Online, Submodular, and Polynomial Optimization with Discrete Structures

Shinsaku Sakaue

Abstract

This dissertation is devoted to online, submodular, and polynomial optimization problems, all of which have been important research subjects in terms of both theory and practice. With the help of tools and perspectives provided by discrete structures, which represent certain relationship between discrete objects, we make various contributions to those problems: we develop practically effective algorithms, prove new theoretical guarantee of existing algorithms, and reveal new theoretical properties.

First, we address the online combinatorial optimization: more precisely, the bandit combinatorial optimization. This optimization problem models various scenarios of sequential decision making: we adaptively take actions in a time-varying environment to maximize our utility. In practical settings, each action we can take is represented as a combination of some elements, and the number of such actions can be too huge to be handled with existing algorithms. We address such problems by leveraging data structures called zero-suppressed binary decision diagrams (ZDDs), which enable us to efficiently deal with such huge sets of combinatorial objects. We prove theoretical guarantees of our algorithm and experimentally validate it with bandit combinatorial optimization instances defined on real-world networks.

We then turn to submodular function maximization problems. Maximization of submodular functions appear in various situations, e.g., machine learning, marketing, and facility location, since many utility functions used in such areas have the submodularity. It is know that efficient greedy-style algorithms are effective for various submodular function maximization problems, but they currently have limitations: those algorithms require constraints of the optimization problems to have some desirable properties, and their theoretical guarantees are sometimes unsatisfying. We develop two algorithms to go beyond these limitations. The first is a ZDD-based algorithm that can deal with various combinatorial constraints with complicated structures, and

the second is an algorithm that can achieve better theoretical guarantees than the greedy-style algorithms. Unfortunately, unlike the greedy-style algorithms, both of the two algorithms can incur prohibitively expensive computation cost in general. However, as we will confirm via various experiments, both algorithms are empirically efficient enough for practical use. We then address maximization of k-submodular functions, which can be seen as generalization of submodular functions, and prove a new theoretical guarantee of the greedy algorithm for the case where constraints are specified with matroids, which form a wide class of discrete structures.

We finally consider binary polynomial optimization, or minimization of polynomial functions with binary variables, which can model various combinatorial optimization problems. The problem has complicated underlying structures, and so many previous studies have considered reformulating the problem as a convex optimization problem, which is typically easier to deal with. At the price of tractable representation, however, the size of the resulting convex optimization problem can be exponential in the number of variables in general. Therefore, to reveal the minimum size of the convex optimization problem required for reformulating the original binary polynomial optimization problem has been an important research subject. We reveal tight upper bounds on the size of the convex optimization problem by utilizing the underlying discrete structures of the binary polynomial optimization problem.

Acknowledgements

First of all, I would like to express my deepest gratitude to my supervisor, Shin-ichi Minato. He always spares time for discussing with me, as well as members in our laboratory, despite his busy schedule, and he has given me various insightful comments and suggestions. My Ph.D. student days have been meaningful and delightful thanks to his invaluable and generous support.

I would like to sincerely thank my collaborators: Kaito Fujii, Tsutomu Hirao, Masakazu Ishihata, Naoki Ito, Sunyoung Kim, Naoki Marumo, Kengo Nakamura, Masaaki Nagata, Masaaki Nishino, Akiko Takeda, and Norihito Yasuda. Some of the joint works with them form building blocks of this dissertation. Especially, I am grateful to Masakazu Ishihata, Masaaki Nishino, and Norihito Yasuda for their continuing support. I also would like to express my special thanks to Akiko Takeda, who is my former supervisor. She kindly encouraged me to enter the world of academic research.

I am also grateful to researchers who gave me valuable and interesting comments on our works: Satoru Iwata, Masakazu Kojima, Takanori Maehara, Yuji Nakatsukasa, Tasuku Soma, and Yutaro Yamaguchi.

My appreciation also goes to current and former members of our laboratory. Especially, I am grateful to Yu Nakahata for valuable and exciting discussion.

I would like to express my sincere thanks to my family, relatives, and friends for their persistent support and considerable encouragement.

Finally, I thank JSPS KAKENHI Grant Numbers 15H05711 and 16K16011 for supporting the works in Chapters 3 and 6. I am also grateful to JST Minato Discrete Structure Manipulation System Project and JSPS KAKENHI(S) Discrete Structure Manipulation System Project for constantly holding invaluable research events.

List of Publications

Publications Included in this Dissertation

This dissertation includes contents of the following six publications.

Refereed Journal Articles

1. Shinsaku Sakaue.
   On Maximizing a Monotone k-submodular Function Subject to a Matroid Constraint.
   Discrete Optimization, 23, pp. 105–113, 2017.

2. Shinsaku Sakaue, Akiko Takeda, Sunyoung Kim, and Naoki Ito.
   Exact Semidefinite Programming Relaxations with Truncated Moment Matrix for Binary Polynomial Optimization Problems.
   SIAM Journal on Optimization, 27(1), pp. 565–582, 2017.

Refereed Conference Proceedings

1. Shinsaku Sakaue, Masaaki Nishino, and Norihito Yasuda.
   Submodular Function Maximization over Graphs via Zero-suppressed Binary Decision Diagrams.
   In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-2018)*, pp. 1422–1430, 2018.

2. Shinsaku Sakaue and Masakazu Ishihata.
   Accelerated Best-first Search with Upper-bound Computation for Submodular Function Maximization.
   In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI-2018)*, pp. 1413–1421, 2018.

3. Shinsaku Sakaue, Masakazu Ishihata, and Shin-ichi Minato.
   Efficient Bandit Combinatorial Optimization Algorithm with Zero-suppressed Binary Decision Diagrams.
   In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS-2018)*, pp. 585–594, 2018.

4. Shinsaku Sakaue.
   Greedy and IHT Algorithms for Non-convex Optimization with Monotone Costs of Non-zeros.
   In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS-2019)*, pp. 206–215, 2019.

Other Refereed Publications

The author also published the following papers, which are not included in this dissertation for consistency of the contents. The last two publications will be contained in the dissertations of the first authors.

1. Shinsaku Sakaue, Tsutomu Hirao, Masaaki Nishino, and Masaaki Nagata.
   Provable Fast Greedy Compressive Summarization with Any Monotone Submodular Function.
   In *Proceedings of the 16th Annual Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-2018)*, pp. 1737–1746, 2018.

2. Kaito Fujii and Shinsaku Sakaue.
   Beyond Adaptive Submodularity: Approximation Guarantees of Greedy Policy with Adaptive Submodularity Ratio.
   In *Proceedings of the 36th International Conference on Machine Learning (ICML-2019)*, pp. 2042–2051, 2019.

3. Kengo Nakamura, Shinsaku Sakaue, and Norihito Yasuda.
   Practical Frank–Wolfe Method with Decision Diagrams for Computing Wardrop Equilibrium of Combinatorial Congestion Games.
   In *Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI-2020)*, To appear (page numbers are not assigned yet), 2020.

4. Shinsaku Sakaue.
   Guarantees of Stochastic Greedy Algorithms for Non-monotone Submodular Maximization with Cardinality Constraint.
   In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS-2020)*, To appear (page numbers are not assigned yet), 2020.

5. Shinsaku Sakaue.
   On Maximization of Weakly Modular Functions: Guarantees of Multi-stage Algorithms, Tractability, and Hardness.
   In *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics (AISTATS-2020)*, To appear (page numbers are not assigned yet), 2020.

Unrefereed Preprints

The author also addressed the following works, which are currently unrefereed preprints and not included in this dissertation.

1. Shinsaku Sakaue.
   Using Multiparameter Eigenvalues for Solving Quadratic Programming with Quadratic Equality Constraints.
   Mathematical Engineering Technical Reports, METR 2016-02, 2016.

2. Shinsaku Sakaue and Naoki Marumo.
   Best-first Search Algorithm for Non-convex Sparse Minimization.
   arXiv preprints, arXiv:1910.01296, 2019.

Contents

Chapter 1

Introduction

This dissertation considers a wide variety of optimization problems. Basically, optimization problems model situations where we want to make the best choice from some set of available choices. The measure that quantifies the goodness of our choice is called the *objective function*, and, typically, each available choice must satisfy certain *constraints*. Formally, optimization problems can be written as follows:

$$\underset{X \in \mathcal{X}}{\text{maximize}} \ (\text{or } \underset{X \in \mathcal{X}}{\text{minimize}}) \quad f(X) \qquad \text{subject to} \quad X \in \mathcal{C}, \tag{1.1}$$

where $\mathcal{X}$ is some space, $f : \mathcal{X} \to \mathbb{R}$ is an objective function, and $\mathcal{C} \subseteq \mathcal{X}$ is a *feasible region*, which corresponds to the set of available choices satisfying constraints. We sometimes refer to the part, $X \in \mathcal{C}$, as the constraint and each X ($\in \mathcal{C}$) as a (feasible) *solution*.

The degree of difficulty of solving optimization problems (1.1) depends on the structures f and $\mathcal{C}$. If both f and $\mathcal{C}$ have simple structures (e.g., specified with linear functions), problem (1.1) is often easy to solve. In many applications, however, f and/or $\mathcal{C}$ have complicated structures, and such problems are often difficult to solve with existing general purpose solvers. Moreover, it is believed to be almost impossible to develop an omnipotent algorithm that can efficiently solve problem (1.1) with any structures of f and $\mathcal{C}$. These facts have been motivating many researchers to develop various optimization algorithms by leveraging problem-specific structures.

Throughout this dissertation, we consider addressing various optimization problems by utilizing discrete structures, which, roughly speaking, are structures of some objects defined on discrete domains. There are various ways

to use discrete structures for addressing optimization problems: some of our proposals consider representing feasible region $\mathcal{C}$ with discrete structures to develop efficient optimization algorithms, and others use discrete structures to understand the nature of optimization problems. Consequently, our works become problem-dependent. Therefore, below we describe the background and our contribution for each optimization problem considered in this dissertation separately: online (bandit) combinatorial optimization, submodular function maximization, and binary polynomial optimization.

1.1 Bandit Combinatorial Optimization

Bandit combinatorial optimization is an online decision making problem where a player choose a combinatorial object in each round. This problem generalizes the well-known multi-armed bandit problem [Robbins, 1952], where a player choose an arm in each round. Bandit combinatorial optimization has first been studied in [Awerbuch and Kleinberg, 2004; McMahan and Blum, 2004], and both of them obtained first but suboptimal theoretical guarantees. After that, many researchers have studied the problem and improved the theoretical guarantees [Dani *et al.*, 2008; Cesa-Bianchi and Lugosi, 2012; Bubeck *et al.*, 2012]. Relative to other online combinatorial optimization problems, bandit combinatorial optimization is quite difficult since the feedback a player can obtain in each round is very little. An extensive survey of online combinatorial optimization is provided in [Audibert *et al.*, 2014].

Since bandit combinatorial optimization has many applications such as network routing [Awerbuch and Kleinberg, 2004; György *et al.*, 2007] and multi-location communication [Imase and Waxman, 1991], it is important to develop practical bandit combinatorial optimization algorithms. Unfortunately, however, existing algorithms, including those listed in the above paragraph, are inefficient since they basically examine all combinatorial objects in each iteration to make a decision. Given a ground set, the number of combinatorial objects is generally exponential in the ground-set size, and thus existing algorithms do not scale to large instances that appear in real-world scenarios.

In Chapter 3, we propose an efficient bandit combinatorial optimization algorithm. Our main idea is to represent all the combinatorial objects compactly by using a certain discrete structure, called zero-suppressed binary decision diagram (ZDD). Our algorithm enjoys theoretical guarantees that

are as good as those of existing ones, and it works efficiently for bandit combinatorial optimization instances defined on networks, which include the applications mentioned above. We confirm the efficiency of our algorithm via experiments.

1.2 Constrained Monotone Submodular Function Maximization

Submodular functions are set functions defined on a finite ground set and have the diminishing return property. Since various practical set functions conform to the law of diminishing marginal utility, submodular function maximization appears in many realistic scenarios: sensor placement [Krause *et al.*, 2008a,b], feature selection [Thoma *et al.*, 2009], document summarization [Lin and Bilmes, 2010], and influence maximization [Kempe *et al.*, 2003].

The most well-know result is the $(1 - 1/e)$-approximation guarantee of the greedy algorithm for the case of monotone submodular maximization with a cardinality constraint [Nemhauser *et al.*, 1978]. Sviridenko [2004] proved that the same guarantee is possible in polynomial time for the case with knapsack constraints. Calinescu *et al.* [2011] proposed a continuous-version of the greedy algorithm, which achieves the $(1 - 1/e)$-approximation guarantee for the case of matroid constraints. It is know that no polynomial time algorithms can improve the above $(1 - 1/e)$-approximation guarantees in general [Nemhauser and Wolsey, 1978; Feige, 1998]. However, if a submodular function has some desirable properties, we can obtain better guarantees; the curvature of submodular functions is often used in this context. If submodular functions have small curvature values, better approximation guarantees can be achieved for the case of the cardinality, matroid, and knapsack constraints [Conforti and Cornuéjols, 1984; Sviridenko *et al.*, 2015; Yoshida, 2019].

