents *geometrically decreasing thresholds* inspired by the Lazy Greedy algorithm. Essentially, Lazy Greedy maintains a continuously decreasing

Algorithm 3.1: The Greedy algorithm for maximizing a monotone normalized set submodular function subject to cardinality constraints.

```
Input : Non-negative set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}_+$, cardinality constraint $r$
        with $r \in \mathbb{Z}_+, r \leq n$.
1 $S_0 \leftarrow \emptyset$;
2 for $j = 1$ to $r$ do
3     Let $e_j \in \mathcal{V} \setminus S_{j-1}$ be an element maximizing $f(e \mid S_{j-1})$;
4     $S_j \leftarrow S_{j-1} \cup \{e_j\}$;
5 return $S_r$;
```

Algorithm 3.2: The Stochastic Greedy algorithm for maximizing a monotone normalized set submodular function subject to cardinality constraints.

```
Input : Non-negative set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}_+$, cardinality constraint $r$
        with $r \in \mathbb{Z}_+, r \leq n$.
1 $s \leftarrow (n/r)\log(1/\varepsilon)$;
2 $S_0 \leftarrow \emptyset$;
3 for $j = 1$ to $r$ do
4     Let $R_j$ be a random subset obtained by sampling $s$ random elements from $\mathcal{V} \setminus S_{j-1}$;
5     Let $e_j \in R_j$ be an element maximizing $f(e \mid S_{j-1})$;
6     $S_j \leftarrow S_{j-1} \cup \{e_j\}$;
7 return $S_r$;
```

threshold and selects elements only when their marginal gain is above this threshold. The idea of Badanidiyuru & Vondrák (2014) is that, by decreasing the threshold, multiplying it by $(1 - \varepsilon)$ and including elements with a marginal gain above the threshold, one can avoid the local search of the maximal marginal gain, saving a factor of $\mathcal{O}(n)$ in the running time. The threshold Θ is initialized $d := \max_{e \in \mathcal{V}} f(e)$, the maximal singleton value.

Algorithm 3.3: The Random Greedy algorithm for maximizing a non necessarily monotone normalized set submodular function subject to cardinality constraints.

```
Input : Non-negative set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}_+$, cardinality constraint $r$
        with $r \in \mathbb{Z}_+, r \leq n$.
1 $S_0 \leftarrow \emptyset$;
2 for $j = 1$ to $r$ do
3     Let $M_j \subseteq \mathcal{V} \setminus S_{j-1}$ be a subset of size $r$ maximizing $\sum_{e \in M_j} f(e \mid S_{j-1})$;
4     Sample $u_j$ uniformly from $M_j$;
5     $S_j \leftarrow S_{j-1} \cup \{e_j\}$;
6 return $S_r$;
```

3.2.5 Constrained Non-Monotone Submodular Maximization

We start by noting that the Greedy algorithm has no constant approximation guarantee when the submodular objective function f subject to cardinality constraint k is non-monotone. For instance, consider the non-monotone set submodular function $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ defined as follows:

$$f(S) = \begin{cases} |S| & \text{if } n \notin S \\ 2 & \text{otherwise.} \end{cases}$$

Let $2 \leq k < n$. In the first step of the algorithm, Greedy adds n to the current solution, as it has a marginal gain of 2, the largest among the other elements. Notice that n remains in the solution until the end of the execution. If S_k is the returned solution, $f(S_k) = 2$, even though the optimal solution can take any k elements other than n to get a value k. Hence, in this non-monotone case Greedy has an approximation ratio of at most $\frac{2}{k}$. To work around this problem, Buchbinder et al. (2014) introduced the discrete Random Greedy algorithm, which achieves a $\frac{1}{e}$-approximation ratio (on average) by introducing randomization into the selection logic. As a side effect, Random Greedy achieves a $(1-\frac{1}{e})$-approximation ratio when f is monotone. We show the random greedy algorithm in Line 3.

Vondrák (2009) proposed a framework for maximizing non-monotone submodular functions subject to matroid constraints. In particular, he obtained a $\frac{1}{4}$-approximation for a matroid with 2 bases. Feldman et al. (2011) introduced a $(\frac{1}{e} - \varepsilon)$-approximation for maximizing a general non-negative submodular function subject to $\mathcal{O}(1)$-knapsack constraints, with constant $\varepsilon > 0$. Feldman et al.'s algorithm relies on the multilinear extension, just like Lee et al.'s one. The first (Feldman et al., 2011), whereas the second presented a $(\frac{1}{p+2+\frac{1}{p}+\varepsilon})$-approximation for the non-monotone maximization problem under a p-system, and a $(\frac{1}{5} - \varepsilon)$-approximation for k knapsack constraints (Lee et al., 2009). Mirzasoleiman et al. (2016) achieved a $(1+\varepsilon)(p+1)(2p+2l+1)/p$-approximation guarantee for a non-necessarily monotone submodular function maximization problem subject to an intersection of a p-system and l-knapsack constraints (in other words, for each knapsack c_i it holds that $\sum_{e \in S} c_i(e) \leq 1$, where $1 \leq i \leq l$). It does so with a runtime of $\mathcal{O}(\frac{nrp\log n}{\varepsilon})$, where $r \in \mathbb{Z}_+$ indicates the size of the largest feasible solution, demonstrating practical applications with experiments on *personalized data summarization*. Finally, Shi et al. (2021) obtained a $\frac{1}{2p+3+\frac{1}{p}}$-approximation ratio with a theoretical runtime of $\mathcal{O}(nrp)$.

3.2.6 Unconstrained Non-monotone Submodular Maximization

Feige et al. (2007) was the first to give constant approximation guarantees for the USM problem in the non-monotone case: a deterministic local search $\frac{1}{3}$-approximation algorithm, and a randomized $\frac{2}{5}$-approximation strategy. Buchbinder et al. (2012) introduced a randomized $\frac{1}{2}$-approximation algorithm that not only hits the best possible theoretical approximation guarantee for this problem, but also does that in linear time w.r.t. the number of value oracle queries: $\mathcal{O}(n)$. One could wonder whether there is a deterministic algorithm that achieves the same results as

Buchbinder et al. (2012) [1]. In an attempt to address this question, Dobzinski & Mor (2015) proposed the first deterministic algorithm that does better than a $\frac{1}{3}$-approximation, reaching a performance guarantee of $\frac{2}{5}$. Finally, in 2018, Buchbinder & Feldman (2018) proposed a deterministic algorithm that achieves the optimal deterministic $\frac{1}{2}$ guarantee in $\mathcal{O}(n^2)$. Resorting to randomization, then, seems to come at a cost of an increase in the runtime of the algorithm.

3.3 Concluding Remarks

In this chapter, we have summarized some of the most relevant algorithms that emerged from set submodular minimization and maximization in the centralized case, and with the value oracle assumption.

We believe that there is still room for improvement for the submodular function minimization problem. Every existing combinatorial algorithm based on the framework of Cunningham (1985) checks whether the current iterate $\boldsymbol{y}$ belongs to the base polyhedronq $\mathcal{B}(f)$ by representing $\boldsymbol{y}$ as a weighted sum with weights λ_e for vertices v_e emerging from the simple greedy algorithm. This does not seem to be much better than a brute-force search to verify that $\boldsymbol{y} \in \mathcal{B}(f)$, but over 35 years of research on this particular problem have not produced any better idea. Moreover, the upper bounds on the runtime of the Fujishige-Wolfe algorithm do not seem to match the practical results, which are significantly better than the other combinatorial algorithms for SFM.

Set submodular function maximization seems to be the most explored class of problems in the literature, w.r.t. SFM. What is particularly fascinating about submodular maximization is that, in some cases, randomization and continuous relaxations with the multilinear extension give better results than the deterministic discrete counterparts, especially when the problem is subject to complex constraints. The Stochastic Greedy algorithm due to Mirzasoleiman et al. (2015), in particular, seems to be the most promising algorithm for the set submodular maximization subject to a cardinality constraint.

We have purposedly not included any discussion on submodular lattice optimization, as it will be briefly discussed in the next chapter, where we will also present our novel results in lattice submodular maximization subject to a cardinality constraint.

[1]This is not a trivial question: derandomization of submodular maximization algorithms are complex because of the interaction with the value oracle, which prevents standard derandomization techniques such as conditional expectations.

4 Stochastic Submodular Function Maximization on the Integer Lattice

Optimization problems with set submodular objective functions are motivated by intuitive properties, like diminishing returns, and find many real-world applications. In discrete scenarios where the same item can be selected more than once, the domain of the target problem is generalized from a finite set to a bounded integer lattice. In this work, we consider the problem of maximizing a monotone submodular function on a bounded integer lattice subject to a cardinality constraint. In particular, we focus on maximizing DR-submodular functions, i.e., functions defined on the integer lattice that exhibit the same *diminishing returns* property that characterizes set submodular functions, which allows for a more efficient approximation. Given any small $\varepsilon > 0$, we present a family of randomized $(1 - \frac{1}{e} - \varepsilon)$-approximation algorithms (in expectation) using a framework inspired by a Stochastic Greedy algorithm developed for set submodular functions (Mirzasoleiman et al., 2015). We then demonstrate that, on realistic instances, applying the proposed algorithms on the integer lattice domain directly performs better than reducing the problem to the set domain and then apply the fastest known set submodular maximization algorithm, while retaining the same approximation ratio. Moreover, we show that as the size of the ground set and the available quantities increase, our proposed algorithms are either more efficient and stable than the two known fastest algorithms for the class of problems we study. We then empirically validate our experiments on both synthetic and real-world DR-submodular functions, in particular in the context of the *optimal budget allocation* problem.

This chapter is built upon our joint paper:

- Alberto Schiabel, Vyacheslav Kungurtsev, and Jakub Mareček (2022). "Randomized Algorithms for Monotone Submodular Function Maximization on the Integer Lattice". arxiv:2111.10175. Submitted to: *Proceedings of the 23rd Conference on Integer Programming and Combinatorial Optimization* (IPCO-22)[1].

4.1 Introduction

In Chapter 2 we have seen that problems with submodular objectives are ubiquitous in several quantitive disciplines, including combinatorial optimization, machine learning, and operations research. For instance, applications of submodular function optimization involve the analysis of

[1]https://www.ipco2022.com/home

influence models in social networks (Kempe et al., 2003) and optimal budget allocation for advertising products (Alon et al., 2012). In these problems, the goal is to pick a subset of a ground set $\mathcal{V} := \{1, 2, \dots, n\}$ that either maximizes or minimizes a set function f defined on the vertices of the unit hypercube $\{0,1\}^{\mathcal{V}}$. Yet, there are practical applications in which one is not only interested in knowing whether an element $e \in \mathcal{V}$ is selected, but is also concerned with the amount of the selected element. One such case is represented by the optimal budget allocation problem (Soma et al., 2014). In these scenarios, the ground set can be considered as a multiset, or equivalently as a cube $\{\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}} \mid \boldsymbol{x} \preccurlyeq b\}$ on the integer lattice $\mathbb{Z}_+^{\mathcal{V}}$, where $\boldsymbol{b} \in \mathbb{Z}_+^{\mathcal{V}}$ is a known vector such that b_e indicates the quantities available for each element $e \in \mathcal{V}$ and $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$ means $x_e \le y_e$ for every element $e \in \mathcal{V}$.
Any set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ can be transformed into a pseudo-Boolean function $\phi : \{0,1\}^{\mathcal{V}} \to \mathbb{R}$ defined on the Boolean lattice $(\{0,1\}^{\mathcal{V}}, \wedge, \vee)$ (Crama & Hammer, 2011), hence submodular optimization performed on the integer lattice $\mathbb{Z}^{\mathcal{V}}$ can be seen as a natural generalization of optimization on the Boolean lattice. There are, however, some important differences to highlight.

We recall that a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is called set *submodular* iff

$$f(A) + f(B) \ge f(A \cup B) + f(A \cap B) \quad \text{for all } A, B \subseteq \mathcal{V} \tag{4.1}$$

for all $A, B \subseteq \mathcal{V}$ (Definition 2.3). When f is a set function, Eq. (4.1) is equivalent to the intuitive *diminishing returns* (DR) property, which is particularly common when modeling utility functions in economical and game theoretical settings: a set function is submodular iff $\forall A \subseteq B \subseteq \mathcal{V}$ and $\forall e \in \mathcal{V} \setminus B$ it holds that

$$f(A \cup \{e\}) - f(A) \ge f(B \cup \{e\}) - f(B). \tag{4.2}$$

Eq. (4.2) is usually referred to as *set DR-submodularity*. However, in $\mathbb{Z}^{\mathcal{V}}$, *lattice submodularity is a weaker property than DR-submodularity*, and *DR-submodularity generally allows for more efficient maximization algorithms* (Soma et al., 2014).

A function $f : \mathbb{Z}^{\mathcal{V}} \to \mathbb{R}$ is said to be *integer-lattice submodular* if

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \ge f(\boldsymbol{x} \wedge \boldsymbol{y}) + f(\boldsymbol{x} \vee \boldsymbol{y}) \tag{4.3}$$

for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_+^n$. On the other hand, $f : \mathbb{Z}^{\mathcal{V}} \to \mathbb{R}$ is called *DR-submodular* if

$$f(\boldsymbol{x} + \mathbf{1}_e) - f(\boldsymbol{x}) \ge f(\boldsymbol{y} + \mathbf{1}_e) - f(\boldsymbol{y}) \tag{4.4}$$

for all $e \in \mathcal{V}$, and for all $\boldsymbol{x}, \boldsymbol{y} \in Z_+^n$ such that $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$. Properties (4.3) and (4.4) are sometimes known as *weak* and *strong DR-submodularity*, respectively (Sahin et al., 2020b).

In this work, we focus on *monotone* submodular function maximization on the integer lattice subject to a cardinality constraint. More precisely, consider the maximization problem:

$$\begin{aligned} \max \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \|\boldsymbol{x}\|_1 \le r \\ & x_e \le b_e \qquad\qquad \text{for all } e \in \mathcal{V} \\ & \boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}, \boldsymbol{b} \in \mathbb{Z}_+^{\mathcal{V}}, r \in \mathbb{Z}_+, \end{aligned} \tag{4.5}$$

where $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ is a monotone submodular function defined on the integer lattice, $\boldsymbol{x} \in \mathbb{Z}^{\mathcal{V}}$ is defined such that $x_e \in \mathbb{Z}_+$ determines how many copies of an element $e \in \mathcal{V}$ should be selected, $\boldsymbol{b} \in \mathbb{Z}^{\mathcal{V}}$ is a known vector where $b_e \in \mathbb{Z}_+$ represents how many copies of an element $e \in \mathcal{V}$ are available in the ground set, and $r \in \mathbb{Z}_+$ denotes the maximum cardinality of a feasible solution. We observe that it should also hold that $r < \|\boldsymbol{b}\|_1$, otherwise Eq. (4.5) becomes an unconstrained submodular maximization problem, which is trivially solved in constant time since f is monotone[2].

As usual, we assume that $f : \mathbb{Z}^{\mathcal{V}} \to \mathbb{R}$ is given via a *value oracle* black box, i.e., given some feasible $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$, the oracle returns $f(\boldsymbol{x})$. We refer to each oracle invocation as an *oracle query*. This value oracle model is standard in submodular optimization, as it abstracts away the details of the specific problem in a generic algorithm that solves a submodular optimization problem.
We also assume that $b_e \geq 1$ for all $e \in \mathcal{V}$ (otherwise, the elements $e' \in \mathcal{V}$ such that $b_{e'} = 0$ can be removed from the ground set $\mathcal{V}$ w.l.o.g.). Moreover, notice that when $b_e = 1$ for all $e \in \mathcal{V}$, Eq. (4.5) is equivalent to a set submodular function maximization problem subject to a cardinality constraint, where we one can select a set $S \subseteq \mathcal{V}$ with at most $|S| = r$ elements.

Considering the wealth of algorithms for the optimization of set submodular functions, which are limited to express binary decisions (i.e., whether to select one element or not), it is natural to consider a *reduction* from the integer lattice setting to the set submodular setting that enables the use of popular set submodular maximization procedures like the Greedy algorithm in the integer lattice domain. The most natural one generates b_e elements in a new set $\mathcal{V}'$ for each element $e \in \mathcal{V}$ Soma & Yoshida (2015). The drawback is that this reduction yields a pseudo-polynomial-time algorithm in n, which negatively affects the runtime of the set submodular algorithms as the value of b_e grows for each e. For DR-submodular functions, Ene & Nguyen (2016) proposed another reduction to set-submodular optimization, which enacts a bit decomposition argument that yields a ground set $\mathcal{V}'$ of size $|\mathcal{V}'| = \mathcal{O}(\log \boldsymbol{b} + \frac{1}{\varepsilon}) \cdot n$.

The algorithm presented in this work is based on the Stochastic Greedy technique introduced by Mirzasoleiman et al. (2015), which is – in turn – based on the Greedy algorithm Nemhauser et al. (1978), which is perhaps the single most famous result in submodular function maximization. Indeed, Nemhauser et al. (1978) showed that a simple greedy approach that picks an unselected element at each iteration maximizing the local marginal gain provides a tight $(1-\frac{1}{e})$-approximation guarantee for maximizing a *monotone* submodular function subject to cardinality constraints.
Mirzasoleiman et al. (2015) devised an algorithm called Stochastic Greedy (SG for short), which greatly improves upon the running time of Greedy while retaining the same approximation ratio in expectation, using only $\mathcal{O}(n \log \frac{1}{\varepsilon})$ oracle queries. The SG algorithm uses a randomized subsampling technique that, for each of the r iterations, randomly selects a subset $Q \subseteq \mathcal{V}$ of a fixed size and finds the element $e \in Q$ that maximizes the marginal gain. The key difference between this approach and Greedy is that Q changes at each iteration. Thus, in expectation, SG does cover the entire dataset. In constrast, Greedy is equivalent to subsampling elements from $\mathcal{V}$ before looking for the most representative elements in it. Here, we extend the idea underlying the Stochastic Greedy algorithm from a set to an integer lattice domain, proving that our algorithms are both

[2] In the monotone case of unconstrained submodular function maximization, selecting $\boldsymbol{x} := \boldsymbol{b}$ guarantees that $f(\boldsymbol{x})$ is maximized.

practically and theoretically faster than Stochastic Greedy consider a reduction of an integer lattice problem to the set domain.

4.1.1 Our Results

Here, we present a family of polynomial-time algorithms for maximizing a monotone DR-submodular function defined on the integer lattice, improving upon the state of the art (Soma & Yoshida, 2018; Lai et al., 2019) in terms of practical run-time, while retaining the same $(1 - \frac{1}{e})$-approximation guarantee (in expectation). We also show that our algorithms are significantly more stable than the previous state of the art in terms of the number of oracle queries required to approximately solve the target problem.
In particular, our contribution comprises:

- a Stochastic Greedy Lattice (SGL for short) algorithm for maximizing monotone DR-submodular functions defined on the integer lattice subject to cardinality constraints;
- an analysis of the approximation ratio of SGL, whose expectation is arbitrarily close to the order of $(1 - \frac{1}{e})$, which is tight (Feige, 1998);
- numerical experiments on a synthetic class of problems which indicate the scalability of SGL, and show the instability of the algorithms of Soma & Yoshida (2018) and Lai et al. (2019), which are considered the current state of the art for the considered constrained maximization problem.
- numerical experiments of our algorithms on a *optimal budget allocation* problem instance using the Wikilens dataset (Frankowski et al., 2007) from the Koblenz Network Collection (Kunegis, 2013).

4.2 Background

We now recall some basic definitions we introduced for set submodular functions, and their counterparts for integer lattice functions.

A set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ is *monotone* if, for every $A \subseteq B \subseteq \mathcal{V}$, $f(A) \leq f(B)$. f is said to be *normalized* if $f(\emptyset) = 0$. The marginal gain obtained by adding an element $e \in \mathcal{V}$ to a set $S \subseteq \mathcal{V}$ is defined as $f(e \mid S) := f(S \cup \{e\}) - f(S)$. We now give the analogous definitions for the integer lattice case.

An integer lattice function $f : Z_+^{\mathcal{V}} \to \mathbb{R}$ is *monotone* if $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$ implies $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ for some $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{Z}_+^{\mathcal{V}}$, and f is *normalized* if $f(\mathbf{0}) = 0$. We only consider monotone and normalized functions. The marginal gain obtained by adding an element $e \in \mathcal{V}$ to a vector $\boldsymbol{x} \in Z_+^{\mathcal{V}}$ is defined as $f(\mathbf{1}_e \mid \boldsymbol{x}) := f(\boldsymbol{x} + \mathbf{1}_e) - f(\boldsymbol{x})$, where the sum operation is applied component-wise.

Given a known vector $\boldsymbol{b} \in Z_+^{\mathcal{V}}$ that defines the multiplicities of the elements $e \in \mathcal{V}$, the problem of maximizing an integer lattice function $f : Z_+^{\mathcal{V}} \to \mathbb{R}$ under a cardinality constraint $r \in \mathbb{Z}_+$ is formalized as selecting a vector $\boldsymbol{x} \in Z_+^{\mathcal{V}}$ whose entries x_e have value at most b_e and such that $\|\boldsymbol{x}\|_1 \leq r$.

4.2.1 Prior Work in Submodular Optimization on the Integer Lattice

Submodular optimization on the lattice domain is almost as old as set-submodular optimization itself Topkis (1978), but efficient algorithms for maximizing integer lattice submodular functions are much more recent, and only a handful of studies have considered this problem. In fact, until a few years ago, it was not even known whether it was possible to solve submodular maximization on the integer lattice subject to a cardinality constraint with a $(1-\frac{1}{e})$-approximation guarantee in polynomial time. Gottschalk & Peis (2015) presented a natural Double Greedy time algorithm, which is pseudopolynomial and attains an approximation ratio of $(1-\frac{1}{e})$. In the case of non-monotone submodular functions defined on the integer lattice without additional constraints, the same algorithm reaches a $\frac{1}{3}$ approximation.

Soma & Yoshida (2018) introduced the first discrete deterministic algorithm for maximizing monotone DR-submodular functions subject to a cardinality constraint with $\mathcal{O}(\frac{n}{\varepsilon}\log\|\boldsymbol{b}\|_\infty \log\frac{r}{\varepsilon})$ running time. We refer to this algorithm as Soma-DR-I. Soma & Yoshida (2018) also proposed another combinatorial algorithm for the more general case of maximizing monotone integer lattice submodular functions (without the *diminishing returns*) property, obtaining the significantly worse running time of $\mathcal{O}(\frac{n}{\varepsilon^2}\log\|\boldsymbol{b}\|_\infty \log\frac{r}{\varepsilon}\log\tau)$, where τ is the ratio of the maximum value of f to the minimum positive increase in the value of f. We refer to this algorithm as Soma-II. Both Soma-DR-I and Soma-II guarantee an $(1-\frac{1}{e}-\varepsilon)$-approximation for the considered problem setting. Soma & Yoshida's algorithms adapt the Decreasing Threshold Greedy framework introduced in Badanidiyuru & Vondrák (2014) to the integer lattice domain. The Soma-DR-I algorithm, in particular, uses a binary search procedure to reduce the oracle queries needed to find the local element with the maximal marginal gain w.r.t. the decreasing threshold's value $\Theta \in \mathbb{R}_+$. We show this algorithm in Algorithm 4.1.

Lai et al. (2019) also proposed two algorithms for the same problem: a *random greedy* algorithm for DR-submodular functions with a $(1-\frac{1}{e})$ approximation ratio that requires $\mathcal{O}((n+1)\cdot r)$ oracle queries, and a deterministic discrete algorithm that achieves only a $(\frac{1}{e})$ approximation guarantee in $\mathcal{O}(n\cdot\|\boldsymbol{b}\|_\infty\cdot r^3)$ oracle queries, but that does not assume the *diminishing returns* property. We refer to the first algorithm as Lai-DR, and we show it in Algorithm 4.2. We notice that Lai-DR requires solving an integer linear program (ILP) with a number of constraints that is polynomial in n, which is an NP-Complete problem by itself, even though several heuristics have been proposed throughout the years.

Moreover, Bach (2019) extended the multilinear extension (Calinescu et al., 2007) to DR-submodularity, and Sahin et al. (2020a) introduced the generalized multilinear extension of the integer-lattice optimization problems in a similar spirit, but without translating these to actual algorithms.

Other generalizations of set submodular functions, like *bisubmodular* and *k-submodular* functions have also been recently explored in the literature Ward & Živný (2016), leading to approximate maximization algorithms. Kuhnle et al. (2018); Qian et al. (2018) provided approximation algorithms for maximizing a non-submodular function on the integer lattice subject to a cardinality constraint. Maehara et al. (2021) consider multiple knapsack constraints on the distributive lattice. Sahin et al. (2020b) considered the optimization of DR-submodular functions subject to discrete polymatroid constraints. Bian et al. (2017a;b) were the first to consider the continuous variants of

Algorithm 4.1: The Soma-DR-I (Soma & Yoshida, 2018) deterministic greedy algorithm for maximizing a monotone normalized DR-submodular function defined on the integer lattice subject to cardinality constraints.

```
Input : Non-negative integer lattice DR-submodular function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}_+$, cardinality
        constraint $k$ with $k \in \mathbb{Z}_+, k \leq n$.
1  $\boldsymbol{x}^0 \leftarrow \boldsymbol{0}$;
2  $d \leftarrow \max_{e \in \mathcal{V}} f(\boldsymbol{1}_e)$;
3  $\Theta_{stop} \leftarrow \frac{\varepsilon}{r} \cdot d$;
4  for $\Theta \leftarrow d; \Theta \geq \Theta_{stop}; \Theta \leftarrow \Theta(1-\varepsilon)$ do
5      for $e \in \mathcal{V}$ do
6          $k_{max} \leftarrow \min\{b_e - x_e, r - \|\boldsymbol{x}\|_1\}$;
7          Find the maximum $k \in \mathbb{Z}_+$ with $1 \leq k \leq k_{max}$ such that $f(k \cdot \boldsymbol{1}_e \mid \boldsymbol{x}) \geq k \cdot \Theta^{t-1}$ via
             binary search;
8          if $\exists k^t$ as above then
9              $\boldsymbol{x} \leftarrow \boldsymbol{x} + k \cdot \boldsymbol{1}_e$;
10 return $\boldsymbol{x}$;
```

Algorithm 4.2: The Lai-DR (Lai et al., 2019) randomized greedy algorithm for maximizing a monotone normalized DR-submodular function defined on the integer lattice subject to cardinality constraints.

```
Input : Non-negative integer lattice DR-submodular function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}_+$, cardinality
        constraint $k$ with $k \in \mathbb{Z}_+, k \leq n$.
1  $\boldsymbol{x}^0 \leftarrow \boldsymbol{0}$;
2  for $j = 1$ to $r$ do
3      Let $\boldsymbol{m}^j \in \mathbb{Z}_+^{\mathcal{V}}$ be a vector that maximizes $\sum_{e \in \mathcal{V}} \boldsymbol{m}_e^j \cdot f(\boldsymbol{1}_e + \boldsymbol{x}^{j-1}) - f(\boldsymbol{x}^{j-1})$ and satisfies
         $\boldsymbol{0} \preccurlyeq \boldsymbol{m}^j \preccurlyeq \boldsymbol{b}$, $\|\boldsymbol{m}^j\|_1 \leq r$, $\boldsymbol{m}^j + \boldsymbol{x}^{j-1} \leq \boldsymbol{b}$;
4      Sample $e^j \in \mathcal{V}$ randomly with probability $\frac{m_{e^j}^j}{\|\boldsymbol{m}^j\|_1}$;
5      $\boldsymbol{x}^j \leftarrow \boldsymbol{x}^{j-1} + \boldsymbol{1}_{e^j}$;
6  return $\boldsymbol{x}^r$;
```

DR-submodular functions, which has become a popular research direction Niazadeh et al. (2020); Feldman & Karbasi (2020); Mokhtari et al. (2020).