Since the submodularity provides various benefits both in theory and in practice as above, extending the notion to more general settings is an interesting and valuable research direction. The k-submodular function [Huber and Kolmogorov, 2012] is one such extension, whose input is given by k disjoint subsets instead of a single subset. For example, it appears when we consider placing k kinds of sensors. Most previous studies on k-submodular function maximization are devoted to unconstrained and cardinality-constrained cases: Ward and Živný [2016] provided a constant-factor approximation algorithm

for unconstrained non-negative k-submodular function maximization, and Iwata *et al.* [2016] improved this result. They also provided an approximation algorithm for non-negative monotone k-submodular maximization and proved the tightness of their result. Ohsaka and Yoshida [2015] proved constant-factor approximation guarantees for non-negative monotone k-submodular maximization with some cardinality constraints.

In Chapters 5, 6, and 7, we make the following three contributions to constrained monotone (k-)submodular function maximization.

Monotone Submodular Function Maximization over Graphs

While submodular function maximization with aforementioned relatively simple constraints have been studied extensively, many practical problems have more complicated constraints. For example, when collecting information by using a mobile robot equipped sensors, we need to design a trajectory of the robot so that it can collect as much information as possible. Such a problem can be formulated as submodular function maximization with constraints defined on graphs. Such constraints are often very complicated, and so the resulting problems are typically hard to deal with. In Chapter 5, we develop a ZDD-based approximation algorithm for such problems; while its computation complexity is generally not polynomial in the size of instances, it is efficient enough to be used for practical moderate-size instances. We provide a curvature-based approximation guarantee, and we confirm the effectiveness of our algorithm via experiments. We also discuss greedy maximization of set functions that has approximate submodularity over some graph-based constraints.

Monotone Submodular Function Maximization under Knapsack Constraint

As mentioned above, regarding monotone submodular function maximization under a knapsack constraint, which for short we call submodular knapsack problem, no polynomial time algorithms can improve the $(1 - 1/e)$-approximation guarantee in general. In some cases, however, we need approximation guarantees that are better than $1 - 1/e \approx 0.63$, particularly when making vital decisions on the basis of computed solutions. In Chapter 6, by leveraging a search strategy described on some discrete structures, we develop an algorithm for submodular knapsack problems that can achieve

any approximation ratio $\alpha \in [0, 1]$, where α is an input of the algorithm that controls the trade-off between running times and approximation guarantees. Our algorithm generally requires computation cost that is exponential in the instance size; this is natural due to the hardness of the original problem. Fortunately, however, our algorithm empirically runs fast enough to be used for moderate-size instances as confirmed via experiments.

k-submodular Function Maximization under Matroid Constraint

The k-submodular function maximization has been studied extensively for the unconstrained and cardinality-constrained cases, but few previous studies have addressed the problems with more general constraints. In particular, while submodular function maximization with a matroid constraint has been well studied as mentioned above, no approximation algorithms for k-submodular function maximization with a matroid constraint have been studied. In Chapter 7, we show that the greedy algorithm can achieve a $1/2$-approximation guarantee for non-negative monotone k-submodular function maximization with a matroid constraint.

1.3 Binary Polynomial Optimization

We finally study the binary polynomial optimization problem (POP), which is a problem of minimizing d-degree polynomial functions with n binary variables. Since polynomial functions can represent various practical objective functions, binary POPs appear in many scenarios, e.g., max-cut and maximum independent set problems. The binary POP is quite hard due to its non-convexity, and so how to reformulate (or relax) it as a convex optimization problem has been extensively studied. One of the most common approach is to use semidefinite programming (SDP), a convex optimization problem with a matrix variable. For the case of max-cut, a special case of binary POPs with $d = 2$, Goemans and Williamson [1995] proposed a 0.87856-approximation algorithm by using SDP relaxations.

When it comes to general binary POPs, the Lasserre's hierarchy [Lasserre, 2001b] provides an well-established framework for obtaining SDP relaxations. The size of SDPs obtained with the framework is specified by the so-called relaxation order ω; an SDP with relaxation order ω is called the ωth SDP relaxation. SDPs with larger ω provide tighter relaxations, but their sizes

increase with ω. Therefore, the minimum value of ω required for obtaining an exact reformulation of the original binary POP has been an attractive research subject. For the case of $d = 2$, Laurent [2003] conjectured that the $\lceil n/2 \rceil$th SDP relaxation always provides exact reformulations of binary POPs, and this was proved by Fawzi *et al.* [2015]. For the case of $d > 2$, Lasserre [2001a] proved that the nth SDP relaxation provides an exact reformulation, i.e., $\omega = n$. However, whether we can reduce $\omega = n$ without spoiling the exactness has been an open problem.

In Chapter 8, we show that the $\lceil (n + d - 1)/2 \rceil$th SDP relaxation provides an exact reformulation of general binary POPs, and it can be reduced to $\lceil (n + d - 2)/2 \rceil$ if objective functions involve only even-degree monomials. Our proof extends the technique of [Fawzi *et al.*, 2015], developed for the case of $d = 2$, to the case of general d. To this end, we utilize underlying discrete structures of binary POPs.

1.4 Notation and Definitions

We denote the set of real and integer numbers by $\mathbb{R}$ and $\mathbb{Z}$, respectively. We use $\mathbb{R}_{\geq 0}$ ($\mathbb{Z}_{\geq 0}$) and $\mathbb{R}_{>0}$ ($\mathbb{Z}_{>0}$) to denote the sets of positive and non-negative real (integer) numbers, respectively. Given any $m \in \mathbb{Z}_{>0}$, we define $[m] := \{1, \ldots, m\}$. Given a finite set F and set of numbers $\mathbb{X}$, e.g., $\mathbb{R}$, $\mathbb{Z}$, and $\{0, 1\}$, we let $\mathbb{X}^F$ denote the space of vectors (or matrices) whose entries belong to $\mathbb{X}$ and are indexed by the elements in F.

Unless otherwise specified, we employ the following notation regarding vectors and matrices. We denote vectors and matrices by bold and capital italic letters, respectively, e.g., $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ and $A, B \in \mathbb{R}^{m \times n}$. Each entry of vectors and matrices are denoted by italic letters, e.g., $a_i \in \mathbb{R}$ and $A_{i,j} \in \mathbb{R}$. The zero and all-one vectors are denoted by $\mathbf{0}$ and $\mathbf{1}$, respectively. Given any vector $\boldsymbol{x} = (x_1, \ldots, x_n)^\top \in \mathbb{R}^n$, we let $\|\boldsymbol{x}\|_p$ denote the ℓ_p-norm defined by $\left(\sum_{i \in n} |x_i^p|\right)^{1/p}$. We sometimes denote the ℓ_2-norm simply by $\|\cdot\|$. Given any square matrix $X \in \mathbb{R}^{n \times n}$, we denote its trace by $\mathrm{tr}(X) \coloneqq \sum_{i=1}^n X_{i,i}$.

Let V be a finite ground. We use 2^V to denote the power set of V. The cardinality of any $X \subseteq V$ is denoted by $|X|$. Given any $X \subseteq V$ and $v \in V$, we sometimes abuse the notation and denote $X \cup \{v\}$ by $X \cup v$ and $X \backslash \{v\}$ by $X \backslash v$. Let $f : 2^V \to \mathbb{R}$ be any set function. For any $X, Y \subseteq V$, we define $f(Y \mid X) \coloneqq f(X \cup Y) - f(X)$. We sometimes abuse notation as above and

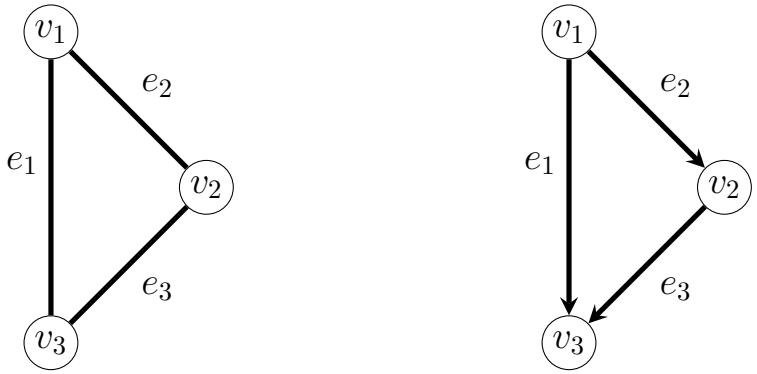


Figure 1.1: Examples of undirected (left) and directed (right) graphs. The directed graph is acyclic, but it is not a tree.

denote $f(\{v\} \mid X)$ by $f(v \mid X)$.

Finally, we introduce notation and definitions related to graphs, which are prevalent discrete structures and will appear repetitively in this dissertation. An undirected graph, $G = (V, E)$, is specified with a finite set of vertices, V, and set of edges $E \subseteq V \times V$; we often refer to an undirected graph simply as graph. A sequence of edges such that $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$ is called a path connecting $v_1 \in V$ and $v_k \in V$. A path that has no repeated vertex is called a simple path, and a path with no repeated edge is called a walk; we take any path appears in what follows to be simple unless otherwise stated. Graph G is said to be connected if any two vertices in G are connected by at least one path; we suppose G to be connected unless otherwise stated. Graph G is called a tree if any pair of vertices are connected by only one path. A cycle in G is a sequence of edges such that $(v_1, v_2), (v_2, v_3), \ldots, (v_k, v_1)$, and graph G with no cycle is said to be acyclic. An undirected graph, G, is a tree iff G is acyclic. If each edge $e = (u, v) \in E$ in G has a direction, we say G is a directed graph, and we sometimes call an edge of a directed graph an arc. Note that a directed graph is not always a tree even if it is acyclic (see, Figure 1.1).

Graph structures are often used to describe constraints of optimization problems. For example, if we are to find the shortest route connecting certain two locations on a road network, we represent the network with a graph and we solve a minimization problem whose constraints force an output solution to be a path connecting the two vertices. Moreover, the decision diagrams,

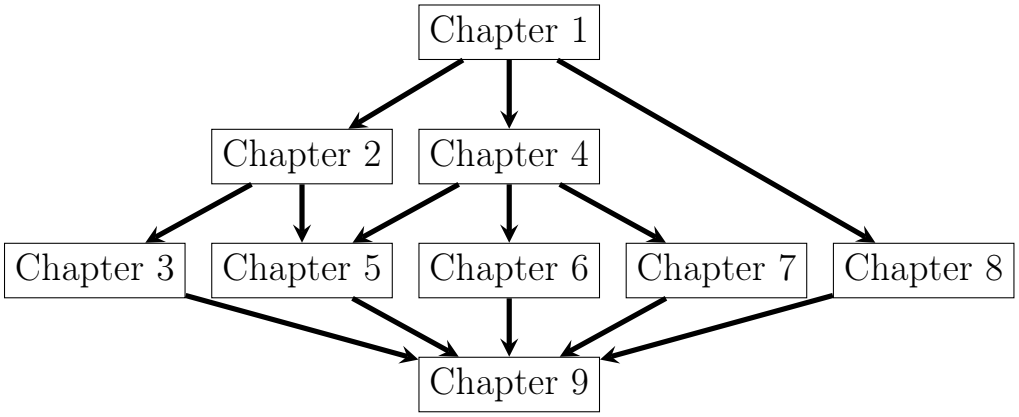


Figure 1.2: Dependence structure of chapters

which we will explain in Chapter 2 and use in Chapters 3 and 5, can also be seen as directed acyclic graphs (DAGs). In Chapter 6, we use directed tree structures to develop an optimization algorithm. In Chapter 8, we study polynomial optimization problems by representing their underlying structures with graphs. We sometimes call graphs as networks, particularly when they should be distinguished from decision diagrams.

1.5 Organization

The rest of this dissertation is organized as follows. Chapter 2 presets background to the decision diagrams, we will use in Chapters 3 and 5. Chapter 3 develops an ZDD-based algorithm for the bandit combinatorial optimization. Chapters 4, 5, 6, and 7 are devoted to submodular function maximization. Specifically, Chapter 4 presents background to submodular function maximization, Chapter 5 presents the ZDD-based approach to monotone submodular function maximization over graphs, Chapter 6 presents the algorithm that can achieve arbitrary approximation ratios for submodular knapsack problems, and Chapter 7 proves the 1/2-approximation guarantee of the greedy algorithm for monotone k-submodular function maximization under a matroid constraint. Chapter 8 considers binary POPs and proves the exactness of the $\lceil (n+d-1)/2 \rceil$th (and $\lceil (n+d-2)/2 \rceil$th) SDP relaxations. Finally, Chapter 9 concludes this dissertation. Figure 1.2 illustrates the dependence structure of the chapters.

Chapter 2

Background to Decision Diagrams

This chapter presents an overview on decision diagrams (DDs): data structures that we will use repetitively in this dissertation. The readers who are familiar with this topic can skip this chapter.

In many realistic scenarios, optimization problems have combinatorial constraints, e.g., that of the shortest path problem mentioned in Chapter 1. If combinatorial constraints enjoy desirable properties, e.g., so-called hereditary and augmentation properties, we can often develop efficient optimization algorithms. Unfortunately, however, constraints of many practical optimization problems do not have such nice properties. In such cases, to store all feasible solutions can be a simple but powerful approach. The following data structures are basically designed to (compactly) store all combinatorial objects satisfying certain constraints.

2.1 Binary Decision Trees (BDTs)

The binary decision tree (BDT) is a naive data structure that can store all feasible solutions. Let E be a finite ground set and $\mathcal{C} \subseteq 2^E$ be a set of all feasible solutions. For example, in the case of the shortest path problem, E is an edge set of a given graph and $\mathcal{C}$ is a set of all paths that connect two specified vertices s and t. A BDT that stores all $X \in \mathcal{C}$ is constructed as follows. We fix a total order of all elements in E; for simplicity, we let $E = \{1, \dots, |E|\}$. In this order, we repeatedly consider two cases: $e \in E$ is chosen or not. Consequently, we have $2^{|E|}$ outcomes, $\{X\}_{X \in 2^E}$. BDT $\mathtt{T}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ is a directed tree that represents all the outcomes. Let $\mathtt{r} \in \mathtt{N}$ be

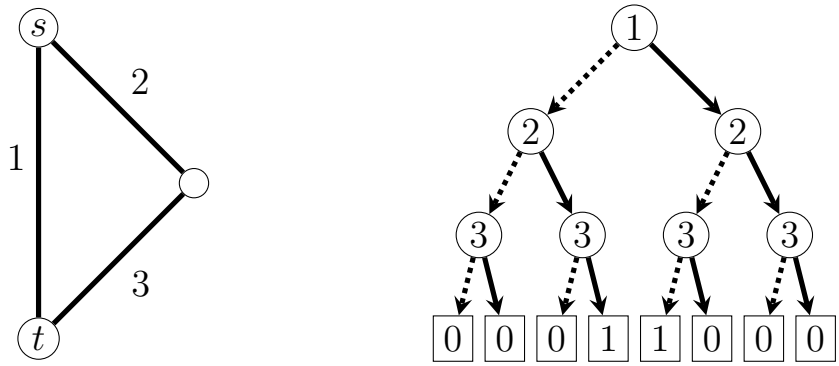


Figure 2.1: Examples of an undirected graph $G = (V, E)$ with two specified vertices, s and t, and BDT $\mathtt{T}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ that stores all s–t paths. Each node $\mathtt{n} \in \mathtt{N}$ of BDT is labeled with $l_{\mathtt{n}} \in E$, and a solid (dashed) arc represents 1-arc (0-arc). Square nodes labeled with 1 (0) are 1-terminal (0-terminal).

a root node of $\mathtt{T}_{\mathcal{C}}$. Each $\mathtt{n} \in \mathtt{N}$ is labeled with $l_{\mathtt{n}} \in E$, and $\mathtt{r}$ is labeled with $l_{\mathtt{r}} = 1$. A BDT has two kinds of arcs: 1-arc and 0-arc. Each 1-arc (0-arc) $(\mathtt{m}, \mathtt{n}) \in \mathtt{A}$ represents the situation where $l_{\mathtt{m}} \in E$ is chosen (not chosen), and every $(\mathtt{m}, \mathtt{n}) \in \mathtt{A}$ satisfies $l_{\mathtt{n}} = l_{\mathtt{m}} + 1$. As a result, a BDT has $2^{|E|}$ terminal nodes, and each path connecting $\mathtt{r}$ to a terminal node is associated with a subset, $X \in 2^E$. If $X \in \mathcal{C}$ ($X \notin \mathcal{C}$), then the corresponding terminal node is labeled with 1 (0), which we call 1-terminal (0-terminal). Note that, for ease of understanding, "1" is used as a label of both an element in E and a terminal node; the meaning of the label will be clear from the context. Figure 2.1 presents an example of a BDT.

2.2 Binary Decision Diagrams (BDDs)

While a BDT can store all feasible solutions, its size (or the number of nodes) always increases exponentially in $|E|$. This is a crucial bottleneck when applying BDTs to realistic large-scale problems. The binary decision diagrams (BDDs) [Lee, 1959; Bryant, 1986] were developed to overcome this hardship. We can obtain BDDs by removing redundant parts of BDTs, and BDD sizes can be empirically much smaller than BDT sizes.

Below we detail how to construct a BDD $\mathtt{B}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$, which is a DAG-shaped data structure that can store all $X \in \mathcal{C}$. We let $\mathtt{b} \in \{0, 1\}$, and we

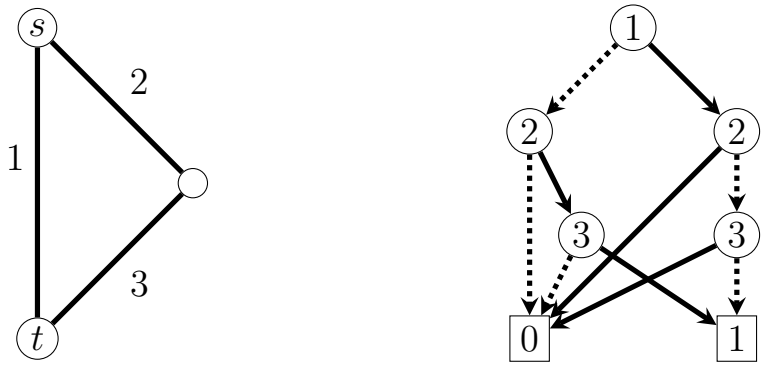


Figure 2.2: Example of BDD $\mathtt{B}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ that stores all s–t paths. Each node $\mathtt{n} \in \mathtt{N}$ is labeled with $l_{\mathtt{n}} \in E$, and a solid (dashed) arc represents 1-arc (0-arc). Square nodes labeled with 1 (0) are 1-terminal (0-terminal).

refer to a node that is pointed to by $\mathtt{b}$-arc outgoing from $\mathtt{n}$ as $\mathtt{b}$-child of $\mathtt{n}$. We use $\mathtt{a}^{\mathtt{b}}_{\mathtt{n}}$ and $\mathtt{c}^{\mathtt{b}}_{\mathtt{n}}$ to denote $\mathtt{n}$'s $\mathtt{b}$-arc and $\mathtt{b}$-child, respectively; consequently $\mathtt{a}^{\mathtt{b}}_{\mathtt{n}} = (\mathtt{n}, \mathtt{c}^{\mathtt{b}}_{\mathtt{n}})$ holds. With these definitions, we describe how to compress BDTs to obtain BDDs.

Merging Rule. A non-terminal node $\mathtt{n} \in \mathtt{N} \backslash \{0, 1\}$ is said to be *sharable* if there exists another node $\mathtt{n}'$ such that $l_{\mathtt{n}} = l_{\mathtt{n}'}$ and $\mathtt{c}^{\mathtt{b}}_{\mathtt{n}} = \mathtt{c}^{\mathtt{b}}_{\mathtt{n}'}$ ($\mathtt{b} \in \{0, 1\}$); namely, $\mathtt{n}$ and $\mathtt{n}'$ have the same label and children. Any sharable node $\mathtt{n}$ can be removed by replacing $(\mathtt{m}, \mathtt{n})$ with $(\mathtt{m}, \mathtt{n}')$.

Elimination Rule. A non-terminal node $\mathtt{n}$ is *redundant* if $\mathtt{c}^0_{\mathtt{n}} = \mathtt{c}^1_{\mathtt{n}}$; i.e., both outgoing edges point to the same node. Any redundant node $\mathtt{n}$ can be removed by replacing all $(\mathtt{m}, \mathtt{n})$ with $(\mathtt{m}, \mathtt{c}^0_{\mathtt{n}})$.

By iteratively removing nodes from a BDT according to the above rules, we can obtain a BDD. We say a BDD is *reduced* if any node is neither sharable nor redundant. Furthermore, a BDD is said to be *ordered* if any $(\mathtt{m}, \mathtt{n}) \in \mathtt{A}$ satisfies $l_{\mathtt{m}} < l_{\mathtt{n}}$. Given a total order of $E = \{1, \ldots, |E|\}$ and $\mathcal{C} \subseteq 2^E$, the reduced and ordered BDD $\mathtt{B}_{\mathcal{C}}$ is known to be unique [Akers, 1978]. Figure 2.2 presents an example of a reduced and ordered BDD.

In practice, the above construction of BDDs that starts from BDTs is often expensive. Alternatively, we can construct BDDs efficiently with *apply*

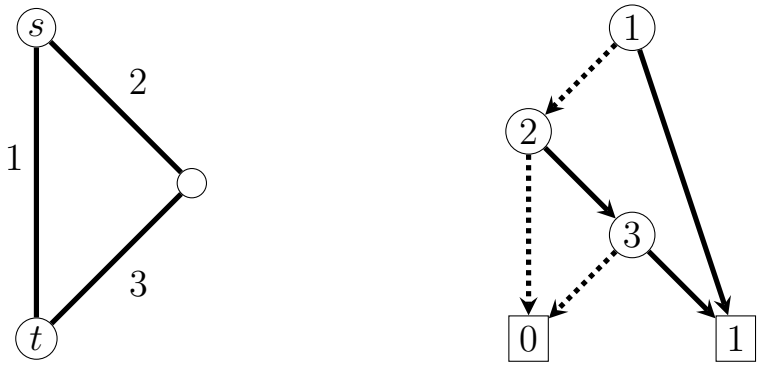


Figure 2.3: Example of ZDD $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ that stores all s–t paths. Each node $\mathtt{n} \in \mathtt{N}$ is labeled with $l_{\mathtt{n}} \in E$, and a solid (dashed) arc represents 1-arc (0-arc). Square nodes labeled with 1 (0) are 1-terminal (0-terminal).

functions [Bryant, 1986], which enable us to perform binary operations (e.g., union and intersection) of two families of sets represented with BDDs.

2.3 Zero-suppressed BDDs (ZDDs)

The above BDDs were originally developed for representing $\mathcal{C} \subseteq 2^E$ that is specified with Boolean functions. In this dissertation, we often consider $\mathcal{C}$ that is specified with combinatorial constraints described on graphs, e.g., s–t paths on a given graph. The zero-suppressed BDD (ZDD) [Minato, 1993] is another DAG-shaped data structure that has been developed for efficiently dealing with such combinatorial constraints.

The definition of ZDD $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ is mostly the same as that of BDDs; the difference is in the elimination rule. Specifically, we employ the following elimination rule alternatively.

Elimination Rule. A non-terminal node $\mathtt{n}$ is *redundant* if $\mathtt{c}^1_{\mathtt{n}} = 0$; i.e., its 1-arc directly points to the 0-terminal. Any redundant node $\mathtt{n}$ can be removed by replacing all $(\mathtt{m}, \mathtt{n})$ with $(\mathtt{m}, \mathtt{c}^0_{\mathtt{n}})$.

By removing all sharable and redundant nodes, we obtain reduced ZDDs. Figure 2.3 shows an example of a ZDD. Compared to the BDD in Figure 2.2, the ZDD has fewer nodes. Empirically, thanks to the elimination rule, ZDDs

Table 2.1: Notation and symbols for ZDDs

$\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$	ZDD that represents $\mathcal{C}$
$\mathtt{r} \in \mathtt{N}$	root node
$0, 1 \in \mathtt{N}$	0- and 1-terminals
$l_{\mathtt{n}} \in E$	label of $\mathtt{n} \in \mathtt{N}$
$\mathtt{a}^{\mathtt{b}}_{\mathtt{n}} \in \mathtt{A}$	$\mathtt{b}$-arc outgoing from $\mathtt{n}$ ($\mathtt{b} \in \{0, 1\}$)
$\mathtt{c}^{\mathtt{b}}_{\mathtt{n}} \in \mathtt{N}$	$\mathtt{n}$-child of $\mathtt{n}$ ($\mathtt{b} \in \{0, 1\}$)
$\mathcal{R}_{\mathtt{m},\mathtt{n}} \subseteq 2^{\mathtt{A}}$	set of all routes connecting $\mathtt{m}$ to $\mathtt{n}$
$X_{\mathtt{R}} \subseteq E$	subset of E represented by route $\mathtt{R}$

tend to be advantageous over BDDs when representing "sparse" sets. For example, when the size of every $X \in \mathcal{C}$ is small relative to $|E|$, ZDDs are often smaller than BDDs. In contrast, BDDs are often advantageous when $\mathcal{C}$ is upward closed; i.e., $X \supseteq Y \in \mathcal{C}$ implies $X \in \mathcal{C}$. This is because their elimination rule (or the "don't care" property) is suitable for representing such $\mathcal{C}$.

For later use, we formally describe the one-to-one correspondence between a subset $X \in \mathcal{C}$ and a path on $\mathtt{Z}_{\mathcal{C}}$ connecting $\mathtt{r}$ to the 1-terminal. We let $\mathcal{R}_{\mathtt{m},\mathtt{n}} \subseteq 2^{\mathtt{A}}$ ($\mathtt{m}, \mathtt{n} \in \mathtt{N}$) to denote a set of directed paths from $\mathtt{m}$ to $\mathtt{n}$. We sometimes call a path on decision diagrams as a route, particularly when paths on graphs and decision diagrams should be distinguished. Given route $\mathtt{R} \subseteq \mathtt{A}$, we define $X_{\mathtt{R}} := \{l_{\mathtt{n}} \in E \mid \mathtt{a}^{1}_{\mathtt{n}} \in \mathtt{R}\}$; i.e., it is a collection of chosen labels on route $\mathtt{R}$. Then, $\mathtt{Z}_{\mathcal{C}}$ satisfies

$$\mathcal{C} = \{X_{\mathtt{R}} \mid \mathtt{R} \in \mathcal{R}_{\mathtt{r},1}\}. \tag{2.1}$$

Namely, there is the one-to-one correspondence between $X \in \mathcal{C}$ and $\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}$. For convenience, we present a summary of notation and symbols related to ZDDs in Table 2.1.