4.2.2 Reduction from Lattice to Set

We are aware of two generic reductions from the integer lattice domain to the set submodular setting, allowing the use of constrained maximization algorithms that target set submodular functions. The most intuitive one (Soma & Yoshida, 2015) generates b_e copies for each element $e \in \mathcal{V}$, letting $\mathcal{V}'$ be the the multiset containing these copies. Thus, problem (4.5) can then be reformulated into a set function maximization problem with respect to the ground set $\mathcal{V}'$, and one can then use the SG algorithm to solve the problem. The drawback is that this reduction yields pseudo-

Algorithm 4.3: Binary-Search algorithm for finding the value $k \in \mathbb{Z}_+$ such that $k_{min} \leq k \leq k_{max}$ it maximizes $f(x + k \cdot \mathbf{1}_e) \geq k \cdot \Theta$.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$, current solution $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$,
         previous solution value $f_{\boldsymbol{x}} := f(\boldsymbol{x})$, decreasing threshold $\Theta \in \mathbb{R}_+$, $k_{min} \in \mathbb{Z}_+$,
         $k_{max} \in \mathbb{Z}_+$.
Output : Maximal $k \in \mathbb{Z}_+$.
1  $k^t_{best} \leftarrow 0$;
2  while $k_{min} \leq k_{max}$ do
3      $\hat{k}^t \leftarrow k^t_{max} - \lfloor \frac{k^t_{max} - k^t_{min}}{2} \rfloor$;
4      $\hat{\boldsymbol{x}}^t \leftarrow \boldsymbol{x}^{t-1} + \hat{k}^t \cdot \mathbf{1}_e$;
5      if $f(\hat{k}^t \cdot \mathbf{1}_e \mid \boldsymbol{x}^{t-1}) \geq \hat{k}^t \cdot \Theta^{t-1}$ then
6          $k_{min} \leftarrow \hat{k}^t + 1$;
7          if $k^t_{best} < \hat{k}^t$ then
8              $k^t_{best} \leftarrow \hat{k}^t$;
9      else
10         $k_{max} \leftarrow \hat{k}^t - 1$;
11 return $k^t_{best}$;
```

polynomial time algorithm, since $|\mathcal{V}'| = \mathcal{O}(\|\boldsymbol{b}\|_\infty \cdot n)$. We denote the SG algorithm simulated in the integer lattice domain by SSG.

Another, more efficient reduction algorithm is proposed by Ene & Nguyen (2016), but it can only be applied to DR-submodular functions. This reduction uses a bit decomposition argument that yields a ground set $\mathcal{V}'$ of size $|\mathcal{V}'| = \mathcal{O}((\log \|\boldsymbol{b}\|_\infty + \frac{1}{\varepsilon}) \cdot n)$. Similarly to the previously considered reduction, however, Ene & Nguyen (2016) results in a pseudo-polynomial algorithm. Moreover, we are not aware of any implementation of such reduction, and it was not clear to us how to implement it in practice. For this reason, we will only consider the intuitive reduction first presented by Soma & Yoshida (2015).

4.3 Our Algorithms

Let us consider the drawbacks of the known reduction algorithms to translate an integer lattice submodular function to a set submodular function, as explained above, and the unpredictable running time of Lai-DR (Lai et al., 2019), which requires solving an integer linear program, whose runtime could be much larger than the time needed to evaluate an oracle query. This motivated us to seek novel algorithms for submodular maximizing of integer lattice submodular functions subject to a cardinality constraint. In particular, we were inspired by the simplicity and performance of the Stochastic Greedy algorithm (Mirzasoleiman et al., 2015), which employs the random subsampling technique, while retaining a tight approximation ratio of $(1 - 1/e)$ on average for the set submodular version of the problem. We thus propose a novel algorithm, which is based on the same random subsampling idea, but combined with the Decreasing Threshold Greedy framework of Badanidiyuru & Vondrák (2014) and the *binary search* strategy of Soma & Yoshida (2018)[3].

Let $s := \lfloor \frac{n}{r} \cdot \log \frac{1}{\varepsilon} \rfloor$ indicate the sample size, as in Mirzasoleiman et al. (2015). Our proposed algorithms start with an empty solution vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ and with a decreasing threshold value Θ (which we use to decide whether to add new elements to $\boldsymbol{x}$) initialized with d, the maximum value of the function f over every singleton $\boldsymbol{e}$, for $e \in \mathcal{V}$.
Until the cardinality of $\boldsymbol{x}$ reaches r, we randomly sample s elements from the available elements in $\mathcal{V}$. Locally, we look for s distinct elements, but we allow sampling with replacement among different iterations. We then look for the maximum integer $k \in \mathbb{Z}_+$ such that the marginal gain $f(k \cdot \mathbf{1}_e \mid \boldsymbol{x})$ is greater or equal than $k \cdot \Theta$. The value of k indicates how many copies of an element $e \in \mathcal{V}$ are added to $\boldsymbol{x}$ (if any at all), so to make sure that we don't surpass the cardinality bound r, at each iteration t it should hold that $k \leq \min\{b_e - x_e, r - \|\boldsymbol{x}\|_1\}$ for any previous solution x^{t-1} and element $e \in \mathcal{V}$. If such value k exists and satisfies the constraint that $f(\boldsymbol{x} + k \cdot \mathbf{1}_e)$ increase the value f, then we add k copies of element e to our solution $\boldsymbol{x}$. The number of times this process is repeated depends on the specific algorithm.

We call this family of algorithms Stochastic Greedy Lattice x (SGL-x for short), where x is a letter in $\{a, b, c, d\}$.

4.3.1 SGL-*a*

SGL-a iterates until the cardinality of $\boldsymbol{x}$ reaches r. It performs a random sample of s elements at each iteration. We show this algorithm in Algorithm 4.4.

4.3.2 SGL-*b*

SGL-b iterates until the cardinality of $\boldsymbol{x}$ reaches r, or until Θ decreases below $\frac{\varepsilon}{r} \cdot d$. At iteration t, it first randomly shuffles the ground set $\mathcal{V}$ to obtain $\mathcal{V}^t$, which is then partitioned into subsets with cardinality at most s. The same procedure as SGL-a is then applied, but this algorithm has the guarantee that at each major iteration every element $e \in \mathcal{V}$ is considered once. We show this algorithm in Algorithm 4.5.

4.3.3 SGL-*c*

SGL-c iterates until the cardinality of $\boldsymbol{x}$ reaches r, or until Θ decreases below $\frac{\varepsilon}{r} \cdot d$. It is similar to SGL-$b$, but it performs an additional greedy step. At iteration t, it first collects every feasible (k^t, e) pair such that $f(k^t \cdot \mathbf{1}_e \mid \boldsymbol{x}) \geq k^t \cdot \Theta^{t-1}$. Then, if any such (k^t, e) exist for the current partition Q of at most s distinct elements, it adds k^{t*} copies of $e^* \in Q$, where (k^{t*}, e^*) is the local pair that yields the largest marginal gain for Q. We show this algorithm in Algorithm 4.6.

[3]We notice that Soma & Yoshida (2018) didn't specify how to implement the binary search procedure, and it may not be immediate to derive it slightly different than the classic "text book" implementation of a binary search; we thus show our implementation in Algorithm 4.3.

Algorithm 4.4: SGL-*a* algorithm for maximizing a monotone integer lattice function subject to cardinality constraint.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ (with $n := |V|$), vector of
         quantities $\boldsymbol{b} \in \mathcal{Z}_+^{\mathcal{V}}$, and cardinality constraint $r \in \mathbb{Z}_+$.
Output : Vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ that approximately solves $\max_{\boldsymbol{x}} f(\boldsymbol{x})$ s.t. $||x||_1 \leq r$.
1  $\boldsymbol{x}^0 \leftarrow \mathbf{0}$;
2  $d \leftarrow \max_{e \in \mathcal{V}} f(\mathbf{1}_e)$;
3  $\Theta^0 \leftarrow d$;
4  $\Theta_{stop} \leftarrow \frac{\varepsilon}{r} \cdot d$; $t \leftarrow 1$;
5  while $||\boldsymbol{x}^{t-1}||_1 < r$ do
6      $Q \leftarrow$ sample $s$ distinct elements from $\{e \in \mathcal{V} \mid x_e < b_e\}$;
7      for $e \in Q$ do
8          $k^t_{max} \leftarrow \min\{b_e - x_e^{t-1}, r - ||\boldsymbol{x}^{t-1}||_1\}$;
9          Find the maximum $k^t \in \mathbb{Z}_+$ with $1 \leq k^t \leq k^t_{max}$ such that $f(k^t \cdot \mathbf{1}_e \mid \boldsymbol{x}) \geq k^t \cdot \Theta^{t-1}$
             via binary search;
10         if $\exists\, k^t$ as above then
11             $\boldsymbol{x}^t \leftarrow \boldsymbol{x}^{t-1} + k^t \cdot \mathbf{1}_e$; // $f$ monotone $\Rightarrow$ $f(\boldsymbol{x}^{t-1} + k^t \cdot \mathbf{1}_e) \geq f(\boldsymbol{x}^{t-1})$
12     $\Theta^t \leftarrow \max\{\Theta^{t-1} \cdot (1 - \varepsilon), \Theta_{stop}\}$;
13     $t \leftarrow t + 1$;
14 return $\boldsymbol{x}^{t-1}$;
```

4.3.4 SGL-*d*

SGL-*d* loops until each batch of size s is iterated r times, or until $\boldsymbol{x}$ reaches the cardinality constraint r. It is similar to SGL-*c*, but it only decreases the threshold Θ when no feasible k is found for a given random partition Q of size at most s. Moreover, the threshold is rapidly decreased by a 0.5 factor, i.e., it is halved iteratively. We show this algorithm in Algorithm 4.7.

4.3.5 Comparison with other Algorithms

The major difference between our SGL-*x* algorithms and Soma-DR-I Soma & Yoshida (2018) is that at each iteration we only consider a subset $Q \subseteq \mathcal{V}$ of at most s elements, whereas Soma-DR-I applies our same binary search procedure to every element in the ground set $\mathcal{V}$, and that Soma-DR-I is fully deterministic. Moreover, we require $||\boldsymbol{x}||_1$ to reach the desired cardinality r, whereas Soma-DR-I only stops when the decreasing threshold Θ is too low. On the other hand, we do not observe similarities between SGL-*x* and Lai-DR, except the fact that both algorithms are randomized.

4.3.6 Additional Proposed Algorithms

We have also developed two simpler novel randomized algorithms for solving submodular function maximization on the integer lattice subject to a cardinality constraint. They are still influenced by the Stochastic Greedy Algorithm by (Mirzasoleiman et al., 2015), but, contrarily to our SGL-*x* algorithms, they do not use any binary search procedure or decreasing threshold frame-

Algorithm 4.5: SGL-*b* algorithm for maximizing a monotone integer lattice function subject to cardinality constraint.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ (with $n := |V|$), vector of
         quantities $\boldsymbol{b} \in \mathcal{Z}_+^{\mathcal{V}}$, and cardinality constraint $r \in \mathbb{Z}_+$.
Output : Vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ that approximately solves $\max_{\boldsymbol{x}} f(\boldsymbol{x})$ s.t. $||x||_1 \le r$.
1  $\boldsymbol{x}^0 \leftarrow \mathbf{0}$;
2  $d \leftarrow \max_{e \in \mathcal{V}} f(\mathbf{1}_e)$;
3  $\Theta^0 \leftarrow d$;
4  $\Theta_{stop} \leftarrow \frac{\varepsilon}{r} \cdot d$; $t \leftarrow 1$;
5  while $\Theta^{t-1} \ge \Theta_{stop}$ and $||\boldsymbol{x}^{t-1}||_1 < r$ do
6      $\mathcal{V}^t \leftarrow$ shuffle $\{e \in \mathcal{V} \mid x_e < b_e\}$;
7      $\mathcal{Q} \leftarrow$ split $\mathcal{V}^t$ into batches of size at most $s$;
8      for $Q \in \mathcal{Q}$ do
9          for $e \in Q$ do
10             $k^t_{max} \leftarrow \min\{b_e - x_e^{t-1}, r - ||\boldsymbol{x}^{t-1}||_1\}$;
11             Find the maximum $k^t \in \mathbb{Z}_+$ with $1 \le k^t \le k^t_{max}$ such that
                 $f(k^t \cdot \mathbf{1}_e \mid \boldsymbol{x}) \ge k^t \cdot \Theta^{t-1}$ via binary search;
12             if $\exists\, k^t$ then
13                 $\boldsymbol{x}^t \leftarrow \boldsymbol{x}^{t-1} + k^t \cdot \mathbf{1}_e$; // $f$ monotone $\Rightarrow$ $f(\boldsymbol{x}^{t-1} + k^t \cdot \mathbf{1}_e) \ge f(\boldsymbol{x}^{t-1})$
14         $\Theta^t \leftarrow \max\{\Theta^{t-1} \cdot (1 - \varepsilon), \Theta_{stop}\}$;
15         $t \leftarrow t + 1$;
16         if $||\boldsymbol{x}^{t-1}||_1 = r$ then
17             break
18 return $\boldsymbol{x}^{t-1}$;
```

work. We refer to these two new algorithms as SGL-I and SGL-II, and to their family as SGL-*n*, to distinguish it from the SGL-*x* algorithm family.

4.3.7 SGL-I

At each major iteration, SGL-I computes a random shuffling of the elements of the ground set $\mathcal{V}$ such that $x_e < b_e$ for each $e \in \mathcal{V}$, and then splits it into batches of size at most s., similarly to SGL-*b*, SGL-*c*, SGL-*d*. Then, for each batch Q, SGL-I adds to the solution $\boldsymbol{x}$ the feasible element e which maximizes the marginal gain $f(\mathbf{1}_e \mid \boldsymbol{x})$. The number of major iterations is bounded by r (notice that in SGL-*d*, it is the number of internal iterations to be bounded by r). SGL-I always add a single element at each internal iteration. We show this algorithm in Algorithm 4.8.

4.3.8 SGL-II

SGL-II is very similar to SGL-II, but it adopts slightly more drastic selection criterion in the internal iterations. For each batch Q, SGL-II first considers, for each element $e \in Q$, the highest possible cardinality k^e_{max} such that $f(k^e_{max} \cdot \mathbf{1}_e \mid \boldsymbol{x})$ holds and $\boldsymbol{x} + k^e_{max} \cdot \mathbf{1}_e$ is a feasible solution. It then adds to the solution k^e_{max} copies of the element e which maximizes the marginal gain $f(k^e_{max} \cdot \mathbf{1}_e \mid \boldsymbol{x})$.

Algorithm 4.6: SGL-*c* algorithm for maximizing a monotone integer lattice function subject to cardinality constraint.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ (with $n := |V|$), vector of
         quantities $\boldsymbol{b} \in \mathcal{Z}_+^{\mathcal{V}}$, and cardinality constraint $r \in \mathbb{Z}_+$.
Output : Vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ that approximately solves $\max_{\boldsymbol{x}} f(\boldsymbol{x})$ s.t. $\|x\|_1 \leq r$.
$\boldsymbol{x}^0 \leftarrow \boldsymbol{0}$;
$d \leftarrow \max_{e \in \mathcal{V}} f(\boldsymbol{1}_e)$;
$\Theta^0 \leftarrow d$;
$\Theta_{stop} \leftarrow \frac{\varepsilon}{r} \cdot d$; $t \leftarrow 1$;
while $\Theta^{t-1} \geq \Theta_{stop}$ and $\|\boldsymbol{x}^{t-1}\|_1 < r$ do
    $\mathcal{V}^t \leftarrow$ shuffle $\{e \in \mathcal{V} \mid x_e < b_e\}$;
    $\mathcal{Q} \leftarrow$ split $\mathcal{V}^t$ into batches of size at most $s$;
    for $Q \in \mathcal{Q}$ do
        $\Lambda^t \leftarrow \emptyset$;
        for $e \in Q$ do
            $k^t_{max} \leftarrow \min\{b_e - x_e^{t-1}, r - \|\boldsymbol{x}^{t-1}\|_1\}$;
            Find the maximum $k^t \in \mathbb{Z}_+$ with $1 \leq k^t \leq k^t_{max}$ such that
              $f(k^t \cdot \boldsymbol{1}_e \mid \boldsymbol{x}) \geq k^t \cdot \Theta^{t-1}$ using binary search;
            if $\exists\, k^t$ then
                $\Lambda^t \leftarrow \Lambda \cup \{(k^t, e)\}$; // $f$ monotone $\Rightarrow f(\boldsymbol{x}^{t-1} + k^t \cdot \boldsymbol{1}_e) \geq f(\boldsymbol{x}^{t-1})$
        if $\Lambda^t \neq \emptyset$ then
            $(k^{t*}, e^*) \leftarrow \operatorname{argmax}_{(k^t,e) \in \Lambda^t} f(k^t \cdot \boldsymbol{1}_{e^*} \mid \boldsymbol{x}^{t-1})$;
            $\boldsymbol{x}^t \leftarrow \boldsymbol{x}^{t-1} + k^t \cdot \boldsymbol{1}_{e^*}$;
        $\Theta \leftarrow \max\{\Theta \cdot (1 - \varepsilon), \Theta_{stop}\}$;
        $t \leftarrow t + 1$;
        if $\|\boldsymbol{x}^{t-1}\|_1 = r$ then
            break
return $\boldsymbol{x}^{t-1}$;
```

We show this algorithm in Algorithm 4.9.

4.4 An Analysis

We now develop guarantees for the proposed algorithm (SGL), both in terms of its approximation ratio and computational complexity expressed in terms of oracle queries.

4.4.1 Probabilistic Approximation Guarantees

We define the following set A_t corresponding to the random subsampling procedure of s elements at iteration t.

$$A_t := \operatorname{supp}(\max\{\boldsymbol{x}^* - \boldsymbol{x}_t, 0\}). \tag{4.6}$$

Algorithm 4.7: SGL-*d* algorithm for maximizing a monotone integer lattice function subject to cardinality constraint.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ (with $n := |V|$), vector of
         quantities $\boldsymbol{b} \in \mathcal{Z}_+^{\mathcal{V}}$, and cardinality constraint $r \in \mathbb{Z}_+$.
Output : Vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ that approximately solves $\max_{\boldsymbol{x}} f(\boldsymbol{x})$ s.t. $||x||_1 \leq r$.
$\boldsymbol{x}^0 \leftarrow \mathbf{0}$;
$d \leftarrow \max_{e \in \mathcal{V}} f(\mathbf{1}_e)$;
$\Theta^0 \leftarrow d$;
$\Theta_{stop} \leftarrow \frac{\varepsilon}{r} \cdot d$; $t \leftarrow 1$;
while $t < r$ do
    $\mathcal{V}^t \leftarrow$ shuffle $\{e \in \mathcal{V} \mid x_e < b_e\}$;
    $\mathcal{Q} \leftarrow$ split $\mathcal{V}^t$ into batches of size at most $s$;
    for $Q \in \mathcal{Q}$ do
        $\Lambda^t \leftarrow \emptyset$;
        for $e \in Q$ do
            $k_{max}^t \leftarrow \min\{b_e - x_e^{t-1}, r - ||\boldsymbol{x}^{t-1}||_1\}$;
            Find the maximum $k^t \in \mathbb{Z}_+$ with $1 \leq k^t \leq k_{max}^t$ such that
              $f(k^t \cdot \mathbf{1}_e \mid \boldsymbol{x}) \geq k^t \cdot \Theta^{t-1}$ using binary search;
            if $\exists\, k^t$ then
                $\Lambda^t \leftarrow \Lambda \cup \{(k^t, e)\}$; // $f$ monotone $\Rightarrow f(\boldsymbol{x}^{t-1} + k^t \cdot \mathbf{1}_e) \geq f(\boldsymbol{x}^{t-1})$
        if $\Lambda^t \neq \emptyset$ then
            $(k^{t*}, e^*) \leftarrow \operatorname{argmax}_{(k^t,e) \in \Lambda^t} f(k^t \cdot \mathbf{1}_{e^*} \mid \boldsymbol{x}^{t-1}))$;
            $\boldsymbol{x}^t \leftarrow \boldsymbol{x}^{t-1} + k^t \cdot \mathbf{1}_{e^*}$;
        else
            $\Theta \leftarrow \max\{\Theta \cdot 0.5, \Theta_{stop}\}$;
        $t \leftarrow t + 1$;
        if $||\boldsymbol{x}^{t-1}||_1 = r$ then
            break
return $\boldsymbol{x}^{t-1}$;
```

We also define

$$\bar{t} := \frac{\log\left[1 - \exp\{-\frac{\log 2}{n}\}\right]}{\log\left(1 - \frac{s}{n}\right)}, \tag{4.7}$$

and we prove the following:

Lemma 4.1. *Consider $\boldsymbol{x}^t$ the current solution at iteration t, and let k be quantity corresponding to a selected element $e \in \mathcal{V}$ that should be added to $\boldsymbol{x}^t$ in Algorithm 4.4. Then, with probability $p > \frac{1}{2}$, it holds that*

$$\frac{f(k^t \cdot \mathbf{1}_e \mid \boldsymbol{x}^t)}{k^t} \geq \frac{(1 - \bar{t}\epsilon)}{r} \sum_{v \in A_t \setminus \{e\}} f(\mathbf{1}_v \mid \boldsymbol{x}^t). \tag{4.8}$$

Proof. We can see that Eq. (4.8) holds for $\Theta = d$ by the same arguments as (Soma & Yoshida, 2018,

Algorithm 4.8: SGL-1 algorithm for maximizing a monotone integer lattice function subject to cardinality constraint.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ (with $n := |V|$), vector of
         quantities $\boldsymbol{b} \in \mathcal{Z}_+^{\mathcal{V}}$, and cardinality constraint $r \in \mathbb{Z}_+$.
Output : Vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ that approximately solves $\max_{\boldsymbol{x}} f(\boldsymbol{x})$ s.t. $||x||_1 \leq r$.
1  $\boldsymbol{x} \leftarrow \boldsymbol{0}$;
2  $d \leftarrow \max_{e \in \mathcal{V}} f(\boldsymbol{1}_e)$;
3  $t \leftarrow 1$;
4  while $t < r$ do
5      $\mathcal{V}^t \leftarrow$ shuffle $\{e \in \mathcal{V} \mid x_e < b_e\}$;
6      $\mathcal{Q}^t \leftarrow$ split $\mathcal{V}^t$ into batches of size at most $s$;
7      for $Q \in \mathcal{Q}^t$ do
8          $e \leftarrow \mathrm{argmax}_{e \in Q}\{f(\boldsymbol{1}_e \mid \boldsymbol{x})\}$;
9          $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{1}_e$;
10         if $||\boldsymbol{x}||_1 = r$ then
11             break;
12     if $||\boldsymbol{x}||_1 = r$ then
13         break;
14     $t \leftarrow t + 1$;
15 return $\boldsymbol{x}$;
```

Lemma 3), which require DR-submodularity. Let us suppose that it also holds for iteration $t > 0$. Indeed, given $v \in A_t \setminus \{e\}$, if v was picked from the previous sample Q_{t-1} in quantity k', we would have that $f(\boldsymbol{1}_v|\boldsymbol{x}^t) \leq \frac{\Theta}{1-\varepsilon}$. This is by contradiction, since if it was otherwise,

$$f((k'+1)\boldsymbol{1}_v \mid \boldsymbol{x}^{t-1}) \geq f(\boldsymbol{1}_v \mid \boldsymbol{x}^t) + f(k'\boldsymbol{1}_v \mid \boldsymbol{x}^{t-1}) > \frac{\Theta}{1-\varepsilon} + \frac{k'\Theta}{1-\varepsilon}$$

would hold, contradicting k' being the largest feasible integer found at the previous iteration $t-1$. Thus, for such a v, it holds that $\frac{f(k\boldsymbol{1}_e \mid \boldsymbol{x}^t)}{k} \geq (1-\varepsilon) f(\boldsymbol{1}_v \mid \boldsymbol{x}^t)$.
On the other hand, if $v \notin Q_{t-1}$, then

$$\frac{f(k\boldsymbol{1}_e \mid \boldsymbol{x}^t)}{k} \geq (1-\varepsilon)^{\hat{t(v,t)}} f(\boldsymbol{1}_v \mid \boldsymbol{x}^t) \geq (1 - \hat{t}(v,t)\varepsilon) f(\boldsymbol{1}_v \mid \boldsymbol{x}^t),$$

where $\hat{t}(v,t)$ denotes the number of iterations which occurred before t since the last time the element v was selected. With a slight abuse of notation, we fix v and t and define $\hat{t} := \hat{t}(v,t)$. Clearly, $\hat{t}$ is a geometric random variable with *success probability* $\hat{p} := \frac{s}{n}$ and with *cumulative distribution function* $1-(1-\hat{p})^{\hat{t}}$. Thus, for every element $v \in \mathcal{V}$ at most $\hat{t}$ iterations old, this would have a probability of $\left(1-(1-\hat{p})^{\hat{t}}\right)^n$. Now,

$$\left(1-(1-\hat{p})^{\hat{t}}\right)^n > \frac{1}{2} \quad \iff \quad \hat{t} < \bar{t} = \frac{\log\left[1-\exp\{-\frac{\log 2}{n}\}\right]}{\log\left(1-\frac{s}{n}\right)}$$

Algorithm 4.9: SGL-II algorithm for maximizing a monotone integer lattice function subject to cardinality constraint.

```
Input  : Monotone integer lattice function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}$ (with $n := |V|$), vector of
         quantities $\boldsymbol{b} \in \mathcal{Z}_+^{\mathcal{V}}$, and cardinality constraint $r \in \mathbb{Z}_+$.
Output : Vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ that approximately solves $\max_{\boldsymbol{x}} f(\boldsymbol{x})$ s.t. $\|x\|_1 \le r$.
1  $\boldsymbol{x} \leftarrow \boldsymbol{0}$;
2  $d \leftarrow \max_{e \in \mathcal{V}} f(\boldsymbol{1}_e)$;
3  $t \leftarrow 1$;
4  while $t < r$ do
5      $\mathcal{V}^t \leftarrow$ shuffle $\{e \in \mathcal{V} \mid x_e < b_e\}$;
6      $\mathcal{Q}^t \leftarrow$ split $\mathcal{V}^t$ into batches of size at most $s$;
7      for $Q \in \mathcal{Q}^t$ do
8          $\{(e, k^e_{max})\}_{e \in Q} \leftarrow \{(e, k^e_{max}) \mid e \in Q, k^e_{max} := \min\{b_e - x_e, r - \|\boldsymbol{x}^{t-1}\|_1\}\}$;
9          $(e, k^e_{max}) \leftarrow \operatorname{argmax}_{(e,k^e_{max}) \in \{(e,k^e_{max})\}_{e \in Q}} f(k^e_{max} \cdot \boldsymbol{1}_e \mid \boldsymbol{x})$;
10         $\boldsymbol{x} \leftarrow \boldsymbol{x} + k^e_{max} \cdot \boldsymbol{1}_e$;
11         if $\|\boldsymbol{x}\|_1 = r$ then
12             break;
13     if $\|\boldsymbol{x}\|_1 = r$ then
14         break;
15     $t \leftarrow t + 1$;
16 return $\boldsymbol{x}$;
```

Hence, with probability $p = \frac{1}{2}$ and for all iterations t and elements $v \in \mathcal{V}$, it holds that

$$\frac{f(k^t \boldsymbol{1}_e \mid \boldsymbol{x}^t)}{k^t} \geq (1-\varepsilon)^{\bar{t}} f(\boldsymbol{1}_v \mid \boldsymbol{x}^t) \geq (1 - \bar{t}\varepsilon) f(\boldsymbol{1}_v \mid \boldsymbol{x}^t).$$

If we average over the elements, we obtain that

$$\frac{f(k^t \boldsymbol{1}_e \mid \boldsymbol{x}^t)}{k^t} \geq \frac{(1 - \bar{t}\varepsilon)}{r} \sum_{v \in A_t \setminus \{e\}} f(\boldsymbol{1}_v \mid \boldsymbol{x}^t).$$

□

Definition 4.1 allows us to prove the following approximation bound.

Theorem 4.2. *Algorithm 4.4 achieves an approximation ratio of* $(1 - \frac{1}{e} - \bar{t}\varepsilon)$ *with probability* $p > \frac{1}{2}$.

Proof. By Definition 4.1, with probability $p > \frac{1}{2}$ we have that

$$\frac{f(k^t \boldsymbol{1}_e \mid \boldsymbol{x}^t)}{k^t} \geq \frac{(1 - \bar{t}\varepsilon)}{r} f(\boldsymbol{x}^* - \boldsymbol{x}^t \mid \boldsymbol{x}^t),$$

where $\boldsymbol{x}^t$ is the solution of Algorithm 4.4 at iteration t, and $\boldsymbol{x}^*$ is the optimal solution. Given k_{sum} the sum of all k selected for a fixed element $e \in \mathcal{V}$ at iteration t, it follows that

$$f(\boldsymbol{x}^{t+1}) - f(\boldsymbol{x}^t) \geq \frac{(1-\bar{t}\varepsilon)k_{sum}}{r} f(\boldsymbol{x}^*) - f(\boldsymbol{x}^t).$$

Let $\boldsymbol{x}^{t_f}$ be the result of Algorithm 4.4. By induction,

$$\begin{aligned} f(\boldsymbol{x}^{t_f}) &\geq \left(1 - \prod_t \left(1 - \tfrac{(1-\bar{t}\varepsilon)k_{sum}}{r}\right)\right) f(\boldsymbol{x}^*) \\ &\geq \left(1 - \prod_t exp(-\tfrac{(1-\bar{t}\varepsilon)k_{sum}}{r})\right) f(\boldsymbol{x}^*) = \left(1 - exp(-\tfrac{(1-\bar{t}\varepsilon)\sum_t k_{sum}}{r})\right) f(\boldsymbol{x}^*) \\ &\geq (1 - \tfrac{1}{e} - \bar{t}\varepsilon) f(\boldsymbol{x}^*). \end{aligned}$$

□

Notice that the probability p in Lemma 4.1 can be adjusted by probability amplification (Motwani, 1995, Section 6.8). In effect, Algorithm SGL-A is likely to achieve an arbitrarily close performance in terms of the approximation ratio as the state of the art Soma & Yoshida (2018), but with a lower overall computational cost.

4.4.2 Bounding the Number of Oracle Queries

Lemma 4.3. *The largest possible value of k^t_{max} is* $\min\{||\boldsymbol{b}||_\infty, r\}$.

Proof. In Algorithm 4.4, k^t_{max} is computed as $\min\{b_e - x_e^{t-1}, r - ||\boldsymbol{x}^{t-1}||_1\}$ for a given $e \in Q$. We recall that $\boldsymbol{b}$ and r are two given constants, and that $||\boldsymbol{x}||_1$ is non-decreasing over time (starting from $\boldsymbol{x} = \boldsymbol{0}$). Thus, in the worst case, $k^t_{max} = \min\{b_e - 0, r - 0\} = \min\{||\boldsymbol{b}||_\infty, r\}$, which can only happen before the first element is selected to be part of the solution. □

1

Proposition 4.4. *The binary search procedure requires $\lceil \log_2(k^t_{max} + 1) \rceil$ oracle queries for a given k^t_{max}. The number of oracle queries performed by one iteration of Algorithm 4.4 is* $\mathcal{O}(\log(\min\{||\boldsymbol{b}||_\infty, r\}))$.

Proof. At each iteration, we consider at most $s := \lfloor \frac{n}{r} \cdot \log \frac{1}{\varepsilon} \rfloor$ distinct elements. From Definition 4.3, we have at most $\mathcal{O}(\log(\min\{||\boldsymbol{b}||_\infty, r\}))$ oracle queries per iteration. □

The number of iterations is a function of a number of parameters, including the function f and its $n := |V|$ and the resulting $\Theta^0 = d$, as well as $\boldsymbol{b}$, r, ε, and the associated $\Theta_{stop} = \frac{\varepsilon}{r} \cdot d$. We present details of the runtime of several variants of the algorithm in an extended version on-line. Let us illustrate the behaviour computationally.

4.5 Experiments

We have validated our proposed algorithm on some synthetic problem instances of increasing size n, cardinality constraint r, and for various uniformly random configurations of the vector of available quantities $\boldsymbol{b}$. We measure the performance of the algorithm both in terms of the number of oracle queries and in terms of the total running time. We compare our SGL algorithms to the following baselines:

- SOMA-DR-I by Soma & Yoshida (2018);
- LAI-DR by Lai et al. (2019);
- SSG, i.e., the Simulated STOCHASTIC GREEDY Mirzasoleiman et al. (2015) algorithm due to Mirzasoleiman *et al.* Mirzasoleiman et al. (2015) lifted from the set to the integer lattice domain.

We did not include SOMA-II (Soma & Yoshida, 2018) in the comparison, as its running time is prohibitely expensive in practice.

Synthetic Monotone Function We define a simple monotone DR-submodular function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}_+$ where $f(\boldsymbol{x}) := \boldsymbol{w}^\top \cdot \boldsymbol{x}$, wherein $w \in \mathbb{Z}_+^{\mathcal{V}}$ is a vector sorted in ascending order whose components w_e are such that $1 \le w_e \le 100$, for every $e \in \mathcal{V}$. One can observe that f is normalized. We consider the following parameter settings:

- $n \in \{100, 200, 500, 750\}$;
- $r \in \{0.25 \cdot n, 0.5 \cdot n, 1 \cdot n, 2 \cdot n\}$;
- Six equidistant $b \in \mathbb{Z}_+ \cap \left[\lfloor \frac{r}{20} \rfloor, \lfloor \frac{r}{2} \rfloor\right]$

$\boldsymbol{b}$ is uniformly sampled in range $\{b, \ldots, b \cdot 4\}$. We discard every unfeasible combination of $(n, \boldsymbol{b}, r)$ such that $r > \|\boldsymbol{b}\|_\infty$ and $r \ge n \cdot \|\boldsymbol{b}\|_\infty$.

Optimal Budget Allocation The *optimal budget allocation* is a particular case of the influence maximization problem. We model it as a bipartite weighted graph $(\mathcal{V}, \mathcal{T}; W)$, where $\mathcal{V}$ is the ground set of the function and represents a collection of advertising channels, whereas T is a collection of customers. Each non-negative weight $p_{st} \in W$ models the probability of channel $s \in \mathcal{V}$ to influence the customer $t \in \mathcal{T}$. The overall objective is to distribute the budget among the advertising channels such that the expected influence on the potential customers is maximized Soma et al. (2014); Hatano et al. (2015). Typically, the budget represents the time for a television campaign, or the space occupied by an inline advertisement. The total influence on customer $t \in \mathcal{T}$ from every channel $s \in \mathcal{V}$ can be modelled as a monotone DR-submodular function $I_t(x)$:

$$I_t(\boldsymbol{x}) := 1 - \prod_{(s,t) \in W} (1 - p_{st})^{x_s}, \tag{4.9}$$

where $\boldsymbol{x} \in \mathbb{Z}_{\ge 0}^n$ is the budget assignment among the advertising channels. The function to maximize is then

$$f(\boldsymbol{x}) := \sum_{t \in T} I_t(\boldsymbol{x}), \tag{4.10}$$

which can be shown to be monotone DR-submodular (as it is a sum of non-negative monotone DR-submodular functions).

We evaluated the optimal budget allocation problem on WIKILENS-RATINGS, a commonly used dataset from the Konect network collection Kunegis (2013). This is a bipartite rating network of

r	75				150				300			
n	150		300		150		300		150		300	
b_{low}	10	10	10	10	10	10	10	10	10	10	10	10
b_{high}	50	90	50	90	50	90	50	90	50	90	50	90

Table 4.1: Summary of the parameter settings for the optimal budget allocation problem. $\boldsymbol{b}$ is uniformly sampled in range $\{b_{low}, \dots, b_{high}\}$.

WikiLens, a wiki-like website, i.e., a graph $\mathcal{G} = (\mathcal{V}, \mathcal{T}, W)$ provided in the form of a list of weighted edges (s, t), with $s \in \mathcal{V}$ representing the source and with $t \in \mathcal{T}$ representing the advertising channels. Every weight in W represents a discrete rating in $\{0, 0.5, \dots, 4.5, 5\}$. We transform the graph $\mathcal{G}$ in the following way:

- We increase the minimum weights to 0.5;
- We normalize each weight by 5 (the maximum rating) to obtain probabilities $p_{st} \in [0, 1]$;
- For a given $n \in \mathbb{Z}_+$, we keep only the out-degree-wise largest n advertising channels, pruning the customers without edges in common to the selected channels.

We refer to Table 4.1 for a summary of the parameter settings used for the optimal budget allocation problem.

Experiments Setting We remark that every algorithm under consideration uses randomization, except SOMA-DR-I. To reduce the bias of our benchmark, for each parameter setting, we repeated every experiment with a randomized algorithm 5 times[4]. Moreover, we seeded our random generator to ensure that every algorithm utilizes exactly the same problem instances, even though those are generated randomly "on the fly".

We fixed the timeout for the experiments at 72 hours in total for both the synthetic monotone function and the optimal budget allocation instances[5]. While every algorithm successfully managed to complete the pipeline of experiments for the synthetic monotone functions, some algorithms timed out before completing the optimal budget allocation pipeline. In particular, for the optimal budget allocation problem:

1. SOMA-DR-I (currently considered among the state of the art for our target problem) did not solve any instance with $n = 300$;
2. SGL-a did not solve any instance with $n = 300$, $b_{low} = 10$, $b_{high} = 90$;
3. SGL-b and SGL-c did not solve the instances with $n = 300$, $b_{low} = 10$, $b_{high} = 90$, and $r \geq 150$;

We conducted our experiments on a Linux server cluster gratefully provided by the "Research Center for Informatics" (part of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765)

[4]Initially we tried 10 repetitions per experiment, but we had to reduce this number due to time constraints
[5]These two types of experiment instances were run at two separate times

equipped with two Intel Xeon CPUs and 384 GBs of RAM. The code for the experiments is written in PYTHON 3.8. For all algorithms that require an error tresholს $\varepsilon > 0$ in input (our included), we fix $\varepsilon := \frac{1}{4n}$, where n is the size of the ground set.

4.5.1 Numerical Results

For both the synthetic monotone function instances and the optimal budget allocation experiments, we report tables and plots with different levels of granularity. Table 4.2 shows the average and median number of oracle queries for the synthetic monotone function instances for $n \in \{100, 200, 500, 750\}$, aggregating over the several configurations of the parameters r and $\boldsymbol{b}$. Similarly, Table 4.3 shows the approximate maximum value returned by the considered algorithms on the same instances. Table 4.4 and Table 4.5 provide a more accurate view of how the oracle queries vary depending on the cardinality bound r. Similarly, Table 4.6 and Table 4.7 show the average and a median result on the synthetic monotone function instances varying both n and r. Considering the optimal budget allocation experiments, instead, Table 4.8 and Table 4.9 show the average and median number of oracle queries and result value obtained by the tested algorithms, respectively, for $n \in \{150, 300\}$. We provide an additional level of granularity in this case as well, showing how the number of oracle queries and the approximated results change depending on both n and r in Table 4.10 and Table 4.11, respectively.

Observations on the Synthetic Monotone Function Instances The first thing we notice is a numerical confirmation of the fact that SSG, the STOCHASTIC GREEDY algorithm "simulated" in a set domain, is indeed infeasible as n grows, and is the worst algorithm in terms of the average number of oracle queries in the case of the synthetic instances in our benchmark. We also point out that, even though LAI-DR and SOMA-DR-I are considered the current state-of-of-the-the-art algorithm for maximizing submodular functions defined in the integer lattice subject to cardinality constraints, they are very expensive to compute, requiring 483 thousands oracle queries and 146 thousands oracle queries on average, respectively, when $n = 750$ in the case of the synthetic monotone function instances; these high numbers are only slightly smaller than the number of oracle queries requested by SSG, i.e., 1.25 million. Our algorithms fare particularly well on the synthetic instances. In particular, SGL-II is the absolute fastest, with SGL-D coming second, requiring less than 2 thousands oracle queries on average even when $n = 750$. The other SGL algorithms appear to be much faster than both LAI-DR and SOMA-DR-I for all values of n, requiring less than 8 thousands oracle queries even in the hardest case ($n = 750$).

We also observe that, in the synthetic instances, the average number of oracle queries increases for each algorithm as n grows, as expected. This is not always true for r. For the SGL algorithms, a higher value of r often implies a smaller number of oracle queries, whereas for LAI-DR the number of oracle queries is strictly growing, since it is exactly $(n + 1) \cdot r$. SOMA-DR-I, instead, does not exhibit any clear pattern in practice.

From the point of view of the actual returned values for the synthetic instances, almost every algorithm yields comparable results. SGL-I, SGL-II, and SGL-D perform noticeably worst than the rest of the algorithms when $r \geq 200$, but the real surprise is LAI-DR, which seems to consistently return solutions that, on average, are about 50% of the values returned by SOMA-DR-I, SSG, SGL-A, SGL-B, SGL-C. Still, our proposed algorithms SGL-I, SGL-II, and SGL-D return significantly higher solutions than Lai-DR. Moreover, SGL-II yields, on average, values that are at most 73.5% smaller than Soma-DR-I's results, and SGL-D yields, on average, values that are at most 80% smaller than Soma-DR-I. A natural observation is that, for every algorithm and ground set size n, a higher cardinality bound r translates to a better (higher) solution.

Observations on the Optimal Budget Allocation Instances The optimal budget allocation problem is harder to maximize than a simple decomposible submodular function made of a sum of weight functions, due to additional nuances like the connectivity degree in the bipartite graph $\mathcal{G}$, and complex form of $I_t(\boldsymbol{x})$. In fact, for this problem the runtime Soma-DR-I explodes, reaching 3.2 million oracle queries on average for $n = 150$, without completing any experiment for $n = 300$. This is particularly impressive if we consider that Soma-DR-I is the only algorithm for which we execute each algorithm only once, as it is deterministic. For comparison, with $n = 150$, SGL-I needs less than 900 oracle queries to yield a solution of 4869 on average, which is even bigger than Soma-DR-I's solution. SGL-D reaches a comparable value of 4306.6 on average with less than 1120 oracle queries. The other SGL-x algorithms suffer from a similar fate than Soma-DR-I, with their number of oracle queries exceeding both Lai-DR-I and SSG (even though the SGL-x does not reach a million oracle query even when $n = 300$, which makes them better than Soma-DR-I in practice). SGL-II is again the absolute fastest algorithm, but also the lowest performant in terms of average result values. If we exclude SGL-II, the results yielded by our SGL algorithms are always more accurate than Lai-DR's, except for one case ($n = 150$ and $r = 300$), where Lai-DR returns 4667.2 and SGL-D returns 4340.1, on average.

Another evidence of the fact that the budget allocation problem is computationally more expensive to maximize emerges from the comparison in of the average number of seconds needed to complete an experiment using a synthetic monotone instance (Table 4.12) or with an optimal budget allocation instance Table 4.13.

4.6 Concluding Remarks

In this chapter, we considered the problem of maximizing a monotone integer lattice and DR-submodular functions subject to cardinality constraints. We also proposed one stochastic algorithm for this problem, inspired by the random sampling technique that, until now, has only been applied to set submodular functions Mirzasoleiman et al. (2015). Experimentally, we have shown that the best of our algorithms (SGL-d) requires considerably fewer value-oracle queries than state-of-the-art deterministic algorithms for the same problem, as well as the naive baseline lifting the integer lattice to a set domain. We have publicly released the code for our algorithms and experiments online[6]. As far as we know, this is the first open source package for submodular function maximization on the integer lattice subject to cardinality constraints.

[6] *https://github.com/jkomyno/lattice-submodular-maximization*

	$n = 100$	$n = 200$	$n = 500$	$n = 750$
Soma-DR-I	7459.9	22261.8	67346.9	145633.8
	6320.5	11945.5	3610.5	3315.5
Lai-DR	8657.1	34457.1	214714.3	482678.4
	5050.0	20100.0	125250.0	281625.0
SSG	16614.3	73978.6	531446.4	1256565.0
	8450.0	41300.0	302562.5	707620.5
SGL-I	481.9	1220.3	3487.8	5947.5
	500.0	1225.0	3661.0	5986.0
SGL-II	108.3	215.0	567.5	890.4
	33.0	36.0	45.0	48.0
SGL-a	1232.8	2540.7	5239.5	7967.4
	864.5	1446.5	3427.5	4139.0
SGL-b	1078.6	2321.7	4574.8	7203.8
	759.5	1282.0	2686.0	3219.0
SGL-c	1137.9	2380.2	4719.9	7509.7
	831.5	1402.5	2900.5	3874.0
SGL-d	296.7	559.3	1242.0	1826.7
	261.0	431.0	823.5	1163.0

Table 4.2: Average and median number of oracle queries grouped by n for the synthetic monotone function instances. Lower values are preferable.

	$n = 100$	$n = 200$	$n = 500$	$n = 750$
Soma-DR-I	8208.4	16615.2	41867.4	62795.7
	4865.0	9880.5	24750.0	37125.0
Lai-DR	4235.0	8528.0	21283.8	32005.9
	2661.5	5159.5	13153.0	19253.0
SSG	8199.4	16625.8	41854.8	62793.5
	4848.0	9862.5	24750.0	37125.0
SGL-I	6474.4	13826.6	34843.4	54099.7
	4418.0	9282.0	23422.0	35141.5
SGL-II	6033.6	13178.7	31981.4	51092.3
	4524.0	9011.0	22664.5	32138.0
SGL-a	8105.2	16478.4	41584.8	62577.6
	4825.0	9880.5	24688.5	37125.0
SGL-b	8162.6	16538.9	41669.4	62654.4
	4857.5	9866.0	24750.0	37125.0
SGL-c	8166.4	16539.8	41723.0	62625.1
	4865.0	9880.5	24750.0	37125.0
SGL-d	6621.0	13485.0	34613.6	52874.3
	4542.5	9374.5	22467.0	33928.5

Table 4.3: Average and median result values grouped by n for the synthetic monotone function instances. Higher values are preferable.

	$n = 100$				$n = 200$			
	$r = 25$	$r = 50$	$r = 100$	$r = 200$	$r = 50$	$r = 100$	$r = 200$	$r = 400$
Soma-DR-I	5806.2	9412.5	6489.7	8031.7	21112.5	32888.0	11793.0	20094.7
	5484.0	6026.0	7457.0	6133.0	19314.5	19361.5	12358.0	8333.0
Lai-DR	2525.0	5050.0	10100.0	20200.0	10050.0	20100.0	40200.0	80400.0
	2525.0	5050.0	10100.0	20200.0	10050.0	20100.0	40200.0	80400.0
SSG	3587.5	7487.5	20766.7	42000.0	16175.0	35150.0	92666.7	184133.3
	3087.5	6000.0	20700.0	42600.0	13500.0	31350.0	92400.0	186000.0
SGL-I	502.2	509.5	500.0	400.0	1239.9	1249.8	1180.0	1195.0
	500.0	507.0	500.0	400.0	1250.0	1252.0	1180.0	1195.0
SGL-II	195.8	162.6	18.7	8.8	382.1	343.9	23.2	12.2
	107.5	55.0	15.0	6.0	130.0	58.5	18.0	9.0
SGL-a	1714.8	1551.3	840.7	557.7	3515.3	3358.1	1341.3	1350.7
	1631.0	982.5	803.0	595.0	2951.5	2254.0	1355.0	1226.0
SGL-b	1464.8	1396.5	700.2	518.5	3221.2	3110.5	1131.3	1261.2
	1333.5	908.0	707.0	554.0	2855.0	2033.0	1058.0	1064.0
SGL-c	1518.4	1488.8	756.5	544.1	3333.9	3219.3	1150.0	1220.0
	1337.5	980.0	774.0	489.0	2849.0	2194.5	1167.0	1201.0
SGL-d	415.8	337.6	213.7	166.1	801.8	651.5	367.6	304.5
	389.5	293.0	197.0	161.0	665.5	496.0	342.0	295.0

Table 4.4: Average and median number of oracle queries grouped by $n \in \{100, 200\}$ and r for the synthetic monotone function instances. Lower values are preferable.

	$n = 500$				$n = 750$			
	$r = 125$	$r = 250$	$r = 500$	$r = 1000$	$r = 187$	$r = 375$	$r = 750$	$r = 1500$
Soma-DR-I	73663.0	121416.5	24944.0	29235.7	155623.8	303320.8	2980.7	64717.7
	24419.0	29398.5	2346.0	4623.0	2667.5	95319.0	3079.0	3324.0
Lai-DR	62625.0	125250.0	250500.0	501000.0	140437.0	281625.0	563250.0	1126500.0
	62625.0	125250.0	250500.0	501000.0	140437.0	281625.0	563250.0	1126500.0
SSG	125375.0	243812.5	660166.7	1327666.7	288915.0	582750.0	1531250.0	3170500.0
	111562.5	220500.0	667000.0	1313000.0	257592.5	525187.5	1550250.0	3148500.0
SGL-I	3670.5	3683.5	3476.0	2995.0	5864.8	5972.5	5986.0	5986.0
	3680.0	3680.0	3476.0	2995.0	5858.0	5986.0	5986.0	5986.0
SGL-II	990.0	967.0	26.6	12.6	1554.5	1527.0	29.9	16.5
	135.0	67.5	21.0	9.0	144.0	72.0	24.0	12.0
SGL-a	7137.1	7182.8	2684.7	2672.9	10642.3	11887.5	3500.5	3640.8
	4993.5	4105.0	2686.0	2438.0	4625.0	6303.5	3596.0	3667.0
SGL-b	5983.1	6590.4	2412.7	2171.5	9700.9	10914.8	2895.3	3234.7
	3286.5	3681.5	2432.0	2192.0	3048.5	5264.0	3082.0	3305.0
SGL-c	6263.0	6626.9	2523.3	2316.3	10005.2	11316.6	3369.7	3246.3
	3182.5	3622.5	2655.0	2264.0	3715.5	5986.0	3715.0	3539.0
SGL-d	1797.6	1523.0	736.9	631.7	2652.1	2264.6	1034.0	935.3
	1238.0	946.0	719.0	622.0	1635.5	1262.5	976.0	918.0

Table 4.5: Average and median number of oracle queries grouped by $n \in \{500, 750\}$ and r for the synthetic monotone function instances. Lower values are preferable.

	$n = 100$				$n = 200$			
	$r = 25$	$r = 50$	$r = 100$	$r = 200$	$r = 50$	$r = 100$	$r = 200$	$r = 400$
Soma-DR-I	2341.2	4526.2	9789.0	19360.0	4775.2	9146.8	19690.0	39285.0
	2356.0	4788.0	9788.0	19600.0	4866.0	9787.0	19704.0	39586.0
Lai-DR	1248.7	2480.8	4813.8	9976.9	2459.8	4835.6	9878.1	20192.1
	1242.5	2487.5	4717.0	9911.0	2400.0	4809.5	9792.0	20393.0
SSG	2333.0	4512.4	9774.3	19362.3	4771.2	9124.1	19707.9	39351.9
	2346.0	4775.0	9822.0	19593.0	4873.5	9782.5	19669.0	39579.0
SGL-I	2268.4	4158.5	8239.4	13405.3	4677.1	8636.8	16905.1	29867.1
	2262.5	4263.0	8254.0	13600.0	4761.5	9125.5	17001.0	29914.0
SGL-II	2210.2	4196.1	7584.4	12030.7	4427.4	8210.4	15696.5	28953.7
	2169.0	4327.0	7600.0	11999.0	4356.5	8364.0	15809.0	29612.0
SGL-a	2340.0	4513.8	9713.7	18972.1	4772.6	9135.9	19574.0	38780.7
	2354.0	4769.0	9731.0	19212.0	4854.5	9751.5	19603.0	38992.0
SGL-b	2341.0	4524.6	9744.9	19193.1	4775.2	9142.4	19626.1	38998.5
	2356.0	4785.0	9762.0	19352.0	4866.0	9770.0	19643.0	39084.0
SGL-c	2341.2	4523.0	9736.2	19221.5	4775.1	9143.0	19648.2	38979.9
	2356.0	4784.0	9760.0	19436.0	4866.0	9770.0	19660.0	39084.0
SGL-d	2255.2	4170.9	8133.8	14196.2	4545.3	8313.5	15867.7	29917.5
	2272.0	4322.5	8160.0	14162.0	4411.5	8391.0	16350.0	30032.0

Table 4.6: Average and median result values grouped by $n \in \{100, 200\}$ and r for the synthetic monotone function instances. Higher values are preferable.