For various practical combinatorial constraints, it is known that ZDDs can be efficiently constructed and that their sizes can be very small. Below we detail ZDD representations of two kinds of constraints: the knapsack constraint and constraints defined on graphs.

Knapsack Constraint. Suppose that each $i \in E$ is associated with a positive integer b_i and that a budget value $B > 0$ is given. We here consider

the *knapsack constraint*; i.e., $\mathcal{C} = \{X \subseteq E \mid \sum_{i \in X} b_i \leq B\}$. In this setting, the ZDD maintains at most B kinds of states for each $i \in E$ (see, also Bergman *et al.* [2016]). Hence the resulting ZDD size is at most $O(dB)$, and it can be constructed in $O(dB)$ time.

Constraints on Graphs. Let $G = (V, E)$ be a graph and $\mathcal{C} \subseteq 2^E$ consist of certain subgraphs (e.g., s–t paths, Steiner trees, matching, and cliques). When constructing ZDDs for such $\mathcal{C}$, the *frontier-based search* (FBS) [Kawahara *et al.*, 2017a] is often used, which is a fast construction algorithm based on Knuth's *Simpath* algorithm [Knuth, 2011]. FBS constructs a ZDD in a top-down manner using the *mate* arrays, which maintain current search states as some positive integers. If ω_G is the *pathwidth* of G and δ is the largest integer stored in the mate arrays, then the size of ZDDs constructed by FBS is bounded from above as $|\mathtt{N}| \leq |E|\delta^{\omega_G}$; empirically, we often have $|\mathtt{N}| \ll |E|\delta^{\omega_G}$ in practice (see, [Inoue and Minato, 2016; Kawahara *et al.*, 2017a] for details). In the worst case we have $\omega_G = |V|$, and thus the resulting ZDD size, $|\mathtt{N}|$, can be exponential in $|V|$. However, if ω_G is bounded by a small constant, $|\mathtt{N}|$ is also bounded. The time and space complexities of FBS are bounded by $O(|E|\delta^{\omega_G})$. In practice, ZDDs for storing subgraphs with various structures can be efficiently obtained by using an existing software [Inoue *et al.*, 2016].

2.4 ZDDs with Vertex Indices

As mentioned above, ZDDs are often used to store all subgraphs $X \in \mathcal{C}$ satisfying certain constraints. Given graph $G = (V, E)$, ZDDs $\mathtt{Z}_{\mathcal{C}}$ usually store families of edges; i.e., $\mathcal{C} \subseteq 2^E$. However, as in Chapter 5, some optimization problems require us to store vertex subsets that induce $X \in \mathcal{C}$, which cannot be stored with standard ZDDs. Fortunately, a variant of ZDDs developed by Suzuki and Minato [2016] can store such vertex subsets. Below we present the details of such ZDDs $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ with vertex indices.

Let $I := E \cup V$ be the set of all edges and vertices of G. We take elements in I to be totally ordered as $I = \{1, \ldots, |V| + |E|\}$ (see, Kawahara *et al.* [2017b] for details of the ordering). Each node $\mathtt{n} \in \mathtt{N} \backslash \{0, 1\}$ is labeled with $l_{\mathtt{n}} \in I$, and the root node is labeled with $1 \in I$, i.e., $l_{\mathtt{r}} = 1$. As with standard ZDDs, each non-terminal node $\mathtt{n}$ has two outgoing arcs (0-arc and 1-arc). Furthermore, we can make ZDDs with vertex indices reduced and ordered. Figure 2.4 presents an example of ZDDs with vertex indices. We define edge

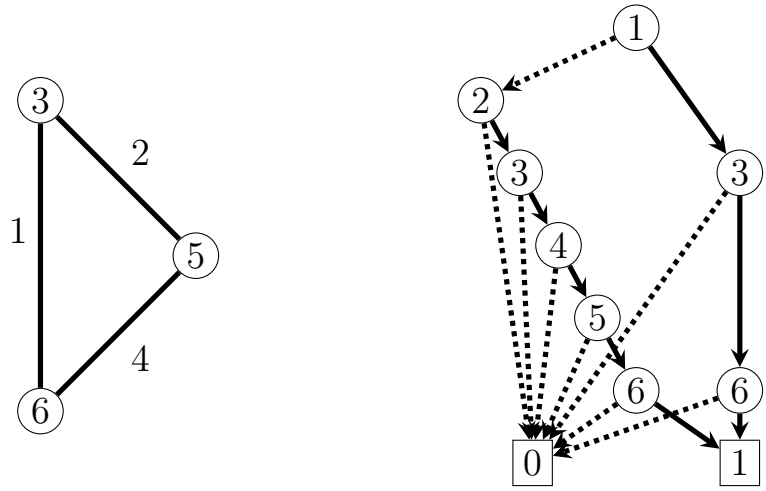


Figure 2.4: Example of ZDD $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ with vertex indices that stores all 3–6 paths. Each node $\mathtt{n} \in \mathtt{N}$ is labeled with $l_{\mathtt{n}} \in E \cup V$, and a solid (dashed) arc represents 1-arc (0-arc). Square nodes labeled with 1 (0) are 1-terminal (0-terminal).

subset $E_{\mathtt{R}} \subseteq E$ and vertex subset $V_{\mathtt{R}} \subseteq V$ that are associated with route $\mathtt{R} \subseteq \mathtt{A}$ as follows:

$$E_{\mathtt{R}} \coloneqq \{l_{\mathtt{n}} \in E \mid \mathtt{a}^1_{\mathtt{n}} \in \mathtt{R}\}, \qquad V_{\mathtt{R}} \coloneqq \{l_{\mathtt{n}} \in V \mid \mathtt{a}^1_{\mathtt{n}} \in \mathtt{R}\}.$$

Analogous to the case of the standard ZDDs, there is a one-to-one correspondence between $X \in \mathcal{C}$ and $\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}$:

$$\mathcal{C} = \{E_{\mathtt{R}} \subseteq E \mid \mathtt{R} \in \mathcal{R}_{\mathtt{r},1}\}.$$

Furthermore, $V_{\mathtt{R}}$ always coincides with the set of vertices that induces $E_{\mathtt{R}}$ for any $\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}$.

We can construct ZDDs with vertex indices by using an algorithm proposed by Kawahara *et al.* [2017b]. The ZDD size can be bounded as $|\mathtt{N}| \leq (|E| + 2|V|)\delta^{\omega_G}$, where ω_G is the pathwidth of G and δ is the largest integer stored in the mate arrays when constructing ZDDs (see, [Suzuki and Minato, 2016] for details).

2.5 Literature Overview on Decision Diagrams and Optimization

Thanks to the recent advances in algorithms for constructing DDs, optimization methods using DDs are receiving much attention. For example, Coudert [1997] presented ZDD-based algorithms for various graph optimization problems. Morrison *et al.* [2016] developed a branch-and-bound algorithm for graph coloring problems by utilizing ZDDs. Those methods are advantageous in that, thanks to their use of DDs representing $\mathcal{C}$, various queries about $\mathcal{C}$ can be answered efficiently, e.g., checking whether given subset X satisfies $X \in \mathcal{C}$ or not, and finding $X \in \mathcal{C}$ that maximizes (or minimizes) linear (or modular) functions of X. For further information, we refer the readers to a comprehensive book by Bergman *et al.* [2016].

Chapter 3

Bandit Combinatorial Optimization with ZDDs

In this chapter, we consider the online combinatorial optimization (OCO) with *bandit* feedback, which we call the *bandit combinatorial optimization* (BCO). A BCO instance generally has a huge set of all feasible solutions, which we here call the *action set*. To avoid dealing with such huge action sets directly, we propose an algorithm that takes advantage of ZDDs, with which we encode action sets as compact DAGs. The proposed algorithm achieves either $O(T^{2/3})$ regret with high probability or $O(\sqrt{T})$ expected regret at any T-th round. Typically, our algorithm works efficiently for BCO problems defined on networks. Experiments show that our algorithm is applicable to various large BCO instances including adaptive routing problems on real-world networks.

3.1 Introduction

The online combinatorial optimization (OCO) problem [Audibert *et al.*, 2014] is a sequential decision problem repeated for $t = 1, \ldots, n$. Let $E \coloneqq [d]$ be a finite set and $\mathcal{C} \subseteq 2^E$ be a collection of all subsets satisfying certain constraints, which we here call an *action set*. In the t-th round, a *player* chooses *action* $X_t \in \mathcal{C}$. Then the player incurs a cost and obtains feedback according to the selected action X_t. The goal of the player is to minimize the cost accumulated up to time horizon n by adaptively choosing an action in each round.

Examples of the OCO include various important problems on networks: the *online shortest path* (OSP) problem [Awerbuch and Kleinberg, 2004; György *et al.*, 2007] and the *dynamic Steiner tree* (DST) problem [Imase and Waxman, 1991]. We can also regard the *congestion game* (CG) [Rosenthal, 1973; Palaiopanos *et al.*, 2017] as a multiple-player version of OCO. For example, in an OSP on a communication network, E is the edge set of the network, an action is an s–r path that connects sender s to receiver r, and action set $\mathcal{C}$ is the set of all s–r paths. The cost of an edge is the transmission time taken to send a message along the edge, and it is expected to dynamically change due to the time-varying congestion or accidents (e.g., cyber attacks). Thus an s–r path must be chosen adaptively each time a message is sent from s to r.

The main difficulty with OCO is that the size of an action set generally increases exponential in $|E|$. To deal with huge action sets, existing methods for OCO problems assume that an action set has certain desirable properties. For instance, some methods assume that an action set is given by m-sets [Kale *et al.*, 2010], permutations [Ailon *et al.*, 2014], or s–r paths on a DAG [Takimoto and Warmuth, 2003; György *et al.*, 2007]. An OCO algorithm that works efficiently with the convex hull of an action set are studied in [Koolen *et al.*, 2010; Combes *et al.*, 2015]. However, many important OCO instances, including OSP, DST, and CG on undirected networks with limited feedback, are still awaiting efficient algorithms.

3.1.1 Our Contribution

We develop an efficient and theoretically guaranteed algorithm for OCO with *bandit* feedback, which we call *bandit combinatorial optimization* (BCO); the details of the feedback are described later. To obtain strong theoretical guarantees, we use the strategy of ComBand [Cesa-Bianchi and Lugosi, 2012], which achieves $O(\sqrt{n})$ *regret* in expectation for BCO problems. We slightly modify ComBand to obtain *any-time guarantees* [Braun and Pokutta, 2016]; we call the modified algorithm CombD (ComBand with decreasing parameters) to distinguish it from the original ComBand. CombD achieves either $O(T^{2/3})$ regret with high probability or $O(\sqrt{T})$ expected regret at any T-th round ($T \in \{1, \dots, n\}$) without knowing n, and we can select which regret value is to be achieved using a hyper parameter.

The naive implementation of CombD, however, is impractical given the huge size of action sets. More precisely, as we will see later, CombD has

two costly steps: sampling an action from the action set and computing unbiased estimators of loss values for all $i \in E$. We address these problems by taking advantage of ZDDs, which we use to encode action sets as compact DAGs. We first show that the sampling step can be completed by applying an existing dynamic programming (DP) method [Takimoto and Warmuth, 2003] to ZDDs. We then provide a novel DP method on ZDDs to compute the unbiased estimators. These methods enable CombD to be performed on ZDDs efficiently, and thus we obtain a practical BCO algorithm, which we call CombD3 (CombD with decision diagrams). The time and space complexities of CombD3 in each round are proportional to ZDD size. While ZDD size can grow exponentially in $|E|$ in general, it is bounded in some cases (e.g., a knapsack constraint and an s–r path constraint on a network with bounded *pathwidth*) as mentioned in Section 2.3; in such cases the complexities of CombD3 are also bounded. In practice, ZDDs tend to be orders of magnitude smaller than action sets, which is experimentally confirmed in [Minato, 1993] and our experiments (Section 3.4). In particular, ZDDs that encode network-based action sets (e.g., those arise in the OSP, DST, and CG) are often small. Experimental results on OSP and DST instances show that CombD3 is far more scalable than ComBand, which deals with action sets directly. We also apply CombD3 to CGs on real-world networks and show that it is useful for practical adaptive routing problems.

3.1.2 Problem Statements

We consider online combinatorial optimization with action set $\mathcal{C} \subseteq 2^E$, where $E = [d]$ is the finite ground set. At each t-th round ($t \in [n]$), an *adversary* secretly chooses *loss vector* $\boldsymbol{\ell}_t \coloneqq (\ell_{t,1}, \dots, \ell_{t,d})^\top \in \mathbb{R}^d$ and the player chooses action $X_t \in \mathcal{C}$. The player then incurs, and observes, the cost $c_t = \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t}$, where $\mathbf{1}_{X_t} \in \{0,1\}^d$ is an indicator vector such that its i-th entry is 1 if $i \in X_t$ and 0 otherwise. This observation setting is called the bandit feedback setting; note that the player *cannot* observe $\boldsymbol{\ell}_t$. The aim of the player is to minimize regret R_T defined as follows for any $T \in [n]$:

$$\mathrm{R}_T \coloneqq \sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \min_{X \in \mathcal{C}} \sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \mathbf{1}_X.$$

The first term is the cumulative cost and the second term is the total cost of the best single action selected with hindsight. Namely, R_T expresses the extra

cost that the player incurs against the best single action. As in [Cesa-Bianchi and Lugosi, 2012], we assume $\max_{t\in[n],X\in\mathcal{C}} |\boldsymbol{\ell}_t^\top \mathbf{1}_X| \leq 1$.

We consider that the player (and the adversary) is allowed to use a randomized strategy to choose X_t (and $\boldsymbol{\ell}_t$). In such cases, R_T is a random variable of a joint distribution $p(\boldsymbol{\ell}_{1:T}, X_{1:T})$, where $X_{1:T} = \{X_1, \ldots, X_T\}$ and $\boldsymbol{\ell}_{1:T} = \{\boldsymbol{\ell}_1, \ldots, \boldsymbol{\ell}_T\}$. Here, p is assumed to satisfy the following conditional independence:

$$p(\boldsymbol{\ell}_{1:T}, X_{1:T}) = \prod_{t\in[T]} p(X_t \mid \boldsymbol{\ell}_{1:t-1}, X_{1:t-1}) p(\boldsymbol{\ell}_t \mid \boldsymbol{\ell}_{1:t-1}, X_{1:t-1}),$$

where $X_{1:0} = \boldsymbol{\ell}_{1:0} = \{\}$. $p(\boldsymbol{\ell}_t \mid X_{1:t-1}, \boldsymbol{\ell}_{1:t-1})$ corresponds to the adversary's strategy and $p(X_t \mid X_{1:t-1}, \boldsymbol{\ell}_{1:t-1})$ corresponds to the player's strategy. Since the player cannot directly observe $\boldsymbol{\ell}_{1:t}$, the player's strategy must satisfy $p(X_t \mid X_{1:t-1}, \boldsymbol{\ell}_{1:t-1}) = p(X_t \mid X_{1:t-1}, c_{1:t-1})$. Using the joint distribution p, expected regret $\overline{\mathrm{R}}_T$ is defined as

$$\overline{\mathrm{R}}_T \coloneqq \max_{X\in\mathcal{C}} \mathbb{E}_{\boldsymbol{\ell}_{1:T}, X_{1:T}\sim p} \left[\sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \sum_{t=1}^{T} \boldsymbol{\ell}_t^\top \mathbf{1}_X \right].$$

The objective of BCO is to design the player's strategy $p(X_t \mid X_{1:t-1}, c_{1:t-1})$ so that it minimizes R_T or $\overline{\mathrm{R}}_T$. We abbreviate $p(X_t \mid X_{1:t-1}, c_{1:t-1})$ by $p_t(X_t)$.

3.1.3 Related Work

There are three well-studied feedback settings for OCO:

Full-information feedback:	$\boldsymbol{\ell}_t$ is observable.
Semi-bandit feedback:	$\{\ell_{t,i} : i \in X_t\}$ is observable.
Bandit feedback:	$c_t = \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t}$ is observable.

BCO is very challenging since its feedback is the least informative, and so it continues to be an attractive research subject [Awerbuch and Kleinberg, 2004; McMahan and Blum, 2004; Bartlett *et al.*, 2008; Dani *et al.*, 2008; Kale *et al.*, 2010; Uchiya *et al.*, 2010; Bubeck *et al.*, 2012; Cesa-Bianchi and Lugosi, 2012; Ailon *et al.*, 2014; Combes *et al.*, 2015]. For further previous works on OCO, we refer the readers to [Audibert *et al.*, 2014]. We here show some relevant existing works focusing on the two features of our algorithm: efficiency and any-time guarantees.

Efficiency. The multiplicative weight update (MWU) strategy with exponential factors (or *Hedge* algorithm [Freund and Schapire, 1999]) is often employed to design efficient OCO algorithms. For the full-information OCO, various efficient Hedge-based algorithms have been developed [Takimoto and Warmuth, 2003; Koolen *et al.*, 2010; Rahmanian *et al.*, 2017]. Our algorithm is related to the Hedge-based algorithm for OSPs on DAGs [Takimoto and Warmuth, 2003]; both their algorithm and ours use the *weight pushing* technique [Mohri, 2009] as their building block. Koolen *et al.* [2010] studied full-information OCO problems whose constraint is characterized by a polytope, and Rahmanian *et al.* [2017] considered full-information OCO problems called *dynamic programming games*; we are interested in a different class of constraints that are represented compactly with ZDDs. Applying the Hedge-based algorithms to BCO problems is not straightforward since those algorithms require $\boldsymbol{\ell}_t$ in each t-th round, which cannot be observed in the bandit feedback setting. A common approach to overcome this difficulty is to use an unbiased estimator $\widehat{\boldsymbol{\ell}}_t$ of $\boldsymbol{\ell}_t$, which can be obtained by computing a *co-occurrence probability matrix* (CPM) as in [Bartlett *et al.*, 2008; Dani *et al.*, 2008; Cesa-Bianchi and Lugosi, 2012]. ComBand [Cesa-Bianchi and Lugosi, 2012] is a general BCO algorithm that employs such an approach and enjoys strong theoretical results. The drawback of ComBand is that its computation cost in each round is generally exponential in d; in particular computing a CPM requires $O(d^2|\mathcal{C}|)$ time. To avoid such expensive computation, Combes *et al.* [2015] proposed CombEXP, which is basically a bandit counterpart of the algorithm presented in [Koolen *et al.*, 2010]. CombEXP scales up ComBand by employing a projection onto the convex hull of an action set via KL-divergence. For some action sets for which the projection can be done efficiently (e.g., m-sets and a set of matchings), CombEXP runs faster than ComBand, while achieving almost the same regrets. However, it is difficult to perform the projection for other action sets (e.g., s–r paths or Steiner trees); actually it is NP-hard to do the projection in the case of OSP and DST on undirected networks. In contrast, our algorithm can work efficiently with those problems thanks to the use of ZDDs, which can represent network-based action sets including those of OSP and DST compactly.

Any-time Guarantees. When using online optimization algorithms in practice, we rarely know how long the algorithms will continue to be executed in advance; in other words the value of time horizon n is seldom available.

Considering this, the regret values of online algorithms should be bounded at any T-th round ($T \in [n]$) without using n. Such regret bounds are called *any-time guarantees* [Braun and Pokutta, 2016]. How to obtain any-time guarantees has been studied for ordinary online learning problems [Cesa-Bianchi and Lugosi, 2006], OCO problems with a variant of semi-bandit feedback [Kocák *et al.*, 2014], and the BCO [Braun and Pokutta, 2016]. However any-time guarantees of ComBand [Cesa-Bianchi and Lugosi, 2012], which we use to construct our BCO algorithm, have not been established;[1] in [Cesa-Bianchi and Lugosi, 2012] it is proved to achieve $O(\sqrt{n})$ expected regret at the n-th round. We prove that ComBand with slight modification, called CombD, achieves any-time guarantees by using the techniques shown in [Cesa-Bianchi and Lugosi, 2006; Braun and Pokutta, 2016].

3.1.4 Overview of Our Algorithm

We provide a high-level sketch of our algorithm. Given action set $\mathcal{C}$, we first construct a ZDD $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ that stores all $X \in \mathcal{C}$ compactly; all procedures of our algorithm are performed on the ZDD, and thus we can avoid dealing with action set $\mathcal{C}$ explicitly. As in Section 2.3, a ZDD is a DAG-shaped data structure, and each action is represented as a path that connects the root and 1-terminal. Therefore the original BCO instance reduces to an OSP on the ZDD with bandit feedback. The BCO algorithm, CombD, for this problem can be performed efficiently on the ZDD by using DP methods. This ZDD-based algorithm is called CombD3, and it runs in $O(d|\mathtt{N}|)$ time in each round. Since $|\mathtt{N}|$ is empirically much smaller than $|\mathcal{C}|$, CombD3 can run much faster than existing BCO algorithms whose complexity depends on $|\mathcal{C}|$.

In Section 3.2 we detail CombD, a BCO algorithm that takes $O(d^2|\mathcal{C}|)$ time in each round in general. In Section 3.3 we explain CombD3.

Remark 1. At first glance, the aforementioned OSP on ZDDs may appear to be solved just by using existing OSP algorithms [Takimoto and Warmuth, 2003; György *et al.*, 2007; Koolen *et al.*, 2010], however it is not true. The problems they considered are OSPs with full-information or semi-bandit feedback, and thus how to obtain the unbiased estimator $\widehat{\boldsymbol{\ell}}_t$ of $\boldsymbol{\ell}_t$, which is

[1] An any-time guarantee of ComBand is stated in [Braun and Pokutta, 2016], but the proof seems to include some mistakes. However, their proof techniques are still useful, and so we prove the $O(T^{2/3})$ high-probability regret bound of CombD by modifying their proof and using the techniques in [Cesa-Bianchi and Lugosi, 2006].

Algorithm 1 CombD($\alpha, \mathcal{C}$)

$\widehat{L}_{0,i} \leftarrow 0$, $w_{1,i} \leftarrow 1$ $(i \in E)$
for $t = 1, \dots, n$ **do**
  $\gamma_t \leftarrow \frac{t^{-1/\alpha}}{2}$, $\eta_{t+1} \leftarrow \frac{\lambda(t+1)^{-1/\alpha}}{2D^2}$
  $X_t \sim p_t$ ▷ output for t-th round
  $c_t \leftarrow \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t}$ ▷ $\boldsymbol{\ell}_t$ is unobservable
  $P_t(i,j) \leftarrow \sum_{X \in \mathcal{C}: i,j \in X} p_t(X)$ $(i,j \in E)$
  $\widehat{\boldsymbol{\ell}}_t \leftarrow c_t P_t^+ \mathbf{1}_{X_t}$
  $\widehat{L}_{t,i} \leftarrow \widehat{L}_{t-1,i} + \widehat{\ell}_{t,i}$ $(i \in E)$
  $w_{t+1,i} \leftarrow \exp\big(-\eta_{t+1} \widehat{L}_{t,i}\big)$ $(i \in E)$
end for

computationally the most difficult part, is unclear in our bandit feedback setting. We overcome this problem by developing novel DP methods that enable us to compute a CPM efficiently on ZDDs; with the CPM we can compute $\widehat{\boldsymbol{\ell}}_t$.

3.2 CombD: ComBand with Decreasing Parameters

We here describe CombD, which is obtained by introducing the decreasing parameters (see, e.g., [Cesa-Bianchi and Lugosi, 2006]) to ComBand [Cesa-Bianchi and Lugosi, 2012].

CombD designs the player's strategy $p_t(X_t)$ as in Algorithm 1. We define $D \coloneqq \max_{X \in \mathcal{C}} \|\mathbf{1}_X\|$ for any given $\mathcal{C} \subseteq 2^E$, where $\|\cdot\|$ is the Euclidian norm. We also define λ as the smallest non-zero eigenvalue of $\mathbb{E}_{X \sim u}[\mathbf{1}_X \mathbf{1}_X^\top]$, where u is the uniform distribution over $\mathcal{C}$. For some cases, the value of λ is proved to be bounded from below [Cesa-Bianchi and Lugosi, 2012; Combes *et al.*, 2015].

Given non-negative vector $\boldsymbol{w} = (w_1, \dots, w_d)^\top \in \mathbb{R}^d$ and action set $\mathcal{C} \subseteq 2^E$, we define the *action distribution* as

$$p(X; \boldsymbol{w}, \mathcal{C}) \coloneqq \frac{w(X)}{Z(\boldsymbol{w}, \mathcal{C})},$$

where

$$w(X) \coloneqq \prod_{i \in X} w_i \qquad \text{and} \qquad Z(\boldsymbol{w}, \mathcal{C}) \coloneqq \sum_{X \in \mathcal{C}} w(X).$$

Using the above, we define the player's strategy $p_t(X_t)$, which appears in Step 4, as follows:

$$p_t(X_t) \coloneqq (1-\gamma_t)p(X_t; \boldsymbol{w}_t, \mathcal{C}) + \gamma_t p(X_t; \mathbf{1}_E, \mathcal{C}), \tag{3.1}$$

where $\boldsymbol{w}_t = (w_{t,1}, \ldots, w_{t,d})^\top$ is the weight vector defined in Step 9, and γ_t is the parameter defined in Step 3. Note that $p(X_t; \mathbf{1}_E, \mathcal{C})$ is the uniform distribution over $\mathcal{C}$. As in (3.1), p_t is a mixture of two action distributions with the mixture rate γ_t.

For any distribution p over $\mathcal{C}$, a matrix P is called a *co-occurrence probability matrix* (CPM) if its (i,j) entry $P(i,j)$ is given by the *co-occurrence probability*

$$p(i \in X, j \in X) \coloneqq \sum_{X \in \mathcal{C}: i,j \in X} p(X).$$

Matrix P_t computed in Step 6 is the CPM of p_t, and P_t^+ used in Step 7 is the pseudo-inverse of P_t. From (3.1), we have

$$P_t(i,j) = \ (1-\gamma_t)p(i \in X, j \in X; \boldsymbol{w}_t, \mathcal{C}) + \gamma_t p(i \in X, j \in X; \mathbf{1}_E, \mathcal{C}).$$

With CPM P_t, the unbiased estimator, $\widehat{\boldsymbol{\ell}}_t$, of $\boldsymbol{\ell}_t$ is obtained as in Step 7; strictly speaking, we have $\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X] = \boldsymbol{\ell}_t^\top \mathbf{1}_X$ for all $X \in \mathcal{C}$, where $\mathbb{E}_t[\cdot]$ is the conditional expectation in the t-th round (see, Section 3.6.2 for details).