	$n = 500$				$n = 750$			
	$r = 125$	$r = 250$	$r = 500$	$r = 1000$	$r = 187$	$r = 375$	$r = 750$	$r = 1500$
Soma-DR-I	11974.0	23374.0	49413.0	98837.3	17940.5	35030.5	74250.0	148168.7
	12361.5	24710.5	49500.0	99000.0	18513.0	36988.0	74250.0	148500.0
Lai-DR	6122.4	12471.6	24866.3	49666.0	9277.8	18987.6	36892.3	74781.5
	6156.5	12399.5	24912.0	49712.0	9332.0	19113.0	37017.0	73790.0
SSG	11959.2	23351.0	49385.9	98856.1	17918.0	34992.8	74246.7	148241.7
	12356.0	24704.5	49500.0	98990.0	18510.5	36995.0	74246.0	148500.0
SGL-I	11691.7	22322.3	43315.4	73935.1	17550.0	33516.2	66002.8	118373.9
	11990.5	23179.5	43337.0	73975.0	17995.5	34989.0	65905.0	118163.0
SGL-II	11123.4	21318.5	40315.4	65675.5	16912.0	29993.9	61186.1	114703.5
	11148.0	21219.0	39845.0	65462.0	16740.0	29397.0	60832.0	116104.0
SGL-a	11966.0	23329.3	49213.6	97788.3	17940.0	34984.3	73966.2	147496.9
	12347.5	24637.0	49354.0	97789.0	18513.0	36912.0	74097.0	147692.0
SGL-b	11974.0	23357.9	49317.9	98030.1	17940.5	35014.0	74102.3	147678.9
	12361.5	24672.0	49375.0	98168.0	18513.0	36988.0	74084.0	148304.0
SGL-c	11974.0	23359.5	49309.8	98286.3	17940.5	34995.3	74082.1	147587.5
	12361.5	24669.5	49384.0	98473.0	18513.0	36859.5	74087.0	147452.0
SGL-d	11285.1	21121.0	41023.3	77298.8	16934.7	31761.8	61382.1	120435.9
	11103.0	21066.0	40096.0	73615.0	16879.5	32168.5	61831.0	117130.0

Table 4.7: Average and median result values grouped by $n \in \{500, 750\}$ and r for the synthetic monotone function instances. Higher values are preferable.

	$n = 150$	$n = 300$
Soma-DR-I	3197500.8	-
	3047197.5	-
Lai-DR	26425.0	52675.0
	22650.0	45150.0
SSG	36812.5	84812.5
	36862.5	84750.0
SGL-I	890.0	2066.0
	900.0	2052.0
SGL-II	29.8	63.5
	30.0	59.5
SGL-a	148321.6	657135.5
	140216.5	614453.0
SGL-b	145326.1	738112.1
	140027.5	769323.5
SGL-c	145355.2	738123.8
	140065.0	769208.5
SGL-d	1113.2	2774.5
	1084.5	2438.5

Table 4.8: Average and median number of oracle queries grouped by n for the optimal budget allocation instances. Lower values are preferable.

	$n = 150$	$n = 300$
Soma-DR-I	4866.8	-
	4851.8	-
Lai-DR	3945.6	3616.3
	4120.6	3661.6
SSG	4896.7	4912.6
	4926.4	4943.2
SGL-I	4869.0	4891.5
	4902.3	4922.2
SGL-II	1634.8	1659.6
	1539.4	1621.6
SGL-a	4896.6	4912.3
	4926.3	4943.1
SGL-b	4896.7	4879.3
	4926.3	4861.6
SGL-c	4896.7	4879.3
	4926.4	4861.7
SGL-d	4306.6	4408.3
	4296.9	4477.1

Table 4.9: Average and median result values grouped by n for the optimal budget allocation instances. Higher values are preferable.

	$n = 150$			$n = 300$		
	$r = 75$	$r = 150$	$r = 300$	$r = 75$	$r = 150$	$r = 300$
Soma-DR-I	2684735.5	3280464.0	4140068.0	-	-	-
	2684735.5	3280464.0	4140068.0	-	-	-
Lai-DR	11325.0	22650.0	45300.0	22575.0	45150.0	90300.0
	11325.0	22650.0	45300.0	22575.0	45150.0	90300.0
SSG	36862.5	36825.0	36750.0	84862.5	84825.0	84750.0
	36862.5	36825.0	36750.0	84862.5	84825.0	84750.0
SGL-I	870.0	900.0	900.0	2052.0	2052.0	2094.0
	870.0	900.0	900.0	2052.0	2052.0	2094.0
SGL-II	33.6	30.0	25.8	70.0	63.0	57.4
	36.0	33.0	24.0	70.0	63.0	59.5
SGL-a	215771.9	140259.1	88933.7	965923.2	614391.8	391091.4
	215778.5	140216.5	88871.0	965784.0	614453.0	391225.0
SGL-b	207161.7	140022.6	88793.9	982464.0	598265.4	389255.0
	207172.5	140027.5	88769.5	982487.0	598307.0	389170.0
SGL-c	207174.4	140073.9	88817.2	982572.1	598094.6	389256.6
	207185.5	140065.0	88797.5	982645.5	598088.0	389231.0
SGL-d	1421.9	1125.7	792.1	3980.7	2488.5	1854.3
	1457.0	1127.5	798.0	3976.0	2555.0	1821.5

Table 4.10: Average and median number of oracle queries grouped by n and r for the optimal budget allocation instances. Lower values are preferable.

	$n = 150$			$n = 300$		
	$r = 75$	$r = 150$	$r = 300$	$r = 75$	$r = 150$	$r = 300$
Soma-DR-I	4777.3	4926.4	4986.3	-	-	-
	4777.3	4926.4	4986.3	-	-	-
Lai-DR	3098.4	4071.1	4667.2	2789.0	3597.6	4462.4
	3079.5	4120.6	4676.0	2743.3	3661.6	4520.8
SSG	4777.2	4926.3	4986.6	4780.2	4943.2	5014.3
	4777.3	4926.4	4986.6	4780.2	4943.2	5014.3
SGL-I	4755.0	4901.9	4950.2	4764.8	4922.5	4987.1
	4754.3	4902.3	4950.4	4767.2	4922.2	4987.0
SGL-II	1348.1	1651.7	1904.6	1267.2	1501.8	2209.8
	1335.2	1608.9	1560.7	1438.9	1531.1	2310.6
SGL-a	4777.1	4926.3	4986.4	4780.2	4943.2	5013.6
	4777.2	4926.3	4986.4	4780.2	4943.1	5013.6
SGL-b	4777.2	4926.3	4986.4	4780.2	4943.1	5013.7
	4777.3	4926.3	4986.4	4780.2	4943.1	5013.7
SGL-c	4777.3	4926.3	4986.4	4780.2	4943.1	5013.7
	4777.3	4926.4	4986.4	4780.2	4943.1	5013.7
SGL-d	4177.8	4401.9	4340.1	4414.1	4328.6	4482.2
	4219.1	4442.6	4317.0	4468.6	4360.1	4568.4

Table 4.11: Average and median result values grouped by n and r for the optimal budget allocation instances. Higher values are preferable.

	$n = 100$	$n = 200$	$n = 500$	$n = 750$
Soma-DR-I	116.7	650.4	6990.0	19589.4
Lai-DR	911.0	3678.5	23829.9	56192.9
SSG	30.4	241.3	4122.3	13315.4
SGL-I	0.7	2.2	10.1	22.6
SGL-II	0.2	0.4	1.8	3.7
SGL-a	1.8	4.3	13.0	26.7
SGL-b	1.6	4.0	12.2	26.0
SGL-c	1.6	3.9	11.9	25.6
SGL-d	1.1	3.2	13.5	29.1

Table 4.12: Average number of seconds of running time grouped by n for the synthetic monotone function instances. Lower values are preferable.

	$n = 150$	$n = 300$
Soma-DR-I	136615.1	-
Lai-DR	9828.8	20195.6
SSG	11800.4	28065.6
SGL-I	307.9	740.6
SGL-II	9.9	21.8
SGL-a	50499.8	113524.9
SGL-b	47445.9	161899.0
SGL-c	52365.0	165404.0
SGL-d	372.0	1058.1

Table 4.13: Average number of seconds of running time grouped by n for the optimal budget allocation instances. Lower values are preferable.

5 Beyond Optimization: Probablistic Submodular Models

So far, we have explored fundamental properties of submodular functions, their most relevant generalizations and problem formulations, plus a review of the most important algorithms in submodular optimization and our novel contributions to this field as well. As we mentioned in the introduction to this thesis, we are interested in exploring submodularity beyond the function optimization perspective.

In this chapter, we turn our attention back to set submodular functions, and we lift known results from submodular optimization to probabilistic inference, taking a Bayesian approach. We present some recent results that allow developing efficient algorithms for approximate inference in probabilistic discrete models, which is a problem with emerging interest in machine learning, statistics, statistichal mechanics, and physics. In particular, we will explore the notion of *probabilistic submodular models*, which provide a generalization to binary[1] Markov Random Fields and Determinantal Point Processes (DPPs). Moreover, these tools can be used to model notions such attractiveness and repulsion between problem variables and capture higher order dependencies among them.

Additional Notation We now introduce some additional notation (complementary to the notation introduced in Chapter 2) that will be useful for this chapter. We consider a finite n-dimensional ground set $\mathcal{V} := \{1, 2, \ldots, n\}$ as usual, but we use $f : 2^{\mathcal{V}} \rightarrow \mathbb{R}$ to denote both set submodular and supermodular functions (the distinction between the two classes of functions will be made clear by the context). Moreover, we denote modular functions as $m : 2^{\mathcal{V}} \rightarrow \mathbb{R}$. Since modular functions can be formulated as $f(S) := \sum_{e \in S} m_i$ for some $S \subseteq \mathcal{V}$ and $m_i \in \mathbb{R}$, such functions can be parametrized by vectors $\boldsymbol{m} \in \mathbb{R}^n$. We also use the shorthand notation $\boldsymbol{m}(S) := \sum_{e \in S} m_e$.
Given two variables x and y, we say that y is directly proportional to x if there is $k \neq 0$ such that $y := k \cdot x$; we denote this relation as $y \propto x$. We also use $\exp(x)$ to indicate the exponential function with argument x, i.e., e^x. Finally, we use $\boldsymbol{\theta} \in \mathbb{R}^n$ to denote a parameter vector to be learned, and $\mathcal{Z}$ to indicate the partition function of a probability distribution.

[1]Recall that set submodular functions of the form $f : 2^{\mathcal{V}} \rightarrow \mathbb{R}$ model the decision problem of selecting any item $e \in \mathcal{V}$, which is a *binary* problem.

5.1 Bayesian Inference

We have already given a characterization of set submodular functions, and in the previous chapters we have observed several cases in which the *structure* of such functions is the core feature that allows for efficient approximate optimization algorithms. As has been recently investigated, this submodular structure can also be exploited in performing statistical inference.

We briefly recall Bayes theorem, giving an intuition to its definition. Assuming we have we sets of outcomes H and E, which we refer to as *events*, we denote the probability of each event as $Pr(H)$ and $Pr(E)$, respectively. The probability that each events co-occurs is denoted as $Pr(H, E)$. This is known as the *joint probability* of H and E, and can be formalized as

$$Pr(H, E) = Pr(H \mid E) \cdot Pr(E), \tag{5.1}$$

i.e., the *conditional probability* of H occurring knowing that E has occurred multiplied by the probability of event E occurring results in the joint probability of H and E. However, since $Pr(H, E) = Pr(E, H)$, it is immediate to see that

$$Pr(H, E) = Pr(E \mid H) \cdot Pr(H). \tag{5.2}$$

Then, considering that Eq. (5.1) and Eq. (5.2) are equal it holds that

$$Pr(H \mid E) \cdot Pr(E) = Pr(E \mid H) \cdot Pr(H).$$

This leads us to Bayes' theorem:

$$Pr(H \mid E) = \frac{Pr(E \mid H) \cdot Pr(H)}{Pr(E)}. \tag{5.3}$$

In order to establish the standard nomenclature in Bayesian inference, we rewrite the definition of Bayes' theorem as follows:

$$\overbrace{Pr(\text{hypothesis} \mid \text{evidence})}^{\text{posterior}} = \frac{\overbrace{Pr(\text{evidence} \mid \text{hypothesis})}^{\text{likelihood}} \cdot \overbrace{Pr(\text{hypothesis})}^{\text{prior}}}{\underbrace{Pr(\text{evidence})}_{\text{marginal likelihood}}}. \tag{5.4}$$

In particular,

- the *hypothesis* is a statement one can formulate whose probability of occurring depends by some data, referred to as *evidence*;
- the *prior probability* is an estimate of the probability of the given hypothesis before having observed the evidence;
- the *posterior probability* is the probability of a hypothesis given the observed evidence;
- the *likelihood* is the probability of of observing some evidence knowing that a given hypothesis holds, and it measures the level of compatibility between the evidence and the hypothesis.

In Eq. (5.3), the *marginal likelihood Pr(evidence)* has the role of a *normalizing constant*, i.e., it ensures that the *posterior distribution* is a valid probability density, e.g., it should range between 0 and 1. For this reason, the *evidence* normalizing constant is often dropped, and the Bayes' theorem can the be written as

$$\text{posterior} \propto \text{likelihood} \cdot \text{prior}, \tag{5.5}$$

where the symbol $\propto$ stands for "proportional to".

In statistical inference, the goal is to infer properties of interest from a probability distribution. The Bayesian approach, in particular, compares different hypotheses by their posterior probability. In essence, Bayesian inference reasons in terms of probabilities that are conditional on the observed evidence. In the definition of Bayesian probabilistic models, the θ variable is often used to denote the parameter vector.

5.2 Probabilistic Submodular Models

We consider discrete distributions over n categorical random variables $X_1, X_2, \dots, X_n$, (each of which takes values in $\{0, 1\}$), which we index in the ground set $\mathcal{V} := \{1, 2, \dots, n\}$. We analyze distributions of the form

$$\pi(S; \theta) = \frac{1}{Z(\theta)} \exp(-f(S; \theta)), \tag{5.6}$$

for all $S \subseteq \mathcal{V}$ where $f : 2^{\mathcal{V}} \to \mathbb{R}$ is called an *energy function.*

Remark. *Following the Bayesian approach, the set S that maximizes the probability of $\pi(S; \theta)$ in Eq. (5.6) is the maximizer over f of $S \subseteq \mathcal{V}$.*

The quantity $\mathcal{Z}(\theta)$ is called *partition function*, and it denotes the normalizing constant of the distribution in Eq. (5.6), so it is defined as

$$\mathcal{Z}(\theta) := \sum_{S \subseteq \mathcal{V}} \exp(-f(S; \theta)). \tag{5.7}$$

Similarly to what we have done in Section 5.1, when defining models of the form of Eq. (5.6) we often omit the partition function (as it is unfeasible to compute in general), and write

$$\pi(S; \theta) \propto \exp(-f(S; \theta)),$$

for all $S \subseteq \mathcal{V}$.

Definition 5.1 (Probabilistic Submodular Model)**.** *When $f : 2^{\mathcal{V}} \to \mathbb{R}$ is a submodular function, we refer to distributions over finite sets of the form of Eq. (5.6) as* probabilistic submodar models. *Distributions of this form are also known* log-submodular.

Remark. *The distribution in Eq. (5.6) is invariant when f is shifted by any scalar, as such change would be canceled by the partition function $\mathcal{Z}(\theta)$.*

Definition 5.2 (Log-Modular Distribution)**.** *Consider a modular function expressed as* $\mathbf{m} \in \mathbb{R}^n$*. Whenever* $f(S) := \mathbf{m}(S)$ *is a modular function, the distribution*

$$\pi(S;\theta) \propto \exp\left(-f(S;\theta)\right),$$

is called log-modular*, and it corresponds to a fully factorized distribution over random binary variables* $\theta_1, \dots, \theta_n$*, where there is a bijection for each element* $e \in \mathcal{V}$ *and variable* θ_e *which tells whether* $e \in S$ *or not. More formally, such a distribution can be formulated as*

$$\pi(S) := \frac{1}{\mathcal{Z}} \exp(\mathbf{m}(S)) = \prod_{e \in S} \sigma(m_e) \prod_{e \notin S} \sigma(-m_e),$$

where

$$\sigma(x) := \frac{1}{1+e^{-x}}$$

is the sigmoid *function (Gotovos et al., 2018).*

5.2.1 Applications

Probabilistic submodular models (and generalizations thereof, like *mixed* PSMs in Djolonga et al. (2016)) naturally model several applications, such as 3D stereo reconstruction, image segmentation, and modeling of user preferences to name a few. The most classical examples, however, originated in statistical physics.

Example 5.3 (Ising model)**.** *In its most simplified form, the Ising model (Ising, 1925) is defined by means of an undirected graph* $\mathcal{G} := (\mathcal{V}, \mathcal{E})$*, and a set of "attractive" pairwise potentials*

$$\sigma_{i,j}(S) := 4\left(\llbracket\{i \in S\}\rrbracket - 0.5\right)\left(\llbracket\{j \in S\}\rrbracket - 0.5\right),$$

for all $\{i,j\} \in \mathcal{E}$*. We let* $\llbracket\cdot\rrbracket$ *indicate the Iverson bracket, which returns value* 1 *when the inner condition evaluates to true, and* 0 *otherwise. It is easy to observe that* $\sigma_{i,j} = 1$ *when* S *contains both or neither of* i, j*, and value* $\sigma_{i,j} = -1$ *if* S *contains only one of* i *or* j*. As a consequence of this* diminishing returns *property,* $\sigma_{i,j}$ *is a set submodular function.*
The Ising distribution is usually defined as

$$\pi(S) \propto \exp\left(\sum_{\{i,j\}\in\mathcal{E}} \sigma_{i,j}(S)\right).$$

It is a probabilistic submodular model, since each $\sigma_{i,j}$ *is supermodular, and supermodular functions are closed under addition. Even though Ising models and Potts models (Potts, 1952) - a generalization of Ising models from binary to k-ary variables - were first formulated in the context of statistical physics, they have been proven to be useful in computer vision tasks (Wang et al., 2013).*

Example 5.4 (Determinantal Point Process)**.** *Determinantal point processes (DPPs) are models of repulsion that appear in quantum physics and random matrix theory (Lyons, 2003; Kulesza & Taskar,* *2012). A DPP is defined as a positive semidefinite matrix $L \in \mathbb{R}^{n\times n}$ with a distribution of the form*

$$\pi(S) = \frac{\det(L_S)}{\det(L+I)},$$

where L_S indicates the square submatrix indexed by set S, and I denotes the identity matrix of size $n \times n$. It can be shown that $f(S) := \log\det(L_S)$ is a set submodular function (Lyons, 2003), so DPPs are probabilistic submodular models. DPPs are a particularly rare known case of distributions where the partition function $\mathcal{Z} := \det(L+I)$ is easily computable, which makes DPPs tractable for exact inference. Even though DPPs were first introduced in statistical physics, thet have recently been adopted to incentivate diversity in machine learning applications, especially in image summarization (Kulesza & Taskar, 2012) and video summarization (Gong et al., 2014).

5.3 Hardness of Exact Inference and Alternatives

Obtaining exact inference in models of the form Eq. (6.1), i.e., computing conditional probabilities such that $\pi(A \subseteq S \subseteq B \mid C \subseteq S \subseteq D)$, is #P-Hard in general (Jerrum & Sinclair, 1993), which means that exact inference is computationally infeasible. For instance, even approximating the value of the partition function of a general Ising model is a hard problem, as there is no FPRAS in this case (Jerrum & Sinclair, 1993). As we have mentioned in Definition 5.4, DPPs are one of the very few cases of tractable probabilistic submodular models (Lyons, 2003; Kulesza & Taskar, 2012).

This difficulties led to alternative approaches for approximate inference algorithms, such as Markov chain Monte Carlo sampling (Levin, 2009), which is the main scope of the following sections. In particular, Rayleigh distributions (Borcea et al., 2008) (a generalization of DPPs resulting from the study of negative dependence between random variables (Pemantle, 2000; Liggett, 2002; Wagner, 2008)) have been shown to allow efficient sampling using a simple Metropolis sampler (Anari et al., 2016; Li et al., 2016).

Besides sampling, the primary alternative for performing approximate inference in discrete models have been variational methods. The fundamental idea of these methods is to choose a distribution among a tractable class to approximate the true distribution at hand. Variational methods have been studied extensively for low-order models, particularly for exponential families. Several well-studied algorithms, such as belief propagation and mean-field methods, fall under this category. In this thesis, we do not expand upon variational methods, but the interested reader may refer to Wainwright & Jordan (2008).

5.4 Previous Work in Probabilistic Submodular Models

Djolonga & Krause (2014) were the first to introduce the study of learning and variational inference in PSMs while at ETH Zurich[2]. Their idea is to bound the value of $\mathcal{Z}$ with a submodular

[2]https://las.inf.ethz.ch/research/probabilistic-submodular-models

polyhedra argument. In particular, $f(S)$ is initially bounded using functions of the form

$$\boldsymbol{m}(S) + t,$$

where $\boldsymbol{m} \in \mathbb{R}^n$ models a set modular function and $t \in \mathbb{R}$. We first consider the *log-supermodular* case. If it holds that $S \subseteq \mathcal{V} \Rightarrow m(S) + t \leq f(S)$ for every set $S \subseteq \mathcal{V}$, then the logarithm of the partition function $\mathcal{Z}$ can be bounded by

$$\log \mathcal{Z} := \log \sum_{S \subseteq \mathcal{V}} \exp(-f(S)) \leq \log \sum_{S \subseteq \mathcal{V}} \exp(-\boldsymbol{m}(S) - t) = \sum_{i=1}^{n} \log(1 + \exp(-m_i))$$

Notice that $\boldsymbol{m}$ and t are the only free parameters. One can then solve the optimization problem

$$\min_{(\boldsymbol{m},t) \in \mathcal{L}(f)} \sum_{i=1}^{n} \log(1 + \exp(-m_i)) - t, \tag{5.8}$$

where $\mathcal{L}(f)$ is the *generalized submodular lower polyhedron* (Iyer & Bilmes, 2015), i.e., the set of every lower bound of f:

$$\mathcal{L}(f) := \{(\boldsymbol{x}, t) \in \mathbb{R}^{n+1} \mid \boldsymbol{x}(S) + t \leq f(S) \text{ for all } S \subseteq \mathcal{V}\}. \tag{5.9}$$

Furthermore, Djolonga & Krause (2014) prove that we can obtain the same minimum optimum if we restrict ourself to $t := 0$ and to the *base polyhedron* $\mathcal{B}(f)$ (Eq. (2.12)):

$$\mathcal{B}(f) := \mathcal{L}(f) \cap \{(\boldsymbol{x}, 0) \in \mathbb{R}^{n+1} \mid \boldsymbol{x}(\mathcal{V}) = f(\mathcal{V})\}.$$

In practice, the base polyhedron $\mathcal{B}(f)$ contains all modular lower bounds of f which are *tight* at the two extremes, $\emptyset$ and $\mathcal{V}$. The algorithm of Edmonds (1971) to minimize linear functions over $\mathcal{B}(f)$ (previously discussed in Eq. (2.15)), and the fact that $\log(1 + \exp(-m_i))$ is $\frac{1}{4}$-smooth for any $m_i \in \mathbb{R}$, allow to solve the linear problem Eq. (5.10) using the Frank-Wolfe algorithm for convex optimization (Frank & Wolfe, 1956; Jaggi, 2013).

In the *log-submodular* case, however, the problem in Eq. (5.10) becomes

$$\min_{(\boldsymbol{m},t) \in \mathcal{U}(f)} \sum_{i=1}^{n} \log(1 + \exp(+m_i)) + t, \tag{5.10}$$

by swapping signs and using $\mathcal{U}(f)$, the *generalized submodular upper polyhedron* (Iyer & Bilmes, 2015), i.e., the set of every upper bound of f:

$$\mathcal{U}(f) := \{(\boldsymbol{x}, t) \in \mathbb{R}^{n+1} \mid \boldsymbol{x}(S) + t \geq f(S) \text{ for all } S \subseteq \mathcal{V}\}. \tag{5.11}$$

Eq. (5.11), which is tightly related to submodular maximization, is however NP-Hard (and in particular, determining whether any $(\boldsymbol{x}, t)$ belongs to $\mathcal{U}(f)$ is NP-Hard). As it turns out, an approximated value of the partition function $\mathcal{Z}$ can be found using a Kullback-Leibler divergence argument and the Lovász extension $\hat{f}$ of f (Definition 2.27), yielding a maximization problem (named L-Field) that is concave and efficient to evaluate.

Djolonga et al. (2016) built upon the results of Djolonga & Krause (2014) to develop variational inference techniques for *mixed submodular models*, i.e., a formulation where PSMs are expressed in the form

$$p(S) \propto \exp(f(S) - g(S)),$$

where both f and g are set submodular functions defined over the same ground set $\mathcal{V}$. These *mixed* models are general, as any set function can be rewritten as the difference of a submodular and a supermodular function (even though such a decomposition is hard to find, as noted in Narasimhan & Bilmes (2005)).

Gotovos et al. (2015) investigated the user of MCMC investigated the use of Markov chain Monte Carlo (MCMC) sampling as a tool to perform approximate inference of probabilistic submodular models, establishing mild conditions under which a simple Gibbs sampler obtains a fast ($\mathcal{O}(n \log n)$) mixing time.

Using the Gibbs sampler as a baseline, Gotovos et al. (2018) proposed a novel sampler named M^3 which uses discrete semigradients (discussed in Section 6.1) to efficiently perform global moves in the state space, allowing to improve the performances of the new sampler w.r.t. Gibbs. This approach requires computing a permutation of the ground set $\mathcal{V}$ using the Greedy algorithm (Nemhauser et al., 1978) for set submodular maximization w.r.t. a set of semigradients that is expanded at each iteration, starting from $\emptyset$. Even though this strategy if effective for both log-submodular and log-supermodular models, the authors noticed that subgradients yield better results in case of log-submodular models, and that, analogously, supergradients give better results for log-supermodular models.