The above CombD is based on ComBand [Cesa-Bianchi and Lugosi, 2012]. In fact, CombD with fixed parameters

$$\gamma_t = \frac{D}{\lambda}\sqrt{\frac{\ln|\mathcal{C}|}{n(d/D^2 + 2/\lambda)}} \qquad \text{and} \qquad \eta_t = \frac{1}{D}\sqrt{\frac{\ln|\mathcal{C}|}{n(d/D^2 + 2/\lambda)}}$$

completely corresponds to the original ComBand and achieves $O(\sqrt{n})$ expected regret as follows:

Proposition 1 ([Cesa-Bianchi and Lugosi, 2012])**.** *For any $\mathcal{C}$,* ComBand *achieves*

$$\overline{\mathrm{R}}_n \leq 2\sqrt{\left(\frac{2D^2}{d\lambda} + 1\right) n \ln|\mathcal{C}|}.$$

Whereas the regret of ComBand is bounded only at the n-th round, CombD can achieve the following any-time guarantees:

Theorem 1. *Given any* $\mathcal{C}$*,* CombD$(\alpha = 3, \mathcal{C})$ *achieves*

$$\mathrm{R}_T \leq O\left(\left(\frac{d\lambda}{D^2} + \sqrt{\frac{D^2}{\lambda}\ln\frac{|\mathcal{C}|+2}{\delta}}\right)T^{2/3}\right)$$

with a probability of at least $1-\delta$ *for any fixed* $T \in [n]$*.*

Theorem 2. *Given any* $\mathcal{C}$*,* CombD$(\alpha = 2, \mathcal{C})$ *achieves*

$$\overline{\mathrm{R}}_T \leq O\left(\left(\frac{d\lambda}{D^2} + \frac{D^2\ln|\mathcal{C}|}{\lambda}\right)\sqrt{T}\right)$$

for all $T \in [n]$*.*

In other words, CombD achieves either $O(T^{2/3})$ regret with high probability or $O(\sqrt{T})$ expected regret as an any-time guarantee by setting the value of hyper parameter α appropriately. The technique of using decreasing parameters for online optimization problems is studied by Cesa-Bianchi and Lugosi [2006], and similar results for BCO problems are obtained by Braun and Pokutta [2016]. However, we note that the proofs of Theorem 1 and Theorem 2 cannot be obtained simply by combining those existing results. For details see Section 3.6.

There are two difficulties when performing CombD in practice; the first is sampling from the player's strategy $p_t(X_t)$ (Step 4), and the second is computing CPM P_t (Step 6). Naive methods for the two steps require $O(d|\mathcal{C}|)$ and $O(d^2|\mathcal{C}|)$ times, respectively, where $|\mathcal{C}|$ is generally exponential in d. In the next section, we show efficient ZDD-based methods for sampling from any given action distribution $p(X; \boldsymbol{w}, \mathcal{C})$ and for computing the CPM of $p(X; \boldsymbol{w}, \mathcal{C})$. Since p_t is a mixture of two action distributions as in (3.1), we can efficiently sample from p_t and compute the CPM of p_t using those methods.

3.3 CombD3: CombD with Decision Diagrams

As shown above, CombD requires sampling from action distributions and computing CPMs as its building blocks, which generally require $O(d|\mathcal{C}|)$ and $O(d^2|\mathcal{C}|)$ computation times, respectively. Moreover, computing D can also require $O(d|\mathcal{C}|)$ time. Those computation costs can be prohibitive since $|\mathcal{C}|$ is generally exponential in d. As the building blocks, we present efficient DP

algorithms performed on a ZDD that represents $\mathcal{C}$. We first briefly introduce ZDDs used here, and then present the DP methods for sampling an action and computing CPMs. The sampling method is based on [Takimoto and Warmuth, 2003; Mohri, 2009], while the CPM computation is our proposal. We can also compute D in a DP manner on a ZDD.

3.3.1 ZDDs for Implementing CombD3

As explained in Section 2.3, ZDD $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ can compactly store all $X \in \mathcal{C}$ as follows:

$$\mathcal{C} = \{X_{\mathtt{R}} \mid \mathtt{R} \in \mathcal{R}_{\mathtt{r},1}\},$$

where $X_{\mathtt{R}} \coloneqq \{l_{\mathtt{n}} \in E \mid \mathtt{a}^1_{\mathtt{n}} \in \mathtt{R}\}$ and $\mathcal{R}_{\mathtt{r},1} \subseteq 2^{\mathtt{A}}$ is a collection of routes connecting $\mathtt{r} \in \mathtt{N}$ to $1 \in \mathtt{N}$. For later use, we take the nodes in $\mathtt{N}$ are labeled as $\mathtt{N} = \{0, 1, \ldots, |\mathtt{N}| - 1\}$, where $0 \in \mathtt{N}$ and $1 \in \mathtt{N}$ are 0- and 1-terminals, respectively, and $|\mathtt{N}| - 1 \in \mathtt{N}$ is root $\mathtt{r}$. The other non-terminal nodes are labeled so that $l_{\mathtt{m}} < l_{\mathtt{n}}$ holds if $\mathtt{n} > \mathtt{m}$; i.e., $\mathtt{Z}_{\mathcal{C}}$ is ordered.

Once such $\mathtt{Z}_{\mathcal{C}}$ is obtained, $D = \max_{X \in \mathcal{C}} \sqrt{|X|}$ is easily computed with a DP method to find $\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}$ that maximizes $|X_{\mathtt{R}}|$. As we will see later, the time and space complexities of CombD3 in each round are both $O(d|\mathtt{N}|)$, and thus it runs fast if the ZDD is small. Although $|\mathtt{N}|$ is generally exponential in d, we can sometimes bound $|\mathtt{N}|$ theoretically as exemplified in Section 2.3, and empirically ZDD sizes are very small relative to those theoretical bounds.

3.3.2 Sampling from Action Distributions

We show an efficient algorithm performed on ZDD $\mathtt{Z}_{\mathcal{C}}$ for sampling from action distribution $p(X; \boldsymbol{w}, \mathcal{C})$, which is based on the weight pushing [Takimoto and Warmuth, 2003; Mohri, 2009]; similar methods on DDs have been studied in the field of logic-based probabilistic modeling [Ishihata *et al.*, 2008; Ishihata and Sato, 2011].

We first introduce the following *forward weight* (FW) $\mathrm{F}_{\mathtt{n}}$ and *backward weight* (BW) $\mathrm{B}_{\mathtt{n}}$ ($\mathtt{n} \in \mathtt{N}$):

$$\mathrm{F}_{\mathtt{n}} \coloneqq \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{r},\mathtt{n}}} w(\mathtt{R}), \qquad \mathrm{B}_{\mathtt{n}} \coloneqq \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{n},1}} w(\mathtt{R}),$$

where $w(\mathtt{R})$ is an abbreviation of $w(X_{\mathtt{R}}) = \prod_{i \in X_{\mathtt{R}}} w_i$. Note that we have $Z(\boldsymbol{w}, \mathcal{C}) = \mathrm{B}_{\mathtt{r}} = \mathrm{F}_1$. The weight values of each node, i.e., $\mathrm{B} \coloneqq \{\mathrm{B}_0, \ldots, \mathrm{B}_{\mathtt{r}}\}$

Algorithm 2 FW($\mathtt{Z}_{\mathcal{C}}, \boldsymbol{w}$)

1: $\mathrm{F}_{\mathtt{r}} \leftarrow 1$
2: $\mathrm{F}_{\mathtt{n}} \leftarrow 0$ $(\forall \mathtt{n} \in \mathtt{N} \backslash \{\mathtt{r}\})$
3: **for** $\mathtt{n} = \mathtt{r}, \ldots, 2$ **do**
4: $\quad \mathrm{F}_{\mathtt{c}_{\mathtt{n}}^0} \mathrel{+}= \mathrm{F}_{\mathtt{n}}$
5: $\quad \mathrm{F}_{\mathtt{c}_{\mathtt{n}}^1} \mathrel{+}= w_{l_{\mathtt{n}}} \mathrm{F}_{\mathtt{n}}$
6: **end for**
7: **return** $\mathrm{F} \coloneqq \{\mathrm{F}_0, \ldots, \mathrm{F}_{\mathtt{r}}\}$

Algorithm 3 BW($\mathtt{Z}_{\mathcal{C}}, \boldsymbol{w}$)

1: $\mathrm{B}_1 \leftarrow 1$
2: $\mathrm{B}_{\mathtt{n}} \leftarrow 0$ $(\forall \mathtt{n} \in \mathtt{N} \backslash \{1\})$
3: **for** $\mathtt{n} = 2, \ldots, \mathtt{r}$ **do**
4: $\quad \mathrm{B}_{\mathtt{n}} \leftarrow \mathrm{B}_{\mathtt{c}_{\mathtt{n}}^0} + w_{l_{\mathtt{n}}} \mathrm{B}_{\mathtt{c}_{\mathtt{n}}^1}$
5: **end for**
6: **return** $\mathrm{B} \coloneqq \{\mathrm{B}_1, \ldots, \mathrm{B}_{\mathtt{r}}\}$

and $\mathrm{F} \coloneqq \{\mathrm{F}_0, \ldots, \mathrm{F}_{\mathtt{r}}\}$, can be efficiently computed in a DP manner on $\mathtt{Z}_{\mathcal{C}}$ as shown in Algorithms 2 and 3, respectively. Once we obtain B, we can draw a sample from $p(X; \boldsymbol{w}, \mathcal{C})$ by top-down sampling on $\mathtt{Z}_{\mathcal{C}}$ without rejection as shown in Algorithm 4, where $\mathrm{Ber}(\theta)$ is the Bernoulli distribution with parameter $\theta \in [0, 1]$. The total space and time complexities are both $O(|\mathtt{N}|)$.

3.3.3 Computing Co-occurrence Probabilities

Given action distribution $p(X; \boldsymbol{w}, \mathcal{C})$, we define $\mathrm{P}_{i,j} \coloneqq p(i \in X, j \in X; \boldsymbol{w}, \mathcal{C})$ as the co-occurrence probability of i and j $(i, j \in E)$. We here propose an efficient algorithm for computing $\mathrm{P}_{i,j}$ $(i \leq j)$, which suffices for obtaining $\mathrm{P}_{i,j}$ for all $i, j \in E$ since $\mathrm{P}_{i,j} = \mathrm{P}_{j,i}$. If we compute $\mathrm{P}_{i,j}$ $(i \leq j)$ naively by traversing a ZDD, we need $O(d^2|\mathtt{N}|)$ computation time, which is too expensive if d is large. Fortunately, they can be computed in $O(d|\mathtt{N}|)$ time by using FW, BW, and *backward weighted co-occurrence* (BWC) as follows.

From (2.1), $\mathrm{P}_{i,j}$ can be written as follows:

$$\mathrm{P}_{i,j} = \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{r},1} : i,j \in X_{\mathtt{R}}} \frac{w(\mathtt{R})}{Z(\boldsymbol{w}, \mathcal{C})}. \tag{3.2}$$

Algorithm 4 Draw($\mathtt{Z}_{\mathcal{C}}, \boldsymbol{w}, \mathrm{B}$)

$X \leftarrow \{\}$, $\mathtt{n} \leftarrow \mathtt{r}$
while $\mathtt{n} > 1$ **do**
  $\theta \leftarrow w_{l_\mathtt{n}} \mathrm{B}_{\mathtt{c}^1_\mathtt{n}} / \mathrm{B}_\mathtt{n}$
  $\mathtt{b} \sim \mathrm{Ber}(\theta)$ ▷ $\mathtt{b} = 1$ with probability θ
  if $\mathtt{b} = 1$ **then**
    $X \leftarrow X \cup \{l_\mathtt{n}\}$
  end if
  $\mathtt{n} \leftarrow \mathtt{c}^\mathtt{b}_\mathtt{n}$
end while
return X

We first consider $\mathrm{P}_{i,i}$ as a special case of $\mathrm{P}_{i,j}$, which can be expressed as

$$\mathrm{P}_{i,i} = \sum_{\mathtt{n} \in \mathtt{N}: l_\mathtt{n} = i} \frac{\mathrm{F}_\mathtt{n} w_i \mathrm{B}_{\mathtt{c}^1_\mathtt{n}}}{\mathrm{B}_\mathtt{r}}$$

since we have

$$\begin{aligned}
\mathrm{P}_{i,i} &= \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}: i \in X_\mathtt{R}} \frac{w(\mathtt{R})}{Z(\boldsymbol{w}, \mathcal{C})} \\
&= \sum_{\mathtt{n} \in \mathtt{N}: l_\mathtt{n} = i} \sum_{\substack{\mathtt{R}' \in \mathcal{R}_{\mathtt{r},\mathtt{n}} \\ \mathtt{R}'' \in \mathcal{R}_{\mathtt{c}^1_\mathtt{n},1}}} \frac{w(\mathtt{R}' \cup \{i\} \cup \mathtt{R}'')}{\mathrm{B}_\mathtt{r}} \quad \because Z(\boldsymbol{w}, \mathcal{C}) = \mathrm{B}_\mathtt{r} \\
&= \sum_{\mathtt{n} \in \mathtt{N}: l_\mathtt{n} = i} \frac{\mathrm{F}_\mathtt{n} w_i \mathrm{B}_{\mathtt{c}^1_\mathtt{n}}}{\mathrm{B}_\mathtt{r}} \qquad \because \begin{array}{l} \mathrm{F}_\mathtt{n} = \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{r},\mathtt{n}}} w(\mathtt{R}) \text{ and} \\ \mathrm{B}_\mathtt{n} = \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}} w(\mathtt{R}). \end{array}
\end{aligned}$$