5.5 Monte Carlo Methods

A Monte Carlo method is a probabilistic approach used to estimate quantities that are hard to compute exactly. Usually, these methods are divided into two categories:

- **Sequential Monte Carlo**, which maintains and propagates a pool of examples by sequential sampling and importance re-weighing, in particular in a low-dimensional space.
- **Markov Chain Monte Carlo** (MCMC), which simulates a Markov chain that explores the state space of the model with a stationary probability, hopefully converging to a given target probability.

In particular, in engineering applications such as machine learning and computer vision defined on either sets or graph representations, Monte Carlo methods are part of the *exact model with asymptotic global computing* paradigm. In this context, Monte Carlo methods simulate a large enough number of samples over time, reaching the globally optimal solution with high probability. These methods serve many different purposes, including estimating a quantity through Monte Carlo integration or optimizing a target probability function to find its modes (i.e., the maxima or minima), but we are mainly interested in simulating a system and its probability distribution π without computing the probability distribution explicitly. We thus focus on Markov Chain Monte Carlo methods, which can be used to obtain samples from a posterior probability.

5.5.1 Markov Chain Monte Carlo

MCMC indicates a class of stochastic methods we can use to sample from probability distributions which are known only up to a normalization constant $\mathcal{Z}$. Before proceeding further, it is important we clarify why sampling is an interesting task in the first place.

5.5.2 On Sampling

Sampling is the process of choosing a random state, called *sample*, from a probability distribution. This process is useful whenever we are interested in either the samples themselves, e.g., when inferring unknown parameters for a Bayesian inference task, or when we want to estimate expected values of functions w.r.t. a probability distribution. In other cases, we might only need the mode of a probability distribution, i.e., its maxima or minima. However, in this last case numerical optimization techniques are more efficient to apply, so full sampling is not required.

Perhaps unsurprisingly, sampling from any but the most simple probability distributions is a complex task. For instance, *inverse transform sampling* (Gass & Fu, 2013) is a basic sampling method, but it requires the cumulative distribution function (CDF) to be known, which in turn implies the knowledge of the partition function $\mathcal{Z}$. *Acceptance-rejection sampling* (Casella et al., 2004), on the other hand, does not need to evaluate or estimate $\mathcal{Z}$, but it requires a priori knowledge of the distribution of interest, and it suffers from the curse of dimensionality[3].

How to obtain representative samples from a target distribution without requiring knowledge of $\mathcal{Z}$, then? MCMC methods are a popular class of algorithms that address that question, which date back to a seminal paper of Metropolis et al. in 1953.

5.6 MCMC Sampling for Probabilistic Submodular Models

We focus our attention on Markov chain Monte Carlo sampling algorithms, which iteratively select random moves in a state space Ω to approximate target probabilistic quantities. The sequence of visited states $(X_0, X_1, \dots)$ composes a Markov chain, which converges to a stationary distribution π under mild and reasonable conditions (see Theorem 4.9 in Levin, 2009). In particular, the probabilities of moving from one state to another are such that the stationary distribution is identical to the target distribution which one wants to sample from. This process of hopping from state to state is called *state transition*. In probabilistic submodular models, the state space is the domain of the set submodular function f, the powerset $2^{\mathcal{V}}$; i.e., $\Omega := 2^{\mathcal{V}}$. The goal is to build a chain over subsets $A \subseteq \mathcal{V}$ that reaches a stationary distribution equivalent to π.

Transition matrix. We denote the *transition matrix* $P : \Omega \times \Omega \to \mathbb{R}$ of a Markov chain as

$$P(A, B) := Pr\left[X_{t+1} = B \,|\, X_t = A\right],$$

[3]The curse of dimensionality refers to a rapid decrease of efficiency in mathematical and computational methods as the number of variables involved increase, due to disturbing phenomena that only arise in high-dimensional spaces. This term was first introduced in Bellman (1961).

for all $A, B \in \Omega$.

Mixing time. The *mixing time* of a Markov chain is the smallest number of iterations t required for the distribution of X_t to get "close" to the stationary distribution π, for some definition of *vicinity*. More formally, the mixing time is defined as

$$t_{\text{mix}}(\epsilon) := \min\{t \mid d(t) \leq \epsilon\},$$

where $d(t)$ indicates the worst-case total variation distance between the distribution of X_t and π (w.r.t. the starting state X_0 of the chain), i.e.,

$$d(t) := \max_{X_0 \in \Omega} \|P^t(X_0, \cdot) - \pi\|_{\text{TV}}.$$

A known result that generalizes Chebyshev's inequality in the context of Markov chain samples (see Theorem 12.19, Levin, 2009) illustrates that finding an upper bound on the mixing time is a sufficient condition to guarantee efficient approximate sampling-based marginal inference.

We now present two famous Markov chains that constitute the most well-known MCMC samplers: the Gibbs and Metropolis samplers. They asymptotically converge to the unique stationary distribution π because they are *reversible* w.r.t. $\pi(\cdot) \propto \exp(f(\cdot))$, i.e., they satisfy the balance conditions

$$\pi(A)P(A, B) = \pi(B)P(B, A),$$

for all $A, B \in \Omega$ and where $P : \Omega \times \Omega \to \mathbb{R}$ is the state transition matrix.

5.6.1 Gibbs Sampler

The (single-site) Gibbs sampler, also known by the name of Glauber dynamics, is a chain that either adds or removes a single element of a time w.r.t. the previous state. It first selects an element $e \in \mathcal{V}$ uniformly at random, then it either includes it or removes it from the current state X_t depending on the probability of the resulting state. We illustrate the Gibbs sampler in Algorithm 5.1, where we use following:

$$\Delta_f(e \mid X_t) = [\![e \notin X_t]\!] f(e \mid X_t) + [\![e \in X_t]\!] f(e \mid X_t \setminus \{e\}).$$

For each $A, B \in \Omega$, this transition rule implies that A and B differ by one and only one element, namely, $||B| - |A|| = 1$. It is immediate to see that every $A \in \Omega$ has n neighbors.

In Fig. 5.1, we show an example step of the Gibbs sampler on a simple tridimensional ground set $\mathcal{V} := \{1, 2, 3\}$. Considering $X_t := \{2\}$ as the current state, there are 3 possible adjacent next states to transition to, i.e., $\emptyset$, $\{1, 2\}$, $\{2, 3\}$. One important thing to remark here is that the conditional probabilities computed by the Gibbs sampler are not influenced by the partition function $\mathcal{Z}$. This has the immediate consequence that this chain can be simulated efficiently, even though the unknown normalizing constant $\mathcal{Z}$ is generally hard to compute. Furthermore, the Gibbs sampler only needs a value oracle for the marginal gains of f.

Algorithm 5.1: Gibbs sampler for probabilistic submodular models.

```
   Input :Ground set $\mathcal{V}$, number of samples $M$, set submodular function $f$.
1  $X_0 \leftarrow$ random subset of $\mathcal{V}$;
2  for $t = 0$ to $M - 1$ do
3      Sample $e \in \mathcal{V}$ uniformly at random;
4      $p_{add} \leftarrow \exp(-\Delta_f(e \mid X_t))/(1 + \exp(-\Delta_f(e \mid X_t)))$;
5      Sample $z \in [0, 1]$ uniformly at random;
6      if $z \leq p_{add}$ then
7          $X_{t+1} \leftarrow X_t \cup \{e\}$;
8      else
9          $X_{t+1} \leftarrow X_t \setminus \{e\}$;
10     yield $X_{t+1}$
```

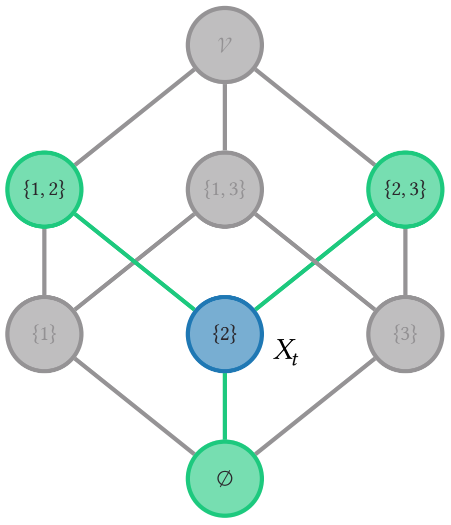


Figure 5.1: An example illustration of a single step of the Gibbs sampler on the small ground set $\mathcal{V} := \{1, 2, 3\}$, considering $X_t = \{2\}$ as the current state. Only one of the exactly $n = 3$ neighbors of X_t is selected uniformly at random, and the next state will be either X_t (again) or the selected neighbor, depending on the corresponding conditional probability.

Algorithm 5.2: Procedure to draw a candidate next state $S \sim q(\cdot \mid X_t)$, where q is the uniform distribution in $[0, 1]$.

```
Input :Current state $X_t \in \Omega$, probability $p_\tau$ of perturbating each possible $e \in S$.
1 $S \leftarrow X_t$;
2 for $e \in \mathcal{V}$ do
3     Sample $\tau \in [0, 1]$ uniformly at random;
4     if $\tau \le p_\tau$ then
5         if $e \in S$ then
6             $S \leftarrow S \setminus \{e\}$;
7         else
8             $S \leftarrow S \cup \{e\}$;
9 return $S$
```

5.6.2 Metropolis Sampler

Another famous and well-studied chain is the Metropolis sampler (Metropolis et al., 1953; Hastings, 1970), which also performs local transitions between neighboring states, using a slightly more complicated procedure than the Gibbs sampler. In particular, Metropolis draws a potential next state according to a proposal distribution $q(\cdot \mid X_t)$, and it determines whether to transition to the candidate state according to the probability ratio of the two local states adjusted by the proposal distribution ratio; more formally, the probability of acceptance is given by

$$p_{acc}(A, B) := \min\left\{1, \frac{q(A \mid B)\, \pi(B)}{q(B \mid A)\, \pi(A)}\right\} = \min\left\{1, \frac{q(A \mid B)\, \exp(-f(B))}{q(B \mid A)\, \exp(-f(A))}\right\}. \tag{5.12}$$

Conditional Set Sampling For generality, we consider the uniform distribution in $[0, 1]$ as the proposal distribution $q(\cdot)$. This implies that the conditional probabilities in Eq. (5.12) cancel each other out, obtaining a simplified definition for the probability of acceptance:

$$p_{acc}(A, B) := \min\left\{1, \frac{\pi(B)}{\pi(A)}\right\} = \min\left\{1, \frac{\exp(-f(B))}{\exp(-f(A))}\right\}. \tag{5.13}$$

Determining how to sample the next state from the chosen distribution $q(\cdot \mid X_t)$ is not immediate. We propose in Algorithm 5.2 a simple procedure for drawing a candidate next state $S \sim q(\cdot \mid X_t)$ that requires $\mathcal{O}(n)$ running time: we start from $S := X$, and we iteratively try to add or remove some elements $e \in \mathcal{V}$ from the candidate state S.

We show the Metropolis sampler with unitary uniform q in Algorithm 5.3.

Similarly to the Gibbs sampler, the acceptance probabilities p_{acc} do not depend on the value of $\mathcal{Z}$, hence the Metropolis chain can be simulated efficiently.

Algorithm 5.3: Metropolis sampler for probabilistic submodular models.

```
Input : Ground set $\mathcal{V}$, number of samples $M$, set submodular function $f$.
1  $X_0 \leftarrow$ random subset of $\mathcal{V}$;
2  for $t = 0$ to $M - 1$ do
3      Sample $S \sim q(\cdot \mid X_t)$;
4      $p_{acc} \leftarrow \min\left\{1, \frac{\exp(-f(S))}{\exp(-f(X_t))}\right\}$;
5      Sample $z \in [0, 1]$ uniformly at random;
6      if $z \leq p_{acc}$ then
7          $X_{t+1} \leftarrow S$;
8      else
9          $X_{t+1} \leftarrow X_t$;
10     yield $X_{t+1}$
```

5.7 Concluding Remarks

In this chapter, we have probabilistic submodular models in the context of Bayesian inference, reviewing a number of practical applications that, even though were originated in statistical physics, are now relevant for computer vision and machine learning tasks. We have also reviewed the most relevant works in the area of probabilistic submodular models, which range from variational inference to Markov chain Monte Carlo sampling strategies. Moreover, we have seen how to perform discrete MCMC sampling using the Gibbs and Metropolis samplers in the context of probabilistic submodular models.

In the following chapter, we will expand on the *semigradients* idea used by Gotovos et al. (2018), and we will sketch a novel sampler that uses the subgradient of the Lovász extension of a log-submodular set function f.

6 A Lovász Projection MCMC Sampler for Bayesian Inference

We recall that a probabilistic submodular model is a distribution defined over n categorical random binary variables of the form

$$\pi(S;\theta) = \frac{1}{Z(\theta)} exp(-f(S;\theta)), \tag{6.1}$$

for all $S \subseteq \mathcal{V}$, where $f : 2^{\mathcal{V}} \to \mathbb{R}$ is a set submodular function.

In Section 5.4, we have seen that Gotovos et al. (2018) proposed a Markov chain Monte Carlo (MCMC) sampler that uses semigradients to sample from the posterior distribution of a probabilistic submodular model, beating the performances of a Gibbs sampler in many cases. A similar usage of gradients emerged in the MCMC Langevin-type algorithms, which are based on the discretization of Langevin dynamics stochastic differential equation, and which use a Metropolis rejection/acceptance step after every iteration of this method (Roberts & Tweedie, 1996).

In this chapter, we will sketch an idea for an alternative MCMC sampler for Bayesian inference of log-submodular distributions based on the Lovász extension and the (sub-)gradient projection technique. First of all, we clarify the definition of *semigradients* of set functions, used in Gotovos et al. (2018).

6.1 Semigradients of Set Functions

Semigradients are modular functions that generalize the concept of *gradient* of a set function. In particular, semigradients provide lower approximations (*subgradients*) and upper approximations (*supergradients*) of a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$ (Fujishige, 2005; Iyer et al., 2013a).

Definition 6.1 (Subgradient). *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set function, and let $A \subseteq \mathcal{V}$. A modular function $m : 2^{\mathcal{V}} \to \mathbb{R}$ is called* subgradient *of f at S if, for all $B \subseteq \mathcal{V}$, it holds that*

$$f(B) \geq f(A) + m(B) - m(A).$$

Definition 6.2 (Supergradient). *Let $f : 2^{\mathcal{V}} \to \mathbb{R}$ be a set function, and let $A \subseteq \mathcal{V}$. A modular function $m : 2^{\mathcal{V}} \to \mathbb{R}$ is called* supergradient *of f at S if, for all $B \subseteq \mathcal{V}$, it holds that*

$$f(B) \leq f(A) + m(B) - m(A).$$

Note that, in general, a set function f is not guaranteed to have subgradients or supergradients at each $A \subseteq \mathcal{V}$. However, whenever f is either submodular or supermodular, it is known that f has semigradients at each $A \subseteq \mathcal{V}$ (Fujishige, 2005; Jegelka & Bilmes, 2011; Iyer & Bilmes, 2012).

6.2 On the Lovász Extension

We recall that the Lovász extension (Lovász, 1983), also known as Choquet integral (Choquet, 1954), is equivalent to the convex closure of a set function $f : 2^{\mathcal{V}} \to \mathbb{R}$, and that its usage emerges in set submodular minimization (recall Section 2.4.2 and Section 3.1.1). We denote the Lovász Extension of f by $\hat{f} : [0,1]^{\mathcal{V}} \to \mathbb{R}$. We have seen that the Lovász Extension is a convex function if and only if f is submodular (a proof of this statement can be found in Proposition 3.6 of Bach, 2013).

6.2.1 Subgradients of the Lovász Extension

Given a set submodular function f and its Lovász extension $\hat{f}$, we denote by $\boldsymbol{z} \in \partial \hat{f}(\boldsymbol{x})$ every subgradient of $\hat{f}$ at $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$. Since $\hat{f}$ is convex, it holds that a subgradient exists at every feasible point $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$, and that the subgradient is unique if and only if $\hat{f}$ is differentiable at that point.

Remark. *The Lovász extension $\hat{f}$ at a point $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$ is defined as a convex combination of a permutation of the elements of the ground set $\mathcal{V}$ and is a piecewise linear function, hence it is continuous but* not necessarily differentiable. *This implies that the Lovász extension may have more than one subgradient.*

There is a simple and generic way of computing subgradients for the Lovász extension. Let $\mathcal{V}' = \{v_1, v_2, \dots, v_n\}$ be a permutation of $\mathcal{V}$ such that $x_{v_1} \geq x_{v_2} \geq \cdots \geq x_{v_n}$. We denote $S_i := \{v_1, \dots, v_i\}$ for $0 \leq i \leq n$. The subgradient at $\boldsymbol{x}$ is defined as

$$\nabla^{-}_{\hat{f}(x_t)} := \sum_{i=1}^{n} \left(f(S_i) - f(S_{i-1})\right) v_i. \tag{6.2}$$

6.3 Gradient Projection

Given a probabilistic submodular model defined as Eq. (6.1), it is clear that maximizing π is equivalent to minimizing the set submodular function f, which is as well equivalent to minimizing the convex function $\hat{f}$, i.e., the Lovász extension of f. We thus consider a problem of the type

$$\begin{aligned} \min \quad & \hat{f}(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} \in [0,1]^{\mathcal{V}} \end{aligned} \tag{6.3}$$

Goldstein (1964) formulated a well-known solution to the problem Eq. (6.3) known as *gradient projection* algorithm[1], which was also independently proposed by Levitin & Polyak (1966) two years later. This iterative method starts from a feasible $\boldsymbol{x}_0 \in [0,1]^{\mathcal{V}}$, and computes the next point as $\boldsymbol{x}_t := \boldsymbol{x}_{t-1} - \nabla \hat{f}(\boldsymbol{x}_{t-1})$. Whenever the newly computed point exceeds the bounds of the feasible convex set $[0,1]^{\mathcal{V}}$, it is *projected* to a feasible point. We consider the Euclidean projection rule, in which every coordinate of $\boldsymbol{x}_t$ that is greater than 1 is set to 1, and every coordinate of $\boldsymbol{x}_t$ that is less than 0 is set to 0. We denote one Euclidean projection step as $\textsc{EuclideanProj}_{[0,1]}(\cdot)$.

6.4 Towards a novel Lovász Subgradient Projection MCMC Sampler

We decided to combine the Lovász extension $\hat{f}$ of the log-submodular function f to simulate with the gradient projection algorithm of Goldstein (1964), creating a new experimental sampler. The key points of this novel idea are:

- Using a subgradient of $\hat{f}$ instead of the true gradient in the projection step, so that we can yield a more general sampler that does not require a value oracle for the true gradient ∇f (which is not available in general, and which may be unreliable to approximate with, e.g., the finite differences method);
- Adding a zero-centered Gaussian *noise* to the subgradient of $\hat{f}$, so that the sampler does not converge to the mode of the distribution π (as we want to simulate sampling from the log-submodular distribution);
- Scaling the gradient step size by a factor η, to observe the difference in behavior for several values of η.

We devised four slightly different "Lovász projection" algorithms, which differ either for the update policy of η (fixed to a given constant or strictly descending), or for the way samples are drawn from each subgradient projection step at every iteration. We call this family of algorithms as LSubProj. Similarly to our discussion of MCMC samplers in Section 5.6, we use M to denote the number of samples to compute, with the idea that hopefully a higher number of M yields a better accuracy in simulating the log-submodular distribution π.

6.4.1 LSubProj-Metropolis

The first algorithm of the LSubProj family combines the Lovász subgradient projection step with a Metropolis-Hastings update rule (recall our discussion of the Metropolis sampler in Section 5.6.2). For this reason, we name this sampler as LSubProj-Metropolis, which we illustrate in Algorithm 6.1.

6.4.2 LSubProj-Metropolis-Desc

LSubProj-Metropolis-Desc is equivalent to LSubProj-Metropolis, but with a strictly descending η hyperparameter. We show it in Algorithm 6.2.

[1]The gradient projection algorithm can solve more general problems than Eq. (6.3), in which x is required to be a generic convex set in $\mathbb{R}^n$

Algorithm 6.1: LSubProj-Metropolis sampler for probabilistic submodular models.

```
Input : Ground set $\mathcal{V}$, number of samples $M$, set submodular function $f$, standard
        deviation $\sigma$, gradient step size $\eta$.
$X_0 \leftarrow$ random subset of $\mathcal{V}$;
$\boldsymbol{x}_0 \leftarrow \mathbf{1}_{X_0}$;
for $t = 0$ to $M-1$ do
    $N \sim \mathcal{N}(0, \sigma^2)$;
    $\boldsymbol{y}_t \leftarrow \boldsymbol{x}_t - (\eta \cdot \nabla^-_{\hat{f}(\boldsymbol{x}_t)}) - (\sqrt{\eta} \cdot N)$;
    $\boldsymbol{s}_t \leftarrow \text{EuclideanProj}_{[0,1]}(\boldsymbol{y}_t)$;
    Sample $S_t$ from the coordinates of $\boldsymbol{s}_t$ uniformly at random;
    $p_{acc} \leftarrow \min\left\{1, \frac{\exp(-f(S_t))}{\exp(-f(X_t))}\right\}$;
    Sample $z \in [0,1]$ uniformly at random;
    if $z \leq p_{acc}$ then
        $X_{t+1} \leftarrow S$;
        $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{s}_t$;
    else
        $X_{t+1} \leftarrow X_t$;
        $\boldsymbol{x}_{t+1} \leftarrow \boldsymbol{x}_t$;
    yield $X_{t+1}$
```

6.4.3 LSubProj-Cont

We notice that LSubProj-Metropolis and LSubProj-Metropolis-Desc only require a $\mathcal{O}(n)$ space occupation, as just two continuous states $X_t, X_{t+1} \in \Omega$ are needed for each iteration of the chain (the same holds for the Gibbs sampler discussed in Section 5.6.1 and for the Metropolis sampler discussed in Section 5.6.2). LSubProj-Cont is significantly different than the previous Lovász subgradient projection samplers, as it:

- keeps in memory the whole set of sample states before they are returned;
- creates an empirical frequency table for each possible subset $S \subseteq \mathcal{V}$ whose frequency values depend on the values of $\boldsymbol{s}_t \in [0,1]$, the candidate next vector at each iteration t.

It is perhaps easier to go through an example first. Consider $\mathcal{V} := \{1,2,3\}$. Let us suppose that, after a Lovász subgradient projection step, we obtain $\boldsymbol{s} := [0.1, 0.4, 0.3]$ (we drop the iteration subscript t for clarity). We then explicitly define a map $\tau : 2^{\mathcal{V}} \to \mathbb{R}_+$, which defines how likely each subset $S \subseteq \mathcal{V}$ occurs w.r.t. $\boldsymbol{s}$ according to the following logic:

Algorithm 6.2: LSubProj-Metropolis-Desc sampler for probabilistic submodular models.

```
Input : Ground set $\mathcal{V}$, number of samples $M$, set submodular function $f$, standard
        deviation $\sigma$.
1  $X_0 \leftarrow$ random subset of $\mathcal{V}$;
2  $x_0 \leftarrow \mathbf{1}_{X_0}$;
3  for $t = 0$ to $M - 1$ do
4      $\eta \leftarrow \frac{1}{2+t}$;
5      $N \sim \mathcal{N}(0, \sigma^2)$;
6      $y_t \leftarrow x_t - (\eta \cdot \nabla^-_{\hat{f}(x_t)}) - (\sqrt{\eta} \cdot N)$;
7      $s_t \leftarrow \textsc{EuclideanProj}_{[0,1]}(y_t)$;
8      Sample $S_t$ from the coordinates of $s_t$ uniformly at random;
9      $p_{acc} \leftarrow \min\left\{1, \frac{\exp(-f(S_t))}{\exp(-f(X_t))}\right\}$;
10     Sample $z \in [0,1]$ uniformly at random;
11     if $z \leq p_{acc}$ then
12         $X_{t+1} \leftarrow S$;
13         $x_{t+1} \leftarrow s_t$;
14     else
15         $X_{t+1} \leftarrow X_t$;
16         $x_{t+1} \leftarrow x_t$;
17     yield $X_{t+1}$
```

$$
\begin{aligned}
\tau(\emptyset) &:= 1 - \max(s) \\
\tau(\{1\}) &:= s_1 \cdot (1 - s_2) \cdot (1 - s_3) \\
\tau(\{2\}) &:= (1 - s_1) \cdot s_2 \cdot (1 - s_3) \\
\tau(\{3\}) &:= (1 - s_1) \cdot (1 - s_2) \cdot s_3 \\
\tau(\{1,2\}) &:= s_1 \cdot s_2 \cdot (1 - s_3) \\
\tau(\{1,3\}) &:= s_1 \cdot (1 - s_2) \cdot s_3 \\
\tau(\{2,3\}) &:= (1 - s_1) \cdot s_2 \cdot s_3 \\
\tau(\{1,2,3\}) &:= s_1 \cdot s_2 \cdot s_3
\end{aligned}
$$

More formally, we define $\tau(\cdot)$ as

$$
\tau(S) := \begin{cases} 1 - \max(\boldsymbol{s}) & \text{if } S = \emptyset \\ \prod_{i \in S} s_i \prod_{j \in \mathcal{V} \setminus S} (1 - s_j) & \text{else} \end{cases} \tag{6.4}
$$

Algorithm 6.3: LSubProj-Cont sampler for probabilistic submodular models.

```
Input : Ground set $\mathcal{V}$, number of samples $M$, set submodular function $f$, standard
        deviation $\sigma$, gradient step size $\eta$.
1  $X_0 \leftarrow$ random subset of $\mathcal{V}$;
2  $x_0 \leftarrow \mathbf{1}_{X_0}$;
3  for $t = 0$ to $M - 1$ do
4      $N \sim \mathcal{N}(0, \sigma^2)$;
5      $y_t \leftarrow x_t - (\eta \cdot \nabla^-_{\hat{f}(x_t)}) - (\sqrt{\eta} \cdot N)$;
6      $s_t \leftarrow \textsc{EuclideanProj}_{[0,1]}(y_t)$;
7      Define $\tau_t(\cdot)$ using Eq. (6.4);
8  $\tau'(S) \leftarrow [\tau_0(S), \tau_1(S), \cdots, \tau_{M-1}(S)]$ for every $S \subseteq \mathcal{V}$;
9  $\phi(S) \leftarrow \left\{\frac{1}{M} \cdot \sum \tau'(S)\right\}$ for every $S \subseteq \mathcal{V}$;
10 $\pi'(A) \leftarrow \left\{\frac{\phi(A)}{\sum_{B \subseteq \mathcal{V}} \phi(B)}\right\}$ for every $A \subseteq \mathcal{V}$;
11 $Xs \leftarrow \{X_i \mid X_i \sim \pi', 0 \leq i < M\}$;
12 return $Xs$
```

for a given $\mathbf{s} \in [0, 1]$, and for every $S \subseteq \mathcal{V}$.

We keep track of each τ across M iterations of the chain, merging them together into a map $\tau' : 2^{\mathcal{V}} \rightarrow \mathbb{R}^M_+$. Effectively, $\tau'(S)$ tracks how likely it is that S should be sampled at each iteration $t < M$. Notice that the values returned by $\tau'(S)$ are not probabilities! We then compute a table of empirical frequencies $\phi : 2^{\mathcal{V}} \rightarrow \mathbb{R}_+$ defined as

$$\phi(S) := \left\{\frac{1}{M} \cdot \sum \tau'(S)\right\},$$

which is then normalized with

$$\pi'(A) := \left\{\frac{\phi(A)}{\sum_{B \subseteq \mathcal{V}} \phi(B)}\right\}$$

to obtain a map of probabilities $\pi' : 2^{\mathcal{V}} \rightarrow [0, 1]$. Finally, we can compute the entire chain by sampling M sets from π'.

6.4.4 LSubProj-Cont-Desc

LSubProj-Cont-Desc is equivalent to LSubProj-Cont, but with a strictly descending η hyperparameter. We show it in Algorithm 6.4.

Algorithm 6.4: LSubProj-Cont sampler for probabilistic submodular models.

```
Input : Ground set $\mathcal{V}$, number of samples $M$, set submodular function $f$, standard
        deviation $\sigma$.
1  $X_0 \leftarrow$ random subset of $\mathcal{V}$;
2  $x_0 \leftarrow \mathbf{1}_{X_0}$;
3  for $t = 0$ to $M - 1$ do
4      $\eta \leftarrow \frac{1}{2+t}$;
5      $N \sim \mathcal{N}(0, \sigma^2)$;
6      $y_t \leftarrow x_t - (\eta \cdot \nabla_{\hat{f}(x_t)}^{-}) - (\sqrt{\eta} \cdot N)$;
7      $s_t \leftarrow$ EuclideanProj$_{[0,1]}(y_t)$;
8      Define $\tau_t(\cdot)$ using Eq. (6.4);
9      $x_{t+1} \leftarrow s_t$;
10 $\tau'(S) \leftarrow [\tau_0(S), \tau_1(S), \cdots, \tau_{M-1}(S)]$ for every $S \subseteq \mathcal{V}$;
11 $\phi(S) \leftarrow \left\{\frac{1}{M} \cdot \sum \tau'(S)\right\}$ for every $S \subseteq \mathcal{V}$;
12 $\pi'(A) \leftarrow \left\{\frac{\phi(A)}{\sum_{B \subseteq \mathcal{V}} \phi(B)}\right\}$ for every $A \subseteq \mathcal{V}$;
13 $Xs \leftarrow \{X_i \mid X_i \sim \pi', 0 \leq i < M\}$;
14 return $Xs$
```

6.5 An Analysis of the Lovász Subgradient Projection methods

We denote as EO the number of value oracle queries for f (or, equivalently, for $\hat{f}$).

Theorem 6.3. *The LSubProj-Metropolis algorithm requires $\mathcal{O}(M \cdot n \cdot EO \log n)$ running time, and $\mathcal{O}(n)$ space.*

Proof. A single iteration of LSubProj-Metropolis requires $\mathcal{O}(n \cdot EO \log n)$ theoretical running time, since we compute a subgradient of $\hat{f}$ applying the greedy permutation sorting procedure of Lovász (1983). Computing p_{acc} requires a constant number of oracle queries. Every other line of Algorithm 6.2 takes at most $\mathcal{O}(n)$. Since there are M iterations, the LSubProj-Metropolis algorithm requires $\mathcal{O}(M \cdot n \cdot EO \log n)$ running time in total. For what concerns the space analysis, we have already noted that only two consecutive states X_t and X_{t+1} are required at each iteration (and their respective vectors), which implies a $\mathcal{O}(n)$ requirements. □

Theorem 6.4. *The LSubProj-Metropolis-Desc algorithm requires $\mathcal{O}(M \cdot n \cdot EO \log n)$ running time.*

Proof. The core of LSubProj-Metropolis-Desc is identical to LSubProj-Metropolis. The only thing that changes is the η that becomes strictly descending, but this policy update requires constant time for each iteration. □

Theorem 6.5. *The LSubProj-Metropolis algorithm requires $\mathcal{O}(M \cdot n \cdot EO \log n)$ running time.*

It is evident that LSubProj-Cont and LSubProj-Cont-Desc are unfeasible in practice. Defining a set function like τ explicitly requires generating and iterating the powerset of $\mathcal{V}$, which requires $\mathcal{O}(2^n)$ time and $\mathcal{O}(n \cdot 2^n)$ space. The explicit evaluation of τ', ϕ, and π' requires $\mathcal{O}(M \cdot 2^n)$ time and space. The LSubProj-Cont and LSubProj-Cont-Desc are practically unfeasible by design, to allow us to observe its behavior at runtime without worrying on formulating space-compression approaches, to formulate hypothesis for possible future improvements using similar (but less expensive) ideas. It is however interesting evaluating the performances of the samplers defined above in terms of how close they simulate a given log-submodular distribution π.

6.6 Experiments

We compare our sampler algorithms to the following two baselines:

- The Gibbs sampler of Algorithm 5.1, which we denote as Gibbs;
- The Metropolis sampler of Algorithm 5.3, which we denote as Metropolis.

For Metropolis, we set the probability p_τ of perturbing each possible $e \in S$, for each possible $S \subseteq \mathcal{V}$, to the fixed value of 0.1.

We also consider the following hyperparameters for the Lovász projection algorithms:

- subgradient step size $\eta \in \{0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001\}$; (does not apply for the LSubProj-Metropolis-Desc and LSubProj-Cont-Desc samplers);
- standard deviation for the Gaussian noise $\sigma \in \{0.1, 0.5, 1, 2\}$.

For every sampler under study, we run them to obtain $M \in \{1000, 10000, 100000\}$ samples. We also consider 0.2 as the initial *burn-in time*. That is, if we want to obtain M samples, we actually run the samplers for $M + 0.2 \cdot M$ iterations, and discard the first $0.2 \cdot M$ samples. This is a common practice when evaluating MCMC methods, as it tends to produce more stable results.
We conducted our experiments on a Linux server cluster gratefully provided by the “Research Center for Informatics” (part of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765) equipped with two Intel Xeon CPUs and 384 GBs of RAM. The code for the experiments is written in Python 3.8. We let the experiments run for 168 hours.

6.6.1 Metrics Evaluation

We use the *total variation distance* over finite sets as a metric to compare the distance between the samples generated by the samplers and the ground truth distribution π. In particular, we fist compute the ground truth probability, creating a map that associates each $S \subseteq \mathcal{V}$ to the probability that S is actually sampled from π. We then compute an *history* of each sampler, i.e., the list of samples yielded in chronological order (we omit the burn-in samples). The *history* is broken down in a chain $\mathcal{B}$ of batches B with step size 50, maintaining the chronological order, obtaining a list of cumulative probabilities. This means that the first batch B_1 contains $\{X_1, \dots, X_{50}\}$, the second batch $B_2 \supseteq B_1$ contains $\{X_1, \dots, X_{50}, \dots, X_{100}\}$, and so on, with the last batch containing $\{X_1, \dots, X_M\}$. We then count the number of occurrences of each $S \subseteq \mathcal{V}$ in these batches, and divide it by the total number of occurrences (w.r.t. the size of each batch, not M), obtaining an empirical discrete probability distribution π'_B for each batch. We then calculate the total variation distance between the empirical distribution π'_B and the ground truth distribution π for each batch $B \in \mathcal{B}$ as

$$d(\pi, \pi'_B) := \sum_{S \subseteq \mathcal{V}} |\pi(S) - \pi'_B(S)|. \tag{6.5}$$

If we plot the value of $d(\pi, \pi'_B)$, one would hope to observe a quasi-monotonically decreasing function, which means that, the larger the time window considered, the better the convergence of the sampling algorithm to the true distribution π (whose partition function $\mathcal{Z}$ is unknown to the sampler).

6.6.2 Problem Instances

We consider two simple set submodular functions $f : 2^{\mathcal{V}} \to \mathbb{R}$, both defined on ground sets of size $n := 3$ and $n := 5$. We use them to define the following probabilistic submodular model:

$$\pi(S) := \frac{1}{\sum_{S \subseteq \mathcal{V}} \exp(-f(S))} \exp(-f(S)). \tag{6.6}$$

Synthetic Monotone Function The first set submodular instance is the set equivalent of the synthetic monotone function used in Section 4.5:

$$f(S) := \mathbf{1}_S \cdot \boldsymbol{w}, \tag{6.7}$$

for each $S \subseteq \mathcal{V}$ and for some random $\boldsymbol{w} \in [0,1]^{\mathcal{V}}_{+}$ such that $w_i \leq w_j$ for each $i < j$.

Loss Function The second set submodular instance is the Δ_4 loss function defined in Eq. 4.7.5 of Yu (2017), which is defined as

$$f(S) := 1 - \exp(-\alpha \cdot |S|) \tag{6.8}$$

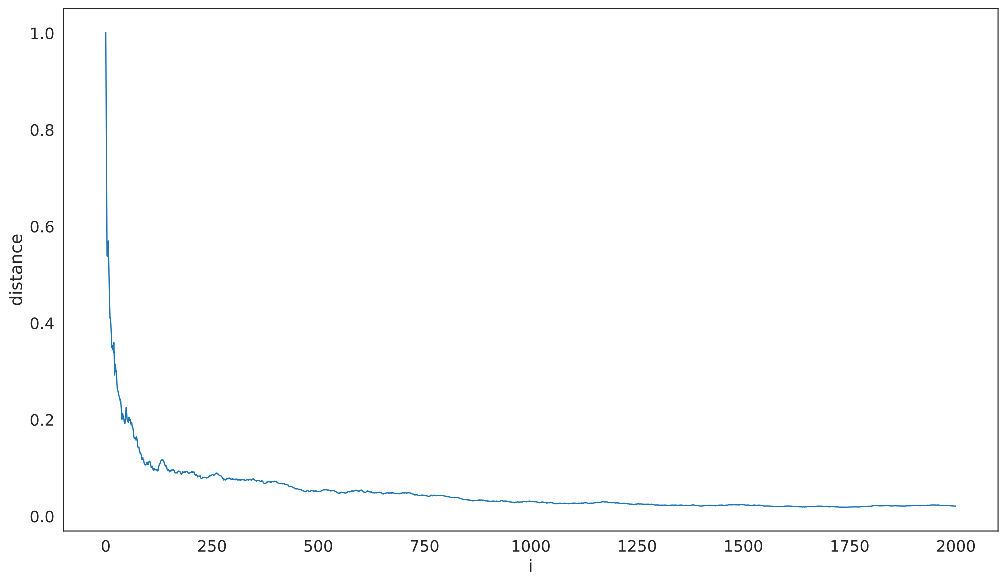
for each $S \subseteq \mathcal{V}$ and for some $\alpha > 0$. In particular, we set $\alpha := 0.5$.

6.6.3 Numerical Results and Insights

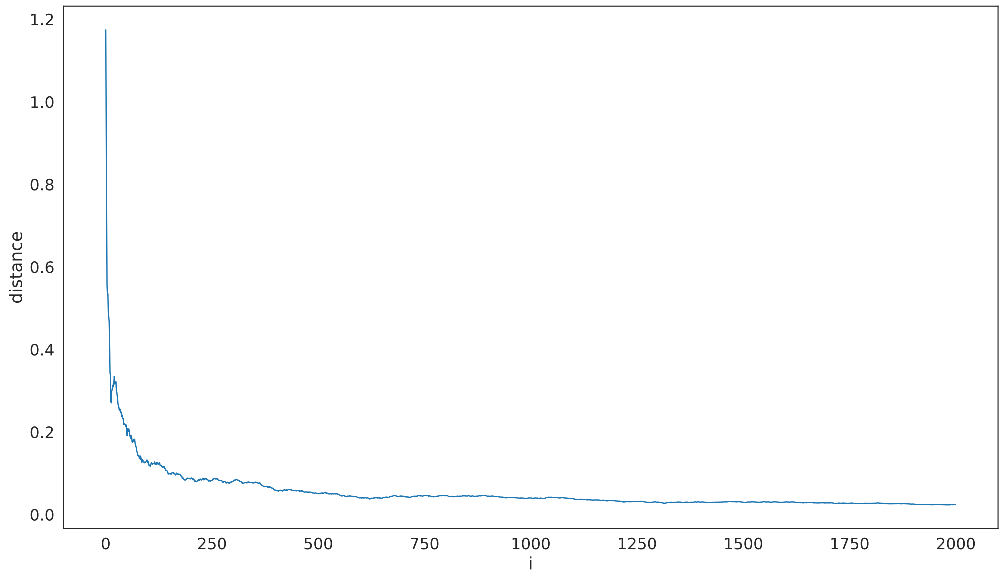
When $n = 3$, Gibbs and Metropolis tend to demonstrate convergence to the true distribution π already after a few iterations, even for $M < 1000$. $n = 5$ seems to be too high-dimensional for Gibbs to yield good results with less than $M = 100000$ samples, but Metropolis continuous convering successfully (albeit with a slower mixing rate). Even though both our synthetic monotone function and the loss function instances have a simple definition, the loss function is better approximated in general, for any sampler. This may be due to the fact that Eq. (6.8) is a strictly concave function.

For the Lovász projection algorithms, $\sigma := 1$ for the standard deviation of the Gaussian noise performs best in general. Both $\sigma < 1$ and $\sigma > 1$ tend to cause an abrupt divergence for the LSubProj-Metropolis and LSubProj-Metropolis-Desc samplers, but not so much for the LSubProj-Cont and LSubProj-Cont-Desc samplers. η has drastic impacts depending on the interactions with the other hyperparameters, but $\eta = 0.001$ seems to be a nice default step size value that performs best on average.

We observed that LSubProj-Metropolis-Desc performs very poorly in all instances, with the cumulative probability distance having either an increasing or an oscillating trend and does not decrease under 0.92 even in its best case (with $M = 100000$ and $\sigma = 1.0$). This is particularly evident in Fig. 6.3, where LSubProj-Metropolis-Desc is compared to LSubProj-Cont-Desc in the loss function instance with $n = 5$.

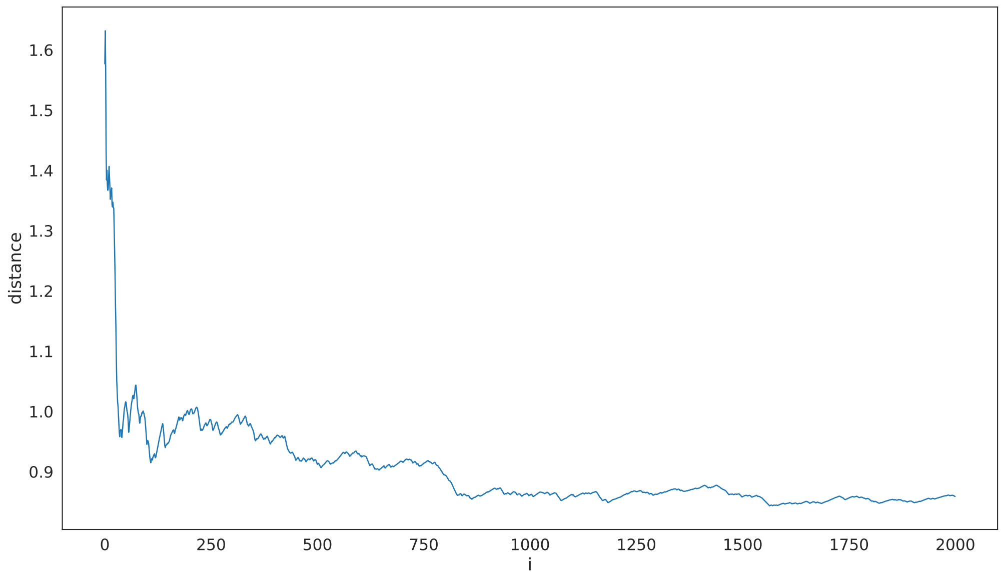


(a) Cumulative variation distance for the Gibbs sampler.

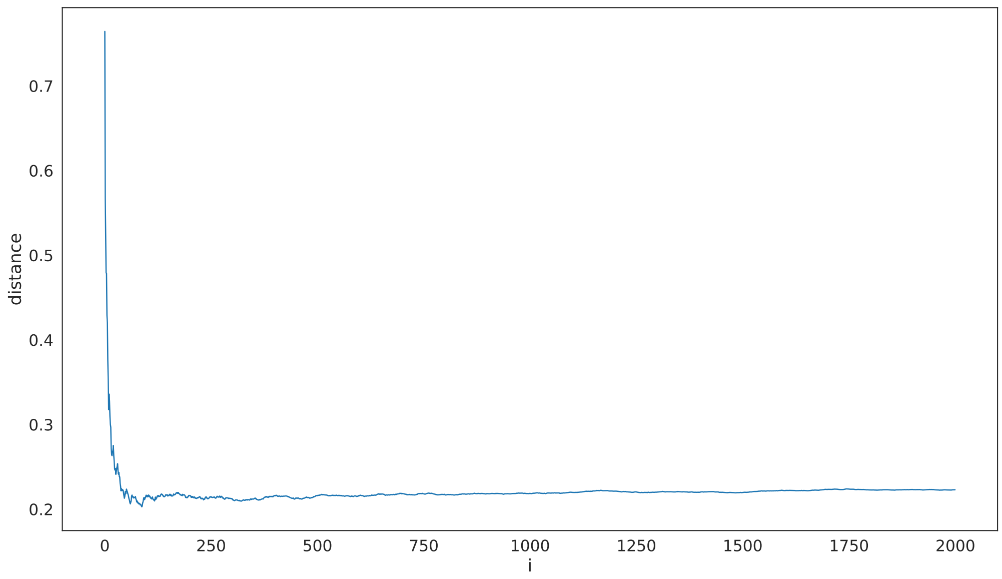


(b) Cumulative variation distance for the Metropolis sampler.

Figure 6.1: An illustration of the cumulative variation distance of the Gibbs and Metropolis samplers for the loss function in Eq. (6.8) for $n = 5$ in batches B_i.



(a) Cumulative variation distance for the LSubProj-Metropolis sampler.

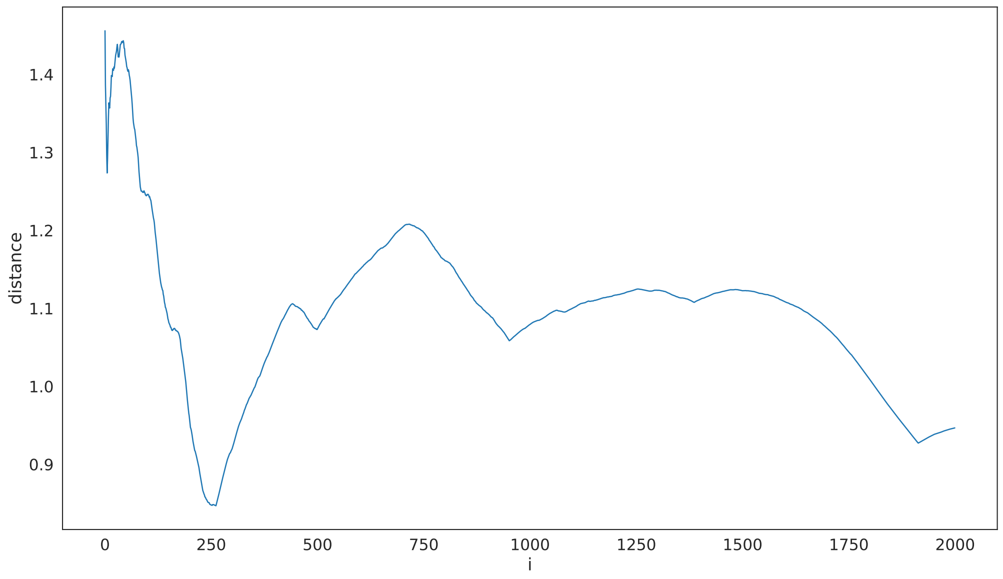


(b) Cumulative variation distance for the LSubProj-Cont sampler.

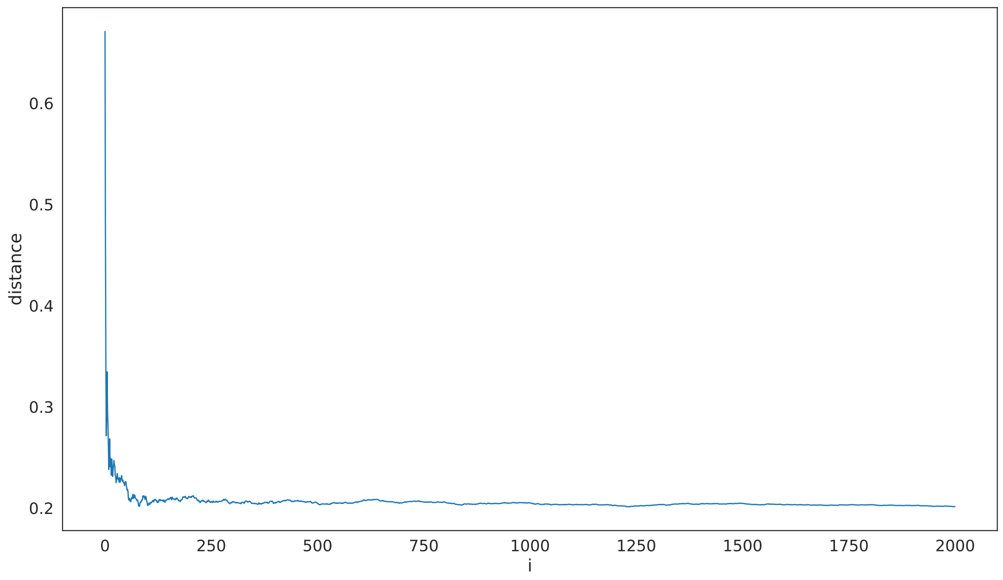
Figure 6.2: An illustration of the cumulative variation distance of the LSubProj-Metropolis and LSubProj-Cont samplers for the loss function in Eq. (6.8) for $n = 5$ in batches B_i.

6.7 Concluding Remarks

In this chapter, we have explored a few ideas related to implementing MCMC samplers that leverage the Lovász extension and its easily computable subgradient. We have attempted to improve the performance of the Metropolis sampler using a convex optimization approach, the gradient projection method, which we have applied to the subgradient of the Lovász extension of the given log-submodular distribution π. We also experimented with a global state transition approach that carries the likelihood of appearance of every possible subset for each iteration. Moreover, we ran



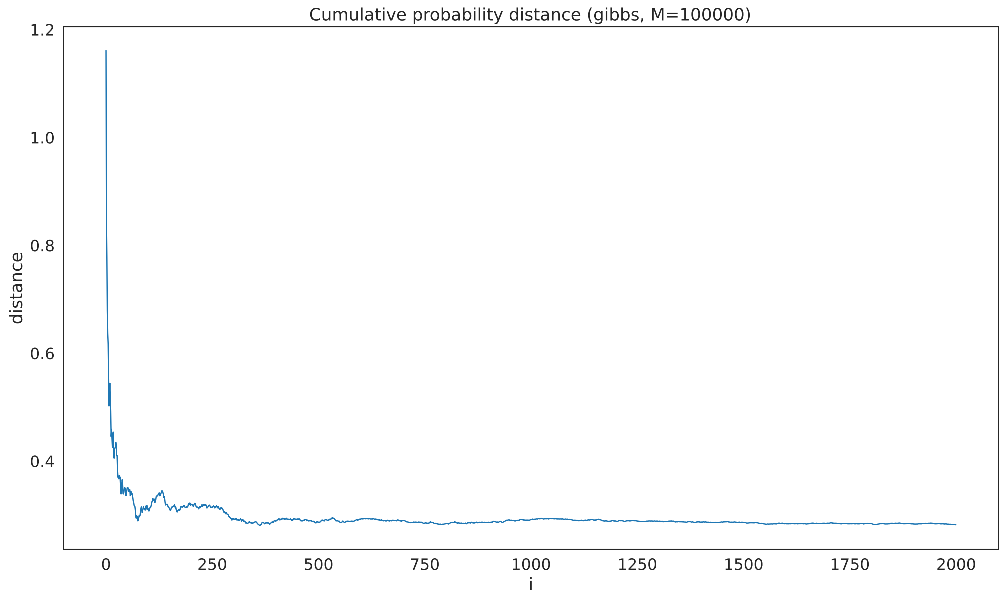
(a) Cumulative variation distance for the LSubProj-Metropolis-Desc sampler.



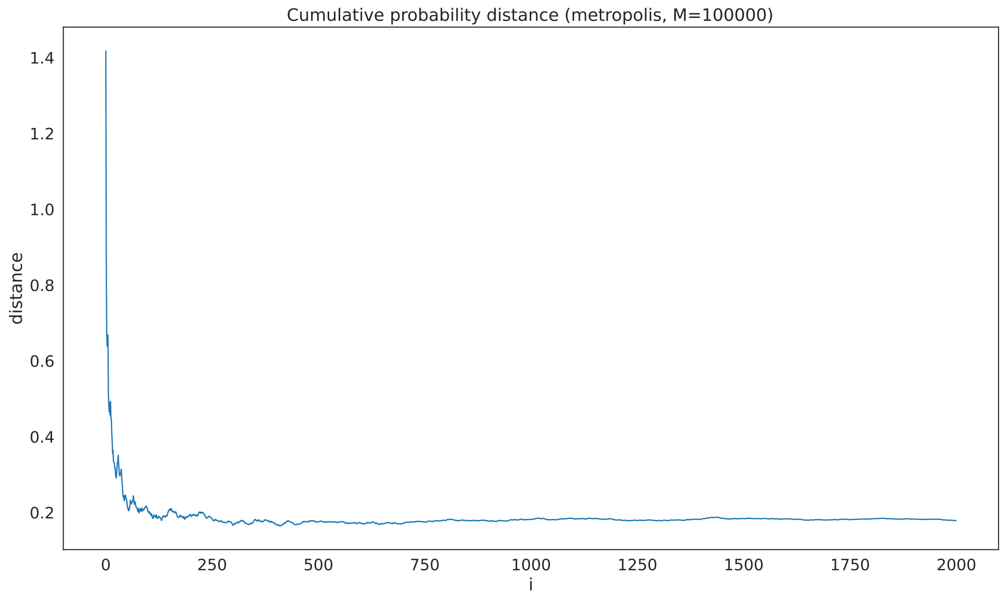
(b) Cumulative variation distance for the LSubProj-Cont-Desc sampler.