Next, to compute $\mathrm{P}_{i,j}$ $(i < j)$, we rewrite the right hand side of (3.2) using BWC $\mathrm{C}_{\mathtt{n},j} \coloneqq \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{n},1}: j \in X_\mathtt{R}} w(\mathtt{R})$ $(j > l_\mathtt{n})$ as

$$\mathrm{P}_{i,j} = \sum_{v \in V: l_v = i} \frac{\mathrm{F}_v w_i \mathrm{C}_{\mathtt{c}^1_v, j}}{\mathrm{B}_r},$$

which can be obtained as follows:

$$\mathrm{P}_{i,j} = \sum_{\mathtt{R} \in \mathcal{R}_{\mathtt{r},1}: i,j \in X_\mathtt{R}} \frac{w(\mathtt{R})}{Z(\boldsymbol{w}, \mathcal{C})}$$

Algorithm 5 BWC($\mathtt{Z}_{\mathcal{C}}, \boldsymbol{w}, \mathrm{B}$)

$\mathrm{C}_{\mathtt{n},j} \leftarrow 0$ $(\forall \mathtt{n} \in \mathtt{N}, \forall j \in E)$
for $\mathtt{n} = 2, \ldots, \mathtt{r}$ **do**
  $\mathrm{C}_{\mathtt{n},l_{\mathtt{n}}} \leftarrow w_{l_{\mathtt{n}}} \mathrm{B}_{\mathtt{c}^1_{\mathtt{n}}}$
  for $j = l_{\mathtt{n}} + 1, \ldots, d$ **do**
    $\mathrm{C}_{\mathtt{n},j} \leftarrow \mathrm{C}_{\mathtt{c}^0_{\mathtt{n}},j} + w_{l_{\mathtt{n}}} \mathrm{C}_{\mathtt{c}^1_{\mathtt{n}},j}$
  end for
end for
return $\mathrm{C} \coloneqq \{\mathrm{C}_{\mathtt{n},j} \mid \mathtt{n} \in \mathtt{N}, j \geq l_{\mathtt{n}}\}$

Algorithm 6 CPM($\mathtt{Z}_{\mathcal{C}}, \boldsymbol{w}, \mathrm{B}, \mathrm{F}, \mathrm{C}$)

$\mathrm{P}_{i,j} \leftarrow 0$ $(\forall i, j \in E)$
for $\mathtt{n} = 2, \ldots, \mathtt{r}$ **do**
  $i \leftarrow l_{\mathtt{n}}$
  $\mathrm{P}_{i,i} \mathrel{+}= \mathrm{F}_{\mathtt{n}} w_i \mathrm{B}_{\mathtt{c}^1_{\mathtt{n}}} / \mathrm{B}_{\mathtt{r}}$
  for $j = i + 1, \ldots, d$ **do**
    $\mathrm{P}_{i,j} \mathrel{+}= \mathrm{F}_{\mathtt{n}} w_i \mathrm{C}_{\mathtt{c}^1_{\mathtt{n}},j} / \mathrm{B}_{\mathtt{r}}$
  end for
end for
return $\mathrm{P} \coloneqq \{\mathrm{P}_{i,j} \mid i, j \in [d], i \leq j\}$

$$= \sum_{\mathtt{n} \in \mathtt{N}: l_{\mathtt{n}} = i} \sum_{\substack{\mathtt{R}' \in \mathcal{R}_{\mathtt{r},\mathtt{n}} \\ \mathtt{R}'' \in \mathcal{R}_{\mathtt{c}^1_{\mathtt{n}},1}: j \in X_{\mathtt{R}''}}} \frac{w(\mathtt{R}' \cup \{i\} \cup \mathtt{R}'')}{\mathrm{B}_{\mathtt{r}}} \quad \because Z(\boldsymbol{w}, \mathcal{C}) = \mathrm{B}_{\mathtt{r}}$$

$$= \sum_{\mathtt{n} \in \mathtt{N}: l_{\mathtt{n}} = i} \frac{\mathrm{F}_{\mathtt{n}} w_i \mathrm{C}_{\mathtt{c}^1_{\mathtt{n}},j}}{\mathrm{B}_{\mathtt{r}}} \quad \because \begin{array}{l} \mathrm{F}_{\mathtt{n}} = \sum_{R \in \mathcal{R}_{\mathtt{r},\mathtt{n}}} w(\mathtt{R}) \text{ and} \\ \mathrm{C}_{\mathtt{n},j} = \sum_{R \in \mathcal{R}_{\mathtt{n},1}: j \in X_{\mathtt{R}}} w(\mathtt{R}). \end{array}$$

Because $\mathrm{C}_{\mathtt{n},j}$ is a variant of $\mathrm{B}_{\mathtt{n}}$, $\mathrm{C} \coloneqq \{\mathrm{C}_{\mathtt{n},j} \mid \mathtt{n} \in \mathtt{N}, j \geq l_{\mathtt{n}}\}$ can be computed in a similar manner to B as shown in Algorithm 5, whose time and space complexities are both $O(d|\mathtt{N}|)$. To conclude, $\mathrm{P}_{i,j}$ can be computed by Algorithm 6. The total space and time complexities of computing $\mathrm{P} \coloneqq \{\mathrm{P}_{i,j} \mid i \leq j\}$ are both $O(d|\mathtt{N}|)$.

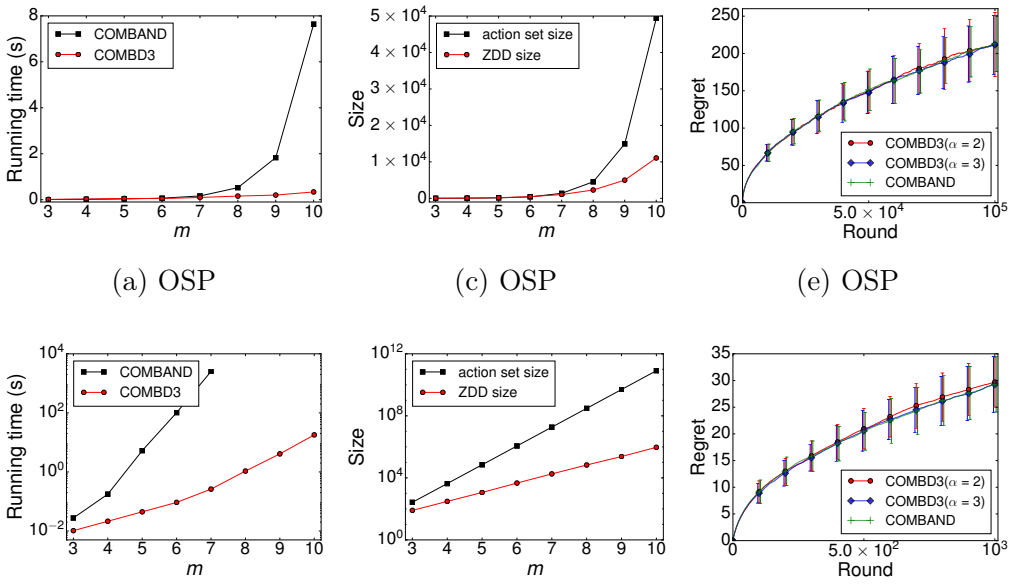
3.4 Experiments

We consider three network-based BCO problems: OSP, DST, and CG. For OSP and DST (Section 3.4.1), we use artificial networks to observe the scalability of algorithms. For CG (Section 3.4.2), we use two real-world networks to show the practical utility of CombD3. All algorithms were implemented in the C programming language, and we used Graphillion [Inoue *et al.*, 2016] to obtain the ZDDs. We note that constructing ZDDs with the software is not a drawback; in all of the following instances ZDD construction took only a few seconds at most.

3.4.1 OSP and DST on Artificial Networks

Experimental Setting. We consider OSP and DST instances on artificial networks, which are undirected grid networks with $3 \times m$ nodes ($m = 3, \dots, 10$). In both problems, E is the edge set of the given network. In OSP, action set $\mathcal{C}$ is the set of all s–r paths from starting node s to goal node r that are placed on diagonal corners of the given grid. In DST, $\mathcal{C}$ is the set of all Steiner trees that contain the four corners of the grid. The aim of the player is to minimize the cumulative cost of the selected subnetworks over some time horizon. In this experiment, we define loss vector $\boldsymbol{\ell}_t$ as follows: We first uniformly sample $\boldsymbol{\mu}_0$ from $[0,1]^d$. In the t-th round, we set $\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t-1}$ with probability 0.9 or draw a new $\boldsymbol{\mu}_t$ uniformly from $[0,1]^d$ with probability 0.1. Then, we set $\ell_{t,i} = 1/d$ with probability $\mu_{t,i}$ and $-1/d$ with probability $1 - \mu_{t,i}$ for each $i \in E$, where $\mu_{t,i}$ is the i-th entry of $\boldsymbol{\mu}_t$. In the short run, this setting is a stochastic OCO problem where each $\ell_{t,i}$ is drawn from $\mathrm{Ber}(\mu_{t,i})$, but the adversary secretly resets $\boldsymbol{\mu}_t$ with probability 0.1 in each round to foil the player.

Computation Cost. We compared CombD3 with ComBand that uses explicit enumeration of possible actions. The running times of the two methods for OSPs and DSTs are shown in Figures 3.1a and 3.1b, respectively; running times of 100 iterations averaged over 100 trials are shown for each method. The results of ComBand for DSTs with $m \geq 8$ are omitted due to memory shortage. Figures 3.1c and 3.1d plot the sizes of action sets, $|\mathcal{C}|$, and corresponding ZDD sizes, $|\mathtt{N}|$, for OSPs and DSTs, respectively. We see that CombD3, which runs in $O(d|\mathtt{N}|)$ time, is far more scalable than ComBand, which runs in $O(d^2|\mathcal{C}|)$ time, thanks to the compactness of ZDDs.



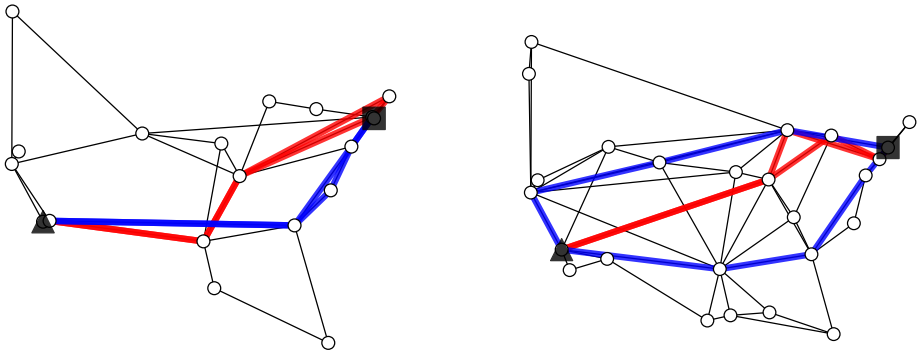
(a) OSP (c) OSP (e) OSP

(b) DST (semi-log) (d) DST (semi-log) (f) DST

Figure 3.1: (a) and (b): Running times that CombD3 and ComBand required for 100 iterations (averaged over 100 trials). (c) and (d): Action set sizes, $|\mathcal{C}|$, and ZDD sizes, $|\mathtt{N}|$. (e) and (f): Achieved regret values of each algorithm on the 3×10 grid; the regret values are averaged over 100 trials and the error bars indicate the standard deviations.

In particular, for DST on the 3×10 grid, we see that $|\mathtt{N}|$ is five orders of magnitude smaller than $|\mathcal{C}|$. In such cases, where existing methods suffer from the complex structure of $\mathcal{C}$ (e.g., no efficient optimization algorithm over $\mathcal{C}$ is available) or the enormous size of $\mathcal{C}$, our CombD3 can be the only practical method.

Empirical Regret. Regret values achieved by CombD3 ($\alpha = 2, 3$) and ComBand for OSP and DST on the 3×10 grid are shown in Figures 3.3a and 3.3b, respectively. As shown previously, ComBand with explicit enumeration is too costly, and thus we here implemented ComBand using ZDDs. We see that the regret values of the three methods actually grow sublinearly. Note that the regret values of CombD3, which does not require n, is comparable to those of ComBand, which requires n as its input. We confirmed that CombD3 yielded lower regret values than those of the theoretical bounds stated in



(a) MCI (b) ATT

Figure 3.2: (a) and (b) are the topologies of the two networks; the triangles (squares) are the starting (goal) nodes. The red (blue) paths indicate the top two paths most frequently chosen by player 1 (2).