Figure 6.3: An illustration of the cumulative variation distance of the LSubProj-Metropolis-Desc and LSubProj-Cont-Desc samplers for the loss function in Eq. (6.8) for $n = 5$ in batches B_i.

several experiments on a simple monotone probabilistic submodular model, observing how different hyperparameters influence the outcome. Unfortunately, we were not able to improve upon the results of either the Gibbs or Metropolis samplers, used as baseline. We believe that aiming for a general approach that does not require to know the gradient of a submodular function f might have backfired: we know that a subgradient of f w.r.t. a vector $\boldsymbol{x} \in [0,1]^{\mathcal{V}}$ is a lower bound on the true gradient, but we do not know how far we can go in the direction of the true gradient exactly. This is evidently a fundamental limitation when applying the gradient projection method.



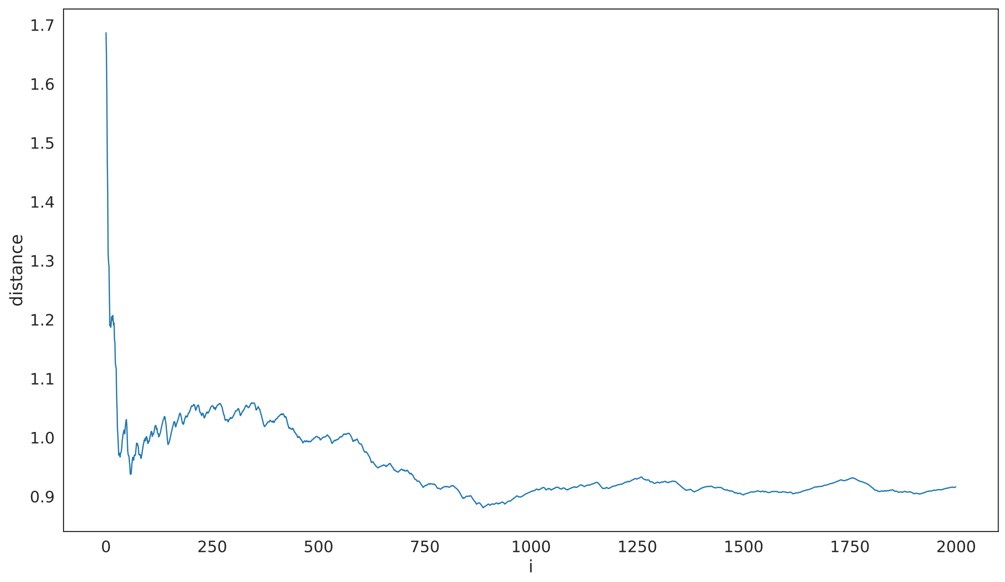
(a) Cumulative variation distance for the Gibbs sampler.



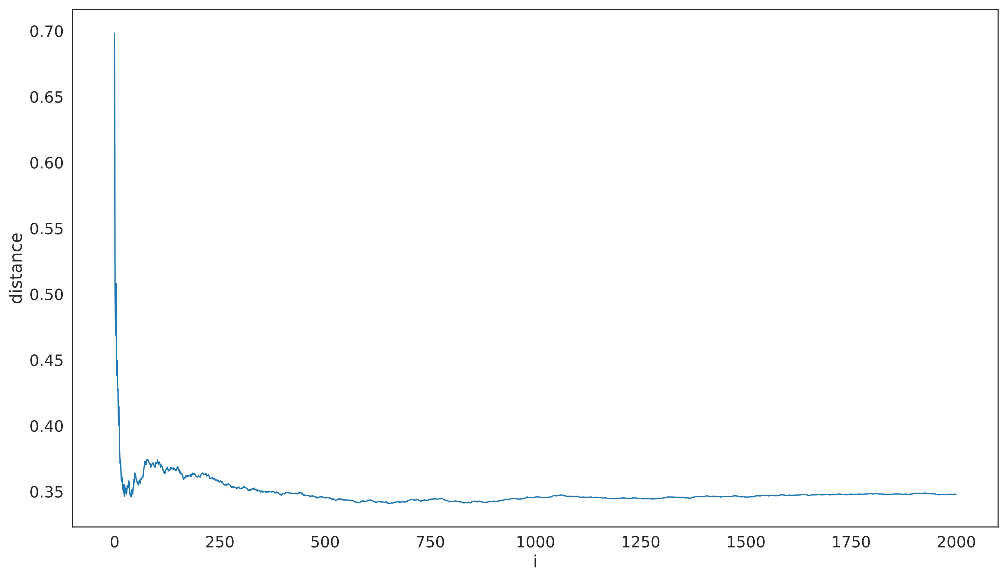
(b) Cumulative variation distance for the Metropolis sampler.

Figure 6.4: An illustration of the cumulative variation distance of the Gibbs and Metropolis samplers for the synthetic monotone function in Eq. (6.7) for $n = 5$ in batches B_i.

Another problem is that the LSubProj-Cont algorithm is not feasible in practice, as it essentially builds a simulated approximation π' of the ground truth distribution π using exponential space. On the other hand, the metric we chose to use (the only available metric actually, considering that we do not have guarantees about the convergence of our approaches) requires several passes over the entire powerset of $\mathcal{V}$ (as it must consider the partition function $\mathcal{Z}$ of π explicitly), which is impractical as well as it requires exponential time and space. It could however be worth exploring how well LSubProj-Cont would behave if we only built small portions of π' at each major iteration, keeping the same overall logic but considering only a fixed number of possible samples



(a) Cumulative variation distance for the LSubProj-Metropolis sampler.

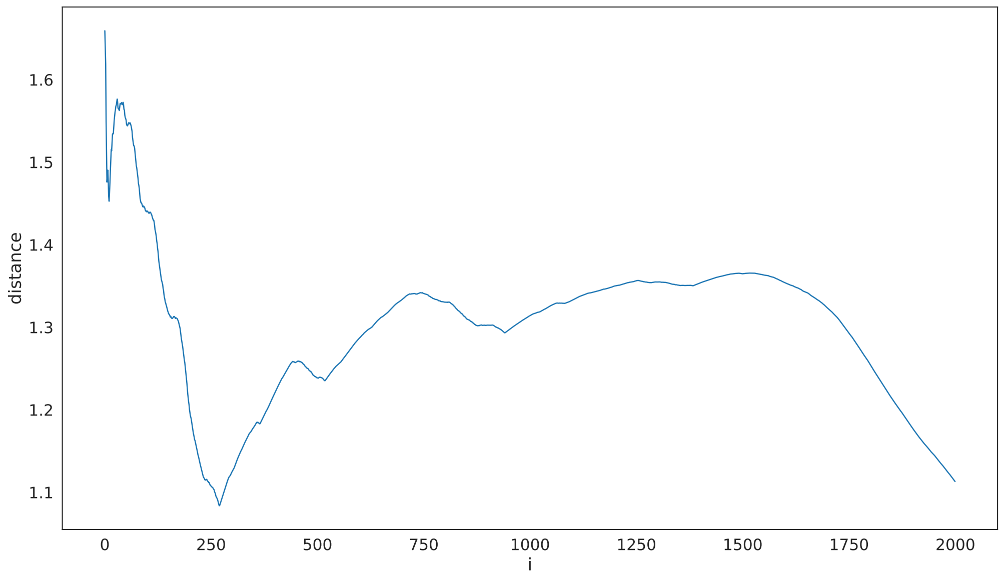


(b) Cumulative variation distance for the LSubProj-Cont sampler.

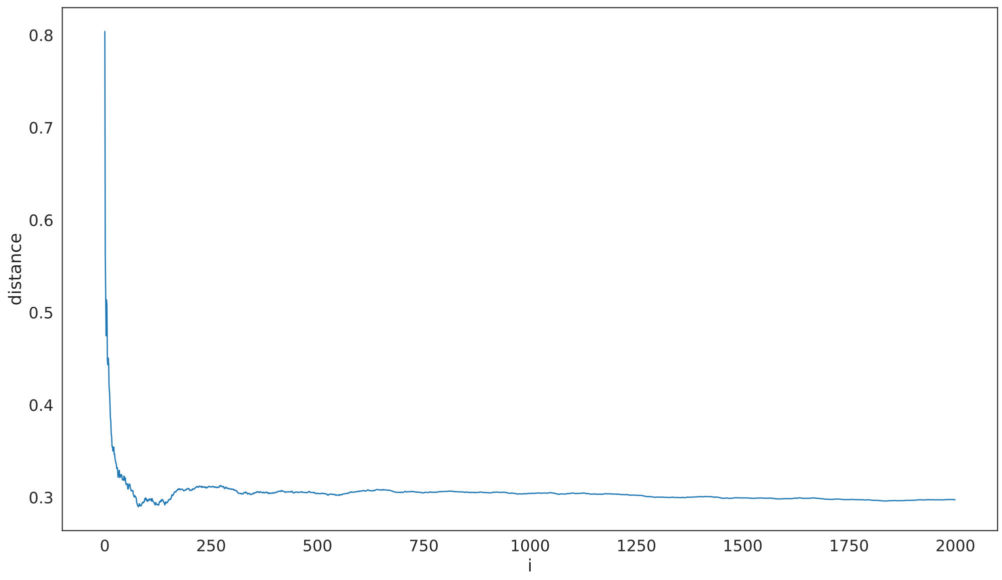
Figure 6.5: An illustration of the cumulative variation distance of the LSubProj-Metropolis and LSubProj-Cont samplers for the synthetic monotone function in Eq. (6.7) for $n = 5$ in batches B_i.

rather than the entire powerset of the feasible spaces in $\Omega := 2^{\mathcal{V}}$.

Despite the somewhat discouraging results we obtained, the Lovász subgradient projection approach is the most promising alternative sampler we have come up with in the past months (we tried too many slightly different approaches to list them exhaustively here). For instance, we had tried to transform the Fujishige-Wolfe algorithm for set submodular minimization (using again the Lovász extension) into a sampler, but we did not identify any clear pattern between the different combinations of hyperparameters and the results of the sampler in terms of total variation distance, and the performance of the sampler was still far from Gibbs and Metropolis.



(a) Cumulative variation distance for the LSubProj-Metropolis-Desc sampler.



(b) Cumulative variation distance for the LSubProj-Cont-Desc sampler.

Figure 6.6: An illustration of the cumulative variation distance of the LSubProj-Metropolis-Desc and LSubProj-Cont-Desc samplers for the synthetic monotone function in Eq. (6.7) for $n = 5$ in batches B_i.

In a future revision of the work in this chapter, it could be interesting to consider the multilinear extension rather than the Lovász extension, and compare the results w.r.t. the Lovász extension.

7 Conclusions

In this thesis, we have given an overview of the recent advances in submodular optimization, analyzing the impact of generalizing problems from the *binary* set domain to the integer lattice domain, and we have discussed practical applications of these optimization algorithms.

We have reasoned about submodular functions sharing some properties with both convex and concave functions, and explained the implications in terms of continuous extensions (Lovász and multilinear), minimization, and maximization. We have reviewed both discrete and partially continuous algorithms for optimizing submodular functions with constraints, and we have discussed about randomized algorithms and monotone submodular functions often yielding better guarantees and results.

We have then discussed about submodular function maximization defined on the integer lattice, and we have presented a family of six novel algorithms for the cardinality-constrained version of this problem. With both empirical and theoretical reasoning, we managed to improve the state of the art in this area, and also presented the first known open source implementation of the Soma-DR (Soma & Yoshida, 2018) and Lai-DR (Lai et al., 2019) algorithms.

Furthermore, we have introduced the topic of probabilistic submodular models, its origins and applications, with a focus on Markov chain Monte Carlo samplers for Bayesian inference. We highlighted the difficulty of sampling from log-submodular distributions even though the underlying submodular function is known. We have finally experimented with a new class of algorithms that attempt to sample from a log-submodular distribution without requiring the computation of the partition function explicitly using properties of the Lovász extension and the gradient projection method. We proposed both a fast Metropolis-like sampler algorithm that performs local transitions between candidate samples, and a slower, more accurate algorithm that uses a novel global transition scheme.

7.1 Open Source Software

We have released the code, configuration, and pipelines written during the course of the thesis as open source software, released under the MIT license on Github.
In particular, the code repository for the comparative analysis and implementation of the algorithms for submodular maximization on the integer lattice subject to a cardinality constraint discussed in Chapter 4 is available at `https://github.com/jkomyno/lattice-submodular-maximization`.

The code repository for the analysis and implementation of the Markov chain Monte Carlo samplers for probabilistic submodular models discussed in Chapter 6 is available at *https://github.com/jkomyno/probabilistic-submodular-models*.

We have used Python 3.8 (with type hints) as the main programming language, *numpy* (Harris et al., 2020) as the numerical library of reference, *pandas* (pandas development team, 2020) to share intermediate CSV results and to generate the tables in the report, *seaborn* (Waskom, 2021) for plotting, and *hydra* (Yadan, 2019) as the experiments' configuration layer.

7.2 Future Work

We list here a few possibly promising directions for future work related to the topics touched in this thesis.

Online Submodular Maximization on the Integer Lattice In this work, we focus our attention to the "offline" kind of submodular maximization, i.e., where the ground set $\mathcal{V}$ of elements is known in advance. However, an additional challenge would be exploring how a direct translation of our randomized SGL algorithms for submodular maximization on the integer lattice subject to a cardinality constraint would fare in the online context, and how to extend these algorithms to better handle this setting. This is particularly interesting since most of the literature in online settings only considers the relatively simple case of set submodular functions.

Multilinear Projection Sampler We have observed some shortcomings of our Lovász subgradient projection approach in Chapter 6, and highlighted that they may be due an excessive underestimation of the true gradient of the submodular function, since we only used a subgradient induced by the Lovász extension of a target submodular function. It would be interesting to consider an even more stochastic approach using the multilinear extension, which is unfeasible to compute exactly, but that can be easily estimated in polynomial time.

A Reduction from Integer Lattice to Set Domain

In this chapter, we present an extract of the abstract classes and utility functions needed to implement a reduction of a submodular optimization problem from the integer lattice to the set domain. In particular, we consider submodular maximization subject to a cardinality constraint, which the Stochastic Greedy algorithm (Mirzasoleiman et al., 2015) solves in $\mathcal{O}(n \log(\frac{1}{\varepsilon}))$ oracle queries with a $(1 - \frac{1}{e} - \varepsilon)$ approximation guarantee (on average) in the case of set submodular functions. We previously introduced the Stochastic Greedy algorithm in Section 3.2.4, and we built upon it to implement our novel algorithms for submodular maximization subject to a cardinality constraint in the integer lattice domain in Chapter 4.

We now recall the formulation of the considered problem in the set submodular and integer lattice submodular domains.

Set Submodular Maximization subject to a Cardinality Constraint Given $f : 2^{\mathcal{V}} \to \mathbb{R}_+$ a non-negative normalized set submodular function and a cardinality constraint $r \in \mathbb{Z}_+$, we want to solve the following problem:

$$\begin{aligned} \max \quad & S \\ \text{s.t.} \quad & \|S\|_1 \leq r \\ & \forall S \in \mathcal{V} \\ & r < |V| \\ & r \in \mathbb{Z}_+ \end{aligned} \tag{A.1}$$

Integer Lattice Submodular Maximization subject to a Cardinality Constraint Given $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}_+$ a non-negative normalized integer lattice submodular function, and a non-negative vector $\boldsymbol{b} \in \mathbb{Z}_+^{\mathcal{V}}$ representing the quantities available for each element $e \in \mathcal{V}$ of the ground set, and a cardinality constraint $r \in \mathbb{Z}_+$, we want to solve the following problem:

$$\begin{aligned} \max \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \|\boldsymbol{x}\|_1 \leq r \\ & x_e \leq b_e & \text{for all } e \in \mathcal{V} \\ & \boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}, \boldsymbol{b} \in \mathbb{Z}_+^{\mathcal{V}}, r \in \mathbb{Z}_+ \end{aligned} \tag{A.2}$$

A.1 Python Code

In the following, we present the code we used to reduce the solution of Eq. (A.2) to Eq. (A.2), allowing us to use the set-based Stochastic Greedy algorithm for integer lattice submodular problems as well. We used Python 3.8 as the programming language, due its versatility and simple syntax.

algorithm 1 defines the base abstract class `Objective` for an integer lattice submodular function, i.e., the contract that any instance of an integer lattice submodular objective should respect. The ground set $\mathcal{V}$ and the vector of available quantities $\boldsymbol{b} \in \mathbb{Z}_+^{\mathcal{V}}$ are required a priori, and they are retrieved via the `V` and `B` class properties, respectively. Analogously, algorithm 2 defines the base abstract class `SetObjective` for a set submodular function.

Consider the ground set $\mathcal{V} := \{0, 1, 2\}$ (in our code implementation, we use 0-based indexes as opposed to 1-based indexes adopted for the mathematical notation). Then, for example, let $\boldsymbol{b} := [2, 3, 2]$. This means that we can select element 0 twice, element 1 three times, and element 2 twice. How to represent this "multiset" using the familiar set submodular framework? It turns out that it is quite easy once we define a new, "expanded" ground set $\mathcal{V}'$. We define it as $\mathcal{V}' := \{0, 1, \dots, \|\boldsymbol{b}\|_1 - 1\}$, and we map the integer lattice submodular function $f : \mathbb{Z}_+^{\mathcal{V}} \to \mathbb{R}_+$ to the equivalent set submodular function $f' : 2^{\mathcal{V}'} \to \mathbb{R}_+$.

algorithm 3 shows a utility to convert a set $S \subseteq \mathcal{V}'$ into a vector $\boldsymbol{x} \in \mathbb{Z}_+^{\mathcal{V}}$ defined on the integer lattice. Similarly, algorithm 4 presents a utility to expand an integer lattice submodular function f into a set submodular function f'. algorithm 5 shows the implementation of the Stochastic Greedy algorithm by Mirzasoleiman et al. (2015), which uses the `SetObjective` abstract class. Finally, algorithm 6 presents the *simulated* version of the Stochastic Greedy algorithm, i.e., the SSG algorithm mentioned in Chapter 4.

```python
from abc import ABC
from typing import List, Tuple
from nptyping import NDArray

class Objective(ABC):
    def __init__(self, ground_set: List[int], B: NDArray[int]):
        """
        Base abstract class for an integer lattice submodular function
        :param ground_set: ground set of f
        :param B: vector of available quantities for each element in
                  the ground_set
        """
        self._ground_set = ground_set
        self._n = len(ground_set)
        self._B = B

    @property
    def V(self) -> List[int]:
        """Return the ground set"""
        return self._ground_set

    @property
    def B(self) -> NDArray[int]:
        """Return the known vector of available quantities"""
        return self._B

    @property
    def n(self) -> int:
        """Return the size of the ground set"""
        return self._n

    def value(self, x: NDArray[int]) -> int:
        """
        Value oracle for the submodular problem.
        :param x: n-dimensional non-negative vector
        :return: value oracle for x
        """
        pass
```

Listing 1: Base abstract class for an integer lattice submodular function

```
import numpy as np
from abc import ABC
from typing import List, AbstractSet
from nptyping import NDArray

class SetObjective(ABC):
    def __init__(self, ground_set: List[int], B: NDArray[int]):
        """
        Base abstract class for a set submodular function that emulates
        an integer lattice submodular function.
        :param ground_set: ground set of f
        :param b: upper bound of the integer lattice domain of f
        """
        self._n = len(ground_set)
        self._B = B
        self._ground_set = set(range(np.sum(B)))

    @property
    def V(self) -> AbstractSet[int]:
        """Return the extended ground set"""
        return self._ground_set

    @property
    def n(self) -> int:
        """Return the size of the extended ground set"""
        return len(self._ground_set)

    @property
    def original_n(self) -> int:
        """Return the size of the ground set in the integer lattice"""
        return self._n

    def value(self, S: AbstractSet[int]) -> int:
        """
        Value oracle for the submodular problem.
        :param S: subset of the ground set
        :return: value oracle for S in the submodular problem
        """
        pass
```

Listing 2: Base abstract class for a set submodular function that emulates an integer lattice submodular function.

```python
import numpy as np
from typing import AbstractSet
from nptyping import NDArray
from collections import Counter

def to_integer_lattice(f: SetObjective,
                       S: AbstractSet[int]) -> NDArray[int]:
    """
    Convert a set submodular solution to an integer lattice solution.
    :param f: set submodular function
    :param S: set submodular solution
    """
    # n is the size of the ground set in the integer lattice
    n = f.original_n
    counter = Counter((e % n for e in S))
    x = np.zeros((n, ), dtype=int)

    for e, c in counter.items():
        x[e] = c

    return x
```

Listing 3: Utility to convert a set submodular function into an integer lattice submodular function.

```python
from typing import AbstractSet

def to_set_objective(f: Objective) -> SetObjective:
    """
    Convert an integer lattice submodular function to a set submodular function
    via ground set expansion.
    """
    class SetObjectiveImpl(SetObjective):
        def __init__(self):
            super().__init__(ground_set=f.V, B=f.B)

        def value(self, S: AbstractSet[int]) -> int:
            """
            Value oracle for the set submodular problem.
            :param S: subset of the ground set
            :return: value oracle for S in the submodular problem
            """
            x = to_integer_lattice(self, S)
            return f.value(x)

    f_prime = SetObjectiveImpl()

    return f_prime
```

Listing 4: Utility to reduce a an integer lattice submodular function to a set submodular function.

```python
import numpy as np
from nptyping import NDArray
from typing import Set, Tuple

def stochastic_greedy(rng: np.random.Generator, f: SetObjective,
                      r: int, eps: float) -> Tuple[Set[int], float]:
    """
    Computes a set A \subseteq V such that |A| \leq r.
    :param rng: numpy random generator instance
    :param f: set-submodular function to maximize
    :param r: cardinality constraint
    :param eps: error threshold
    """
    # compute s, the sample size
    s = max(int(-np.log(eps) * n / r, 1))

    # the solution starts from the empty set
    A: Set[int] = set()

    # prev_value keeps track of the value of f(A)
    prev_value = 0

    while len(A) < r:
        # R is a random subset obtained by sampling s random elements
        # from V - A
        sample_space = list(f.V - A)
        R: NDArray[int] = rng.choice(sample_space, size=min(s,
         ↪  len(sample_space)), replace=False)
        prev_value, marginal_gain, a = max((
            (cand_value := f.value(A | {a}), cand_value - prev_value, a)
            for a in R
        ), key=lambda u: u[1])
        A.add(a)

    return A, prev_value
```

Listing 5: Stochastic Greedy algorithm for set submodular maximization subject to a cardinality constraint r.

```python
import numpy as np
from typing import Tuple
from nptyping import NDArray

def SSG(rng: np.random.Generator,
        f: Objective, r: int, eps: float) -> Tuple[NDArray[int], int]:
    """
    Simulated Stochastic Greedy algorithm in the integer lattice domain.
    :param rng: numpy random generator instance
    :param f: integer-lattice submodular function objective
    :param r: cardinality constraint
    :param eps: error threshold
    """
    f_prime: SetObjective = to_set_objective(f)
    S, value = set_algo.stochastic_greedy(rng, f_prime, r, eps)
    x = to_integer_lattice(f_prime, S)

    return x, value
```

Listing 6: Simulated Stochastic Greedy (SSG) algorithm for integer lattice submodular maximization subject to a cardinality constraint r.

Bibliography

Ageev, A. and Sviridenko, M. Pipage rounding: A new method of constructing algorithms with proven performance guarantee. *Journal of Combinatorial Optimization*, 8(3):307–328, September 2004. doi: 10.1023/b:joco.0000038913.96607.c2. URL *https://doi.org/10.1023/b:joco.0000038913.96607.c2*.

Agrawal, R., Squires, C., Yang, K., Shanmugam, K., and Uhler, C. Abcd-strategy: Budgeted experimental design for targeted causal structure discovery. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 3400–3409. PMLR, 2019.

Alon, N., Gamzu, I., and Tennenholtz, M. Optimizing budget allocation among channels and influencers. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, pp. 381–388, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312295. doi: 10.1145/2187836.2187888. URL *https://doi.org/10.1145/2187836.2187888*.

Anari, N., Gharan, S. O., and Rezaei, A. Monte Carlo Markov chain algorithms for sampling strongly Rayleigh distributions and determinantal point processes. In *Conference on Learning Theory (COLT)*, 2016.

Atallah, M. J. and Blanton, M. *Algorithms and Theory of Computation Handbook*. Chapman & Hall/CRC, 2nd edition, 2009. ISBN 1584888180.

Austrin, P., Benabbas, S., and Georgiou, K. Better balance by being biased: A 0.8776-approximation for max bisection. *ACM Trans. Algorithms*, 13(1), oct 2016. ISSN 1549-6325. doi: 10.1145/2907052. URL *https://doi.org/10.1145/2907052*.

Bach, F. Structured sparsity-inducing norms through submodular functions. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL *https://proceedings.neurips.cc/paper/2010/file/4b0a59ddf11c58e7446c9df0da541a84-Paper.pdf*.

Bach, F. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends (R) in Machine Learning*, 6(2-3):145–373, 2013. doi: 10.1561/2200000039. URL *https://doi.org/10.1561%2F2200000039*.

Bach, F. Submodular functions: from discrete to continuous domains. *Mathematical Programming*, 175(1):419–459, 2019.

Badanidiyuru, A. and Vondrák, J. Fast algorithms for maximizing submodular functions. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, pp. 1497–1514, USA, 2014. Society for Industrial and Applied Mathematics. ISBN 9781611973389.

Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., and Krause, A. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pp. 671–680, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329569. doi: 10.1145/2623330.2623637. URL `https://doi.org/10.1145/2623330.2623637`.

Bateni, M. H., Esfandiari, H., and Mirrokni, V. Optimal distributed submodular optimization via sketching. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, pp. 1138–1147, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3220081. URL `https://doi.org/10.1145/3219819.3220081`.

Bellman, R. *Adaptive Control Processes.* Princeton University Press, 1961.

Bian, A., Levy, K. Y., Krause, A., and Buhmann, J. M. Continuous dr-submodular maximization: Structure and algorithms. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pp. 486–496. Curran Associates Inc., 2017a. ISBN 9781510860964.

Bian, A. A., Mirzasoleiman, B., Buhmann, J. M., and Krause, A. Guaranteed non-convex optimization: Submodular maximization over continuous domains. In Singh, A. and Zhu, X. J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, AISTATS 2017, 20-22 April 2017, Fort Lauderdale, FL, USA*, volume 54 of *Proceedings of Machine Learning Research*, pp. 111–120. PMLR, 2017b. URL `http://proceedings.mlr.press/v54/bian17a.html`.

Borcea, J., Brändén, P., and Liggett, T. M. Negative dependence and the geometry of polynomials. *Journal of the American Mathematical Society*, 2008.

Buchbinder, N. and Feldman, M. Deterministic algorithms for submodular maximization problems. *ACM Trans. Algorithms*, 14(3), June 2018. ISSN 1549-6325. doi: 10.1145/3184990. URL `https://doi.org/10.1145/3184990`.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. A tight linear time (1/2)-approximation for unconstrained submodular maximization. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pp. 649–658, 2012. doi: 10.1109/FOCS.2012.73.

Buchbinder, N., Feldman, M., Naor, J., and Schwartz, R. Submodular maximization with cardinality constraints. In *Symposium on Discrete Algorithms (SODA)*, 2014.

Buschjäger, S., Honysz, P.-J., Pfahler, L., and Morik, K. Very fast streaming submodular function maximization. In *Machine Learning and Knowledge Discovery in Databases*, pp. 151–166. Springer International Publishing, 2021. doi: 10.1007/978-3-030-86523-8_10. URL `https://doi.org/10.1007/978-3-030-86523-8_10`.

Calinescu, G., Chekuri, C., Pál, M., and Vondrák, J. Maximizing a submodular set function subject to a matroid constraint. In *International Conference on Integer Programming and Combinatorial Optimization*, pp. 182–196. Springer, 2007.

Calinescu, G., Chekuri, C., Pál, M., and Vondrák, J. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM Journal on Computing*, 2011.

Casella, G., Robert, C. P., and Wells, M. T. Generalized accept-reject sampling schemes. In *Institute of Mathematical Statistics Lecture Notes - Monograph Series*, pp. 342–347. Institute of Mathematical Statistics, 2004. doi: 10.1214/lnms/1196285403. URL https://doi.org/10.1214/lnms/1196285403.

Chakrabarty, D., Jain, P., and Kothari, P. Provable submodular minimization using wolfe's algorithm. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pp. 802–809, Cambridge, MA, USA, 2014. MIT Press.

Choquet, G. Theory of capacities. *Annales de l'Institut Fourier*, 5:131–295, 1954. doi: 10.5802/aif.53. URL http://www.numdam.org/articles/10.5802/aif.53/.

Conforti, M. and Cornuejols, G. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the rado-edmonds theorem. *Discrete Applied Mathematics*, 7:251–274, 1984.

Cornuejols, G., Fisher, M., and Nemhauser, G. L. On the uncapacitated location problem. In Hammer, P., Johnson, E., Korte, B., and Nemhauser, G. (eds.), *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pp. 163–177. Elsevier, 1977. doi: https://doi.org/10.1016/S0167-5060(08)70732-5.

Crama, Y. and Hammer, P. L. Boolean functions - theory, algorithms, and applications. In *Encyclopedia of mathematics and its applications*, 2011.

Cunningham, W. H. On submodular function minimization. *Combinatorica*, 5(3):185–192, September 1985. doi: 10.1007/bf02579361. URL https://doi.org/10.1007/bf02579361.

Das, A. and Kempe, D. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, pp. 1057–1064, Madison, WI, USA, 2011. Omnipress. ISBN 9781450306195.

Djolonga, J. and Krause, A. From map to marginals: Variational inference in bayesian submodular models. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'14, pp. 244–252, Cambridge, MA, USA, 2014. MIT Press.

Djolonga, J., Tschiatschek, S., and Krause, A. Variational inference in mixed probabilistic submodular models. In *Neural Information Processing Systems (NIPS)*, 2016.

Dobzinski, S. and Mor, A. A deterministic algorithm for maximizing submodular functions, 2015.

Dueck, D. and Frey, B. J. Non-metric affinity propagation for unsupervised image categorization. *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, 2007.

Edmonds, J. Minimum partition of a matroid into independent subsets. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, 69B(1 and 2):67, 1965.

Edmonds, J. Matroids, submodular functions and certain polyhedra. *Combinatorial Structures and their Applications*, pp. 69–87, 1970.

Edmonds, J. Matroids and the greedy algorithm. *Mathematical Programming*, 1(1):127–136, December 1971. doi: 10.1007/bf01584082. URL https://doi.org/10.1007/bf01584082.

Edmonds, J. and Fulkerson, D. R. Transversals and matroid partition. *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics*, 69B(3):147, 1965.

Ene, A. and Nguyen, H. L. A reduction for optimizing lattice submodular functions with diminishing returns. *CoRR*, abs/1606.08362, 2016. URL http://arxiv.org/abs/1606.08362.

Feige, U. A threshold of $ln(n)$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998. ISSN 0004-5411. doi: 10.1145/285055.285059. URL https://doi.org/10.1145/285055.285059.

Feige, U., Mirrokni, V. S., and Vondrák, J. Maximizing non-monotone submodular functions. In *Symposium on Foundations of Computer Science*, 2007.

Feldman, M. and Karbasi, A. Continuous submodular maximization: Beyond dr-submodularity, 2020.

Feldman, M., Naor, J. S., and Schwartz, R. A unified continuous greedy algorithm for submodular maximization. In *Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pp. 570–579, USA, 2011. IEEE Computer Society. ISBN 9780769545714. doi: 10.1109/FOCS.2011.46. URL https://doi.org/10.1109/FOCS.2011.46.

Filmus, Y. and Ward, J. A tight combinatorial algorithm for submodular maximization subject to a matroid constraint. In *2012 IEEE 53rd Annual Symposium on Foundations of Computer Science*, pp. 659–668, 2012. doi: 10.1109/FOCS.2012.55.

Fisher, M. L., Nemhauser, G. L., and Wolsey, L. A. An analysis of approximations for maximizing submodular set functions—II. In *Mathematical Programming Studies*, pp. 73–87. Springer Berlin Heidelberg, 1978. doi: 10.1007/bfb0121195. URL https://doi.org/10.1007/bfb0121195.

Frank, M. and Wolfe, P. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, March 1956. doi: 10.1002/nav.3800030109. URL https://doi.org/10.1002/nav.3800030109.

Frankowski, D., Lam, S. K., Sen, S., Harper, F. M., Yilek, S., Cassano, M., and Riedl, J. Recommenders everywhere: The WikiLens community-maintained recommender system. In *Proc. Int. Symp. on Wikis*, pp. 47–60, 2007.

Frieze, A. and Jerrum, M. Improved approximation algorithms for MAXk-CUT and MAX BISECTION. *Algorithmica*, 18(1):67–81, May 1997. doi: 10.1007/bf02523688. URL https://doi.org/10.1007/bf02523688.

Frieze, A. M. A cost function property for plant location problems. *Mathematical Programming*, 7(1):245–248, December 1974. doi: 10.1007/bf01585521. URL *https://doi.org/10.1007/bf01585521*.

Fujishige, S. Lexicographically optimal base of a polymatroid with respect to a weight vector. *Mathematics of Operations Research*, 5(2):186–196, May 1980. ISSN 0364-765X. doi: 10.1287/moor.5.2.186. URL *https://doi.org/10.1287/moor.5.2.186*.

Fujishige, S. Submodular systems and related topics. In *Mathematical Programming Studies*, volume 2, pp. 113–131. Springer Berlin Heidelberg, 1984. doi: 10.1007/bfb0121012. URL *https://doi.org/10.1007/bfb0121012*.

Fujishige, S. *Submodular Functions and Optimization*. Elsevier, Amsterdam Boston Oxford, 2005. ISBN 9780444520869.

Fujishige, S. and Isotani, S. A submodular function minimization algorithm based on the minimum-norm base. In *Pacific Journal of Optimization*, volume 7, pp. 3–17, 2011.

Fujishige, S., Hayashi, T., and Isotani, S. The minimum-norm-point algorithm applied to submodular function minimization and linear programming. In *Research Institute for Mathematical Sciences (RIMS)*, Kyoto, Japan, 2006.

Gandhi, R., Khuller, S., Parthasarathy, S., and Srinivasan, A. Dependent rounding and its applications to approximation algorithms. *J. ACM*, 53(3):324–360, May 2006. ISSN 0004-5411. doi: 10.1145/1147954.1147956. URL *https://doi.org/10.1145/1147954.1147956*.

Gass, S. I. and Fu, M. C. (eds.). *Inverse Transform Method*. Springer US, Boston, MA, 2013. ISBN 978-1-4419-1153-7. doi: 10.1007/978-1-4419-1153-7_200343. URL *https://doi.org/10.1007/978-1-4419-1153-7_200343*.

Goemans, M. X. and Williamson, D. P. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, November 1995. ISSN 0004-5411. doi: 10.1145/227683.227684. URL *https://doi.org/10.1145/227683.227684*.

Goldstein, A. A. Convex programming in Hilbert space. *Bulletin of the American Mathematical Society*, 70(5):709 – 710, 1964. doi: bams/1183526263. URL *https://doi.org/*.

Golovin, D. and Krause, A. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *J. Artif. Int. Res.*, 42(1):427–486, September 2011. ISSN 1076-9757.

Gomes, R. and Krause, A. Budgeted nonparametric learning from data streams. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 391–398, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Gong, B., Chao, W.-L., Grauman, K., and Sha, F. Diverse sequential subset selection for supervised video summarization. In *Neural Information Processing Systems (NIPS)*, 2014.

Gotovos, A., Hassani, H. S., and Krause, A. Sampling from probabilistic submodular models. In *Neural Information Processing Systems (NIPS)*, 2015.

Gotovos, A., Hassani, H., Krause, A., and Jegelka, S. Discrete sampling using semigradient-based product mixtures. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2018.

Gottschalk, C. and Peis, B. Submodular function maximization on the bounded integer lattice. In *International Workshop on Approximation and Online Algorithms*, pp. 133–144. Springer, 2015.

Goundan, P. R. and Schulz, A. S. Revisiting the greedy approach to submodular set function maximization. 2007.

Grötschel, M., Lovász, L., and Schrijver, A. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, June 1981. doi: 10.1007/bf02579273. URL https://doi.org/10.1007/bf02579273.

Halperin, E. and Zwick, U. Combinatorial approximation algorithms for the maximum directed cut problem. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pp. 1–7, USA, 2001. Society for Industrial and Applied Mathematics. ISBN 0898714907.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

Hastings, W. K. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 1970.

Hatano, D., Fukunaga, T., Maehara, T., and Kawarabayashi, K.-i. Lagrangian decomposition algorithm for allocating marketing channels. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 1144–1150. AAAI Press, 2015. ISBN 0262511290.

Hausmann, D. and Korte, B. K-greedy algorithms for independence systems. *Zeitschrift für Operations Research*, 22(1):219–228, December 1978. doi: 10.1007/bf01917662. URL https://doi.org/10.1007/bf01917662.

Hausmann, D., Korte, B., and Jenkyns, T. A. Worst case analysis of greedy type algorithms for independence systems. In M.W., P. (ed.), *Combinatorial Optimization. Mathematical Programming Studies*, pp. 120–131. Springer, Berlin, Heidelberg, 1980. doi: 10.1007/bfb0120891. URL https://doi.org/10.1007/bfb0120891.

Ising, E. Beitrag zur theorie des ferromagnetismus. *Zeitschrift für Physik*, 1925.

Iwata, S. and Orlin, J. B. A simple combinatorial algorithm for submodular function minimization. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pp. 1230–1237, USA, 2009. Society for Industrial and Applied Mathematics.

Iwata, S., Fleischer, L., and Fujishige, S. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *J. ACM*, 48(4):761–777, July 2001. ISSN 0004-5411. doi: 10.1145/502090.502096. URL https://doi.org/10.1145/502090.502096.

Iyer, R. and Bilmes, J. The submodular Bregman and Lovász-Bregman divergences with applications. In *Neural Information Processing Systems (NIPS)*, 2012.

Iyer, R. and Bilmes, J. Submodular optimization with submodular cover and submodular knapsack constraints. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, volume 2 of *NIPS'13*, pp. 2436–2444, Red Hook, NY, USA, 2013. Curran Associates Inc.

Iyer, R., Jegelka, S., and Bilmes, J. Fast semidifferential-based submodular function optimization. In *International Conference on Machine Learning (ICML)*, 2013a.

Iyer, R., Jegelka, S., and Bilmes, J. Curvature and optimal algorithms for learning and minimizing submodular functions. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 2742–2750, Red Hook, NY, USA, 2013b. Curran Associates Inc.

Iyer, R., Jegelka, S., and Bilmes, J. Fast semidifferential-based submodular function optimization. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. III–855–III–863. JMLR.org, 2013c.

Iyer, R. K. and Bilmes, J. A. Polyhedral aspects of submodularity, convexity and concavity. *CoRR*, abs/1506.07329, 2015. URL `http://arxiv.org/abs/1506.07329`.

Jaggi, M. Revisiting frank-wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pp. I–427–I–435. JMLR.org, 2013.

Jegelka, S. and Bilmes, J. Submodularity beyond submodular energies: Coupling edges in graph cuts. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011.

Jegelka, S., Lin, H.-C., and Bilmes, J. A. On fast approximate submodular minimization. In *NIPS*, 2011.

Jegelka, S., Bach, F., and Sra, S. Reflection methods for user-friendly submodular optimization. In *Proceedings of the 26th International Conference on Neural Information Processing Systems*, NIPS'13, pp. 1313–1321, Red Hook, NY, USA, 2013. Curran Associates Inc.

Jerrum, M. and Sinclair, A. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing*, 1993.

Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., and Karbasi, A. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proceedings of the 36th International Conference on Machine Learning*, pp. 3311–3320. PMLR, 2019. URL `http://proceedings.mlr.press/v97/kazemi19a.html`.

Kempe, D., Kleinberg, J., and Tardos, E. Maximizing the spread of influence through a social network. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pp. 137–146, New York, NY, USA, 2003. Association for Computing Machinery. ISBN 1581137370. doi: 10.1145/956750.956769. URL `https://doi.org/10.1145/956750.956769`.

Khuller, S., Moss, A., and Naor, J. S. The budgeted maximum coverage problem. *Information Processing Letters*, 70(1):39–45, 1999. ISSN 0020-0190. doi: https://doi.org/10.1016/S0020-0190(99)00031-9. URL *https://www.sciencedirect.com/science/article/pii/S0020019099000319*.

Krause, A. and Cevher, V. Submodular dictionary selection for sparse representation. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pp. 567–574, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Krause, A. and Golovin, D. *Submodular Function Maximization*. Cambridge University Press, 2014. doi: 10.1017/CBO9781139177801.004.

Krause, A. and Guestrin, C. Near-optimal nonmyopic value of information in graphical models. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pp. 324–331, Arlington, Virginia, USA, 2005. AUAI Press. ISBN 0974903914.

Krause, A. and Guestrin, C. Near-optimal observation selection using submodular functions. In *Proceedings of the 22nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, pp. 1650–1654, 2007. ISBN 9781577353232.

Krause, A., Guestrin, C., Gupta, A., and Kleinberg, J. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th international conference on Information processing in sensor networks*, pp. 2–10, 2006.

Kuhnle, A., Smith, J. D., Crawford, V., and Thai, M. Fast maximization of non-submodular, monotonic functions on the integer lattice. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 2786–2795. PMLR, 10–15 Jul 2018. URL *https://proceedings.mlr.press/v80/kuhnle18a.html*.

Kulesza, A. and Taskar, B. Determinantal point processes for machine learning. *Foundations and Trends in Machine Learning*, 2012.

Kulik, A., Shachnai, H., and Tamir, T. Maximizing submodular set functions subject to multiple linear constraints. In *SODA*, 2009.

Kunegis, J. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pp. 1343–1350, 2013. URL *http://dl.acm.org/citation.cfm?id=2488173*.

Lai, L., Ni, Q., Lu, C., Huang, C., and Wu, W. Monotone submodular maximization over the bounded integer lattice with cardinality constraints. *Discrete Mathematics, Algorithms and Applications*, 11(06):1950075, December 2019. doi: 10.1142/s1793830919500757. URL *https://doi.org/10.1142/s1793830919500757*.

Lee, J., Mirrokni, V. S., Nagarajan, V., and Sviridenko, M. Non-monotone submodular maximization under matroid and knapsack constraints. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, pp. 323–332, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585062. doi: 10.1145/1536414.1536459. URL *https://doi.org/10.1145/1536414.1536459*.

Levin, D. *Markov chains and mixing times.* American Mathematical Society, Providence, R.I, 2009. ISBN 9780821847398.

Levitin, E. and Polyak, B. Constrained minimization methods. *USSR Computational Mathematics and Mathematical Physics*, 6(5):1–50, 1966. ISSN 0041-5553. doi: https://doi.org/10.1016/0041-5553(66)90114-5. URL *https://www.sciencedirect.com/science/article/pii/0041555366901145*.

Li, C., Jegelka, S., and Sra, S. Fast mixing markov chains for strongly Rayleigh measures, DPPs, and constrained sampling. In *Neural Information Processing Systems (NIPS)*, 2016.

Liggett, T. M. Negative correlations and particle systems. *Markov Processes and Related Fields*, 2002.

Lin, H. and Bilmes, J. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pp. 510–520, USA, 2011. Association for Computational Linguistics. ISBN 9781932432879.

Lovász, L. Submodular functions and convexity. In *Mathematical Programming - The State of the Art*, pp. 235–257. Springer Berlin Heidelberg, 1983.

Lyons, R. Determinantal probability measures. *Publications mathématiques de l'IHÉS*, 2003.

Maehara, T., Nakashima, S., and Yamaguchi, Y. Multiple knapsack-constrained monotone dr-submodular maximization on distributive lattice. *Mathematical Programming*, pp. 1–35, 2021.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. doi: 10.1063/1.1699114. URL *https://doi.org/10.1063/1.1699114*.

Minoux, M. Accelerated greedy algorithms for maximizing submodular set functions. In Stoer, J. (ed.), *Optimization Techniques*, pp. 234–243. Springer Berlin Heidelberg, 1978. ISBN 978-3-540-35890-9. doi: 10.1007/bfb0006528. URL *https://doi.org/10.1007/bfb0006528*.

Mirzasoleiman, B., Karbasi, A., Sarkar, R., and Krause, A. Distributed submodular maximization: Identifying representative elements in massive data. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pp. 2049–2057, Red Hook, NY, USA, 2013. Curran Associates Inc.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. Lazier than lazy greedy. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, AAAI'15, pp. 1812–1818. AAAI Press, 2015. ISBN 0262511290.

Mirzasoleiman, B., Badanidiyuru, A., and Karbasi, A. Fast constrained submodular maximization: Personalized data summarization. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pp. 1358–1366. JMLR.org, 2016.

Mokhtari, A., Hassani, H., and Karbasi, A. Decentralized submodular maximization: Bridging discrete and continuous settings. In *ICML*, 2018.

Mokhtari, A., Hassani, H., and Karbasi, A. Stochastic conditional gradient methods: From convex minimization to submodular maximization. *Journal of machine learning research*, 2020.

Motwani, R. *Randomized algorithms.* Cambridge University Press, Cambridge New York, 1995. ISBN 978-0521474658.

Murota, K. Discrete convex analysis. *Mathematical Programming*, 83(1-3):313–371, January 1998. doi: 10.1007/bf02680565. URL *https://doi.org/10.1007/bf02680565*.

Murota, K. *Discrete convex analysis.* Society for Industrial and Applied Mathematics, Philadelphia, 2003. ISBN 978-0898715408.

Narasimhan, M. and Bilmes, J. A submodular-supermodular procedure with applications to discriminative structure learning. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, UAI'05, pp. 404–412, Arlington, Virginia, USA, 2005. AUAI Press. ISBN 0974903914.

Narayanan, H. *Submodular functions and electrical networks.* Number 54 in Annals of discrete mathematics. Elsevier, 1997. ISBN 9780444825230.

Nemhauser, G. and Wolsey, L. *Integer and Combinatorial Optimization.* John Wiley & Sons, Inc., June 1988. doi: 10.1002/9781118627372. URL *https://doi.org/10.1002/9781118627372*.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions. *Mathematical Programming*, 14(1):265–294, December 1978. doi: 10.1007/bf01588971. URL *https://doi.org/10.1007/bf01588971*.

Niazadeh, R., Roughgarden, T., and Wang, J. R. Optimal algorithms for continuous non-monotone submodular and dr-submodular maximization. *J. Mach. Learn. Res.*, 21:125–1, 2020.

Orlin, J. B. A faster strongly polynomial time algorithm for submodular function minimization. *Mathematical Programming*, 118(2):237–251, November 2007. doi: 10.1007/s10107-007-0189-2. URL *https://doi.org/10.1007/s10107-007-0189-2*.

pandas development team, T. pandas-dev/pandas: Pandas, February 2020. URL *https://doi.org/10.5281/zenodo.3509134*.

Pemantle, R. Towards a theory of negative dependence. *Journal of Mathematical Physics*, 2000.

Potts, R. B. Some generalized order-disorder transformations. *Mathematical Proceedings of the Cambridge Philosophical Society*, 1952.

Qian, C., Zhang, Y., Tang, K., and Yao, X. On multiset selection with size constraints. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Roberts, G. O. and Tweedie, R. L. Exponential convergence of Langevin distributions and their discrete approximations. *Bernoulli*, 2(4):341 – 363, 1996. doi: bj/1178291835. URL *https://doi.org/*.

Sahin, A., Bian, Y., Buhmann, J., and Krause, A. From sets to multisets: Provable variational inference for probabilistic integer submodular models. In *International Conference on Machine Learning*, pp. 8388–8397. PMLR, 2020a.

Sahin, A., Buhmann, J., and Krause, A. Constrained maximization of lattice submodular functions. In *ICML 2020 workshop on Negative Dependence and Submodularity for ML, Vienna, Austria, PMLR 119, 2020.*, 2020b.

Schrijver, A. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000. ISSN 0095-8956. doi: https://doi.org/10.1006/jctb.2000.1989. URL *https://www.sciencedirect.com/science/article/pii/S0095895600919890*.

Schrijver, A. *Combinatorial optimization: polyhedra and efficiency*, volume B. Springer, Berlin New York, 2003. ISBN 978-3-540-44389-6.

Shi, M., Yang, Z., Kim, D., and Wang, W. Non-monotone submodular function maximization under k-system constraint. *Journal of Combinatorial Optimization*, 41(1):128–142, January 2021. doi: 10.1007/s10878-020-00672-3. URL *https://doi.org/10.1007/s10878-020-00672-3*.

Soma, T. and Yoshida, Y. A generalization of submodular cover via the diminishing return property on the integer lattice. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL *https://proceedings.neurips.cc/paper/2015/file/7bcdf75ad237b8e02e301f4091fb6bc8-Paper.pdf*.

Soma, T. and Yoshida, Y. Maximizing monotone submodular functions over the integer lattice. *Mathematical Programming*, 172(1–2):539–563, November 2018. ISSN 0025-5610.

Soma, T., Kakimura, N., Inaba, K., and Kawarabayashi, K.-i. Optimal budget allocation: Theoretical guarantee and efficient algorithm. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pp. I–351–I–359. JMLR.org, 2014. doi: 10.5555/3044805.3044846.

Stobbe, P. and Krause, A. Efficient minimization of decomposable submodular functions. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, pp. 2208–2216, Red Hook, NY, USA, 2010. Curran Associates Inc.

Streeter, M. and Golovin, D. An online algorithm for maximizing submodular functions. In *Proceedings of the 21st International Conference on Neural Information Processing Systems*, NIPS'08, pp. 1577–1584, Red Hook, NY, USA, 2008. Curran Associates Inc. ISBN 9781605609492.

Testa, A., Farina, F., and Notarstefano, G. Distributed submodular minimization via block-wise updates and communications. *IFAC-PapersOnLine*, 53(2):2678–2683, 2020. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2020.12.386. URL *https://www.sciencedirect.com/science/article/pii/S2405896320306728*. 21st IFAC World Congress.

Topkis, D. M. Minimizing a submodular function on a lattice. *Operations Research*, 26(2):305–321, 1978. URL *https://EconPapers.repec.org/RePEc:inm:oropre:v:26:y:1978:i:2:p:305-321*.

Vondrák, J. Submodularity in combinatorial optimization. In *Ph.D. Thesis, Charles University*, 2007.

Vondrak, J. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pp. 67–74, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605580470. doi: 10.1145/1374376.1374389. URL *https://doi.org/10.1145/1374376.1374389*.

Vondrák, J. Symmetry and approximability of submodular maximization problems. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*. IEEE, October 2009.

Wagner, D. G. Negatively correlated random variables and mason's conjecture for independent sets in matroids. *Annals of Combinatorics*, 2008.

Wainwright, M. J. and Jordan, M. I. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 2008.

Wang, C., Komodakis, N., and Paragios, N. Markov random field modeling, inference & learning in computer vision & image understanding: A survey. *Computer Vision and Image Understanding*, 2013.

Ward, J. and Živný, S. Maximizing k-submodular functions and beyond. *ACM Trans. Algorithms*, 12(4), August 2016. ISSN 1549-6325. doi: 10.1145/2850419. URL *https://doi.org/10.1145/2850419*.

Waskom, M. L. seaborn: statistical data visualization. *Journal of Open Source Software*, 6(60):3021, 2021. doi: 10.21105/joss.03021. URL *https://doi.org/10.21105/joss.03021*.

Whitney, H. On the abstract properties of linear dependence. *American Journal of Mathematics*, 57(3):509–533, 1935. ISSN 00029327, 10806377.

Williamson, D. P. and Shmoys, D. B. *The Design of Approximation Algorithms*. Cambridge University Press, USA, 1st edition, 2011. ISBN 0521195276.

Wolfe, P. Finding the nearest point in a polytope. *Mathematical Programming*, 11(1):128–149, December 1976. doi: 10.1007/bf01580381. URL *https://doi.org/10.1007/bf01580381*.

Xu, X., Lu, Y., Huang, S., Xiao, Y., and Wang, W. Incremental sensor placement optimization on water network. In Blockeel, H., Kersting, K., Nijssen, S., and Železný, F. (eds.), *Machine Learning and Knowledge Discovery in Databases*, pp. 467–482, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40994-3.

Yadan, O. Hydra - a framework for elegantly configuring complex applications. Github, 2019. URL *https://github.com/facebookresearch/hydra*.

Yu, J. *Minimisation du risque empirique avec des fonctions de perte nonmodulaires*. Theses, Université Paris-Saclay, March 2017. URL *https://tel.archives-ouvertes.fr/tel-01514162*.

Yudin, D. and Nemirovskii, A. Data complexity and effective methods for solution of convex extremal problems. *Ékon. Mat. Metody*, 12(2):357–369, 1976.