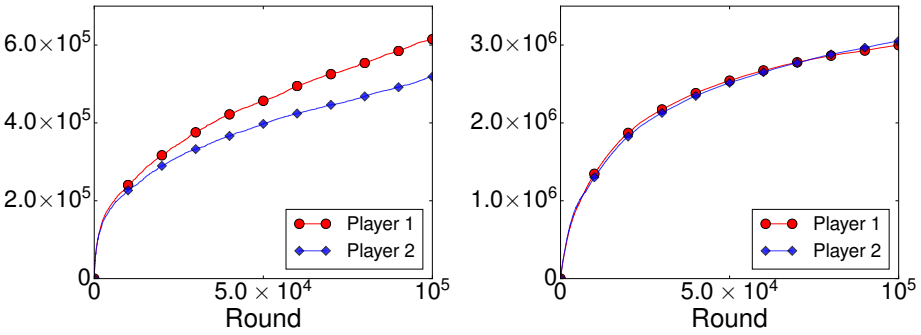
Theorems 1 and 2; the precise values of the bounds are shown in Section 3.6.

3.4.2 CG on Real-world Networks

Experimental Setting. We applied CombD3 to the congestion game (CG), which is a multi-player version of the OSP, on two real-world networks. CG is described as follows: Given m players and an undirected network with starting node s and goal node r, the players concurrently send a message from s to r. The aim of each player is to minimize the cumulative time needed to send n messages. In this problem E is the edge set of the given network, an action is an s–r path, and action set $\mathcal{C}$ is the set of all s–r paths. The loss value of an edge in E is the transmission time taken to send a message along the edge, and the cost of an action is the total transmission time needed to send a message along the selected s–r path. We assume that the loss of each edge increases with the number of players who use the same edge at the same time; formally a player regards the other players as adversaries. We use $X_t^k \in \mathcal{C}$ ($k \in [m]$) to denote the k-th player's choice in the t-th round and use $X_{t,i}^k \in \{0,1\}$ to denote the i-th entry of $\mathbf{1}_{X_t^k}$. We also use $\ell_{t,i}^k$ to denote the transmission time taken by the k-th player to send a message along the i-th edge in the t-th round. We here define $\ell_{t,i}^k := \beta_i \xi^{N_{t,i}^{-k}}$, where $\beta_i \in \mathbb{R}$ is the length of the edge, ξ is an overhead constant, and $N_{t,i}^{-k} := \sum_{k' \neq k} X_{t,i}^{k'}$ is the number of adversaries

Table 3.1: Statistics for two real-world networks.

	# nodes	# edges	# s–r paths	ZDD size
MCI	19	33	1,444	756
ATT	25	56	213,971	37,776



(a) CG on MCI (b) CG on ATT

Figure 3.3: (a) and (b) present regret values of each player for CG on MCI and ATT, respectively.

who also choose the i-th edge at the t-th round. Namely, we assume that the transmission time of each edge increases exponentially with the number of players using the same edge at the same time. Consequently, to reduce the total transmission time, the players should adaptively avoid contending with each other. In this experiments, we set $m = 2$ and $\xi = 10$. This setting violates the assumption $|c_t| < 1$; however, in practice, this violation barely matters. We let both players use CombD3, and the objective of this problem is to let each player avoid contending with each other and reduce his/her transmission time.

We use two real-world communication networks in the Internet topology zoo [Knight *et al.*, 2011]: the InternetMCI network (MCI), and the ATT North America network (ATT). Figures 3.2a and 3.2b illustrate the topologies of MCI and ATT, respectively. Both networks correspond to the U.S. map and we choose Los Angeles as the starting point s and New York as the goal r. The statistics for each network are shown in Table 3.1.

Results. Table 3.1 shows that the action sets of real-world networks can also be represented as compact ZDDs, and thus CombD3 is more efficient than BCO algorithms that deal with the action sets directly. Figures 3.3a and 3.3b plot the regret values of each player for MCI and ATT, respectively. The figures show that each player attained sublinear regrets. Figures 3.2a and 3.2b show the top two most frequently selected paths for each player. We see that each player successfully avoided congestion. For the case where the players can observe the costs of all s–r paths after choosing the current paths, it is known that the MWU-based algorithms can achieve the Nash equilibria on CG [Kleinberg *et al.*, 2009; Palaiopanos *et al.*, 2017]. In this experiment, even though we employed the bandit feedback setting where each player can observe only the cost of the selected path, the players successfully found almost optimal strategies on both networks by using CombD3. To conclude, the results suggest that CombD3 is useful for adaptive routing problems on real-world networks.

3.5 Conclusion

We proposed CombD3, which is a practical and theoretically guaranteed algorithm for BCO. CombD3 is effective particularly for network-based BCO instances, including OSP, DST, and CG. The efficiency of CombD3 is thanks to its use of ZDDs, which offer compact representation of action sets. The time and space complexities of CombD3 are bounded by ZDD size; more precisely, they are $O(d|\mathtt{N}|)$. ZDD size is bounded in some important cases (e.g., a knapsack constraint and some constraints defined on networks with bounded pathwidth), and the complexities of CombD3 are also bounded in such cases. When implementing the BCO algorithm with ZDDs, the most difficult part is designing how to compute the unbiased estimator $\widehat{\boldsymbol{\ell}}_t$ of $\boldsymbol{\ell}_t$, which is obtained using CPMs. We overcame this difficulty by developing DP-based methods for computing CPMs on ZDDs. Experiments showed that CombD3 is far more scalable than the existing method that deals with action sets directly. CombD3 is theoretically guaranteed to achieve either $O(T^{2/3})$ regret with high probability or $O(\sqrt{T})$ expected regret as an any-time guarantee, and we experimentally confirmed that CombD3 attained sublinear regrets. The results on CG showed that CombD3 is useful for adaptive routing problems on real-world networks.

3.6 Proofs of the Theorems

In what follows we prove Theorems 1 and 2. Section 3.6.1 presents two concentration inequalities that are important in the proofs. In Section 3.6.2 we provide some preliminaries for the proofs. Section 3.6.3 and Section 3.6.4 provide the proofs of Theorems 1 and 2, respectively. Most of the following discussions are based on [Cesa-Bianchi and Lugosi, 2006; Braun and Pokutta, 2016], but the proofs cannot be obtained just by combining the existing works. The key to completing the proofs is Lemma 3 presented later, which is important for bounding the difference between the player's choice and the best single action. Lemma 2, which is a variant of [Cesa-Bianchi and Lugosi, 2006, Lemma 2.3], plays an important role in the proof of Lemma 3.

3.6.1 Concentration Inequalities

The following concentration inequalities play crucial roles in the subsequent discussion.

Proposition 2 (Azuma–Hoeffding inequality)**.** *If a martingale difference sequence $\{Z_t\}_{t=1}^T$ satisfies $a_t \le Z_t \le b_t$ almost surely with some constants a_t, b_t for $t = 1, \dots, T$, then the following inequality holds with a probability of at least $1 - \delta$:*

$$\sum_{t=1}^{T} Z_t \le \sqrt{\frac{\ln(1/\delta)}{2} \sum_{t=1}^{T} (b_t - a_t)^2}.$$

Proposition 3 (Bennett's inequality Fan *et al.* [2012])**.** *If a supermartingale difference sequence $\{Z_t\}_{t=1}^T$ with respect to a filtration $\{\mathcal{F}_t\}_{t=0}^{T-1}$ satisfies $Z_t \le b$ with some constant $b > 0$ for $t = 1, \dots, T$, then, for any $v \ge 0$, we have the following with a probability of at least $1 - \delta$:*

$$\sum_{t=1}^{T} \mathrm{Var}[Z_t \mid \mathcal{F}_{t-1}] \ge v \quad \textit{or} \quad \sum_{t=1}^{T} Z_t \le \frac{b}{3} \ln \frac{1}{\delta} + \sqrt{2v \ln \frac{1}{\delta}}.$$

3.6.2 Preliminaries for the Proofs

We here rewrite Algorithm 1 equivalently as in Algorithm 7, which will be helpful to understand the subsequent discussion. We let $K \coloneqq |\mathcal{C}|$ and $\mu \coloneqq 1/K$. We define $\mathbb{E}_t[\cdot] \coloneqq \mathbb{E}[\cdot \mid X_{1:t-1}, \boldsymbol{\ell}_{1:t}]$ as the conditional expectation

Algorithm 7 CombD$(\alpha, \mathcal{C})$

$\widehat{L}_{0,i} \leftarrow 0$, $w_{1,i} \leftarrow 1$ $(i \in E)$
for $t = 1, \ldots, n$ **do**
  $\gamma_t \leftarrow \frac{t^{-1/\alpha}}{2}$, $\eta_{t+1} \leftarrow \frac{\lambda (t+1)^{-1/\alpha}}{2D^2}$
  $X_t \sim p_t$ ▷ output for t-th round
  $c_t \leftarrow \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t}$ ▷ $\boldsymbol{\ell}_t$ is unobservable
  $P_t \leftarrow (1 - \gamma_t) Q_t + \gamma_t U$
  $\widehat{\boldsymbol{\ell}}_t \leftarrow c_t P_t^+ \mathbf{1}_{X_t}$
  $\widehat{L}_{t,i} \leftarrow \widehat{L}_{t-1,i} + \widehat{\ell}_{t,i}$ $(i \in E)$
  $w_{t+1,i} \leftarrow \exp\big(-\eta_{t+1} \widehat{L}_{t,i}\big)$ $(i \in E)$
end for

in the t-th round given the entire history of rounds $1, \ldots, t-1$ and the loss vector in round t. Similarly, we define the conditional variance in the t-th round as $\mathrm{Var}_t[\cdot] \coloneqq \mathrm{Var}[\cdot \mid X_{1:t-1}, \boldsymbol{\ell}_{1:t}]$. For any matrix $P \in \mathbb{R}^{m \times m}$, we denote its i, j entry as $P(i, j)$. We denote the spectral norm of P as $\|P\|$, i.e., $\|P\|$ is the largest singular value of P. For any symmetric matrices $P, Q \in \mathbb{R}^{m \times m}$, we use $P \succeq Q$ to express the fact that the smallest eigenvalue of $P - Q$ is non-negative.

For convenience, we let $\eta_0 = \eta_1 = \frac{\lambda}{2D^2}$. For all $t \in [T]$, we define distributions u and q_t over $\mathcal{C}$, and $d \times d$ matrices U and Q_t as follows:

$$
\begin{aligned}
u(X) &\coloneqq p(X; \mathbf{1}_E, \mathcal{C}) = \mu, \\
q_t(X) &\coloneqq p(X; \boldsymbol{w}_t, \mathcal{C}) = \frac{w_t(X)}{\sum_{X' \in \mathcal{C}} w_t(X')}, \\
U &\coloneqq \mathbb{E}_{X \sim u}[\mathbf{1}_X \mathbf{1}_X^\top] = \sum_{X \in \mathcal{C}} \mu \mathbf{1}_X \mathbf{1}_X^\top, \\
Q_t &\coloneqq \mathbb{E}_{X \sim q_t}[\mathbf{1}_X \mathbf{1}_X^\top] = \sum_{X \in \mathcal{C}} q_t(X) \mathbf{1}_X \mathbf{1}_X^\top.
\end{aligned}
$$

Recall that λ is the smallest non-zero eigenvalue of $U = \mathbb{E}_{X \sim u}[\mathbf{1}_X \mathbf{1}_X^\top]$, and that $|c_t| \leq 1$ holds because of the loss value assumption: $\max_{t \in [n], X \in \mathcal{C}} |\boldsymbol{\ell}_t^\top \mathbf{1}_X| \leq 1$. The following basic results will be used repetitively in what follows.

Lemma 1 (Basic results)**.** *For any $X \in \mathcal{C}$ and $t \in [T]$, we have*

$$
\|P_t^+\| \leq \frac{1}{\gamma_t \lambda} \quad \textit{and} \quad |\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X| \leq \frac{D^2}{\gamma_t \lambda},
$$

$$\mathbb{E}_t[\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}] \le d, \tag{3.3}$$
$$P_t P_t^+ \mathbf{1}_X = \mathbf{1}_X, \tag{3.4}$$
$$\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X] = \boldsymbol{\ell}_t^\top \mathbf{1}_X. \tag{3.5}$$

Proof. The first inequality of eqrefbasic1 comes from $P_t \succeq \gamma_t U$, and the second one is obtained from $|c_t| \le 1$ as follows:

$$|\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X| = |c_t \mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_X| \le \|\mathbf{1}_{X_t}\| \|P_t^+\| \|\mathbf{1}_X\| \le \frac{D^2}{\gamma_t \lambda}.$$

We can obtain (3.3) as follows:

$$\mathbb{E}_t[\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}] = \mathbb{E}_t[\mathrm{tr}(P_t^+ \mathbf{1}_{X_t} \mathbf{1}_{X_t}^\top)] = \mathrm{tr}(P_t^+ P_t) \le d.$$

The proof of (3.4) is presented in [Cesa-Bianchi and Lugosi, 2012, Lemma 14]. Finally, (3.5) is obtained from (3.4) as follows:

$$\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X] = \mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} \mathbf{1}_{X_t} P_t^+ \mathbf{1}_X] = \boldsymbol{\ell}_t^\top P_t P_t^+ \mathbf{1}_X = \boldsymbol{\ell}_t^\top \mathbf{1}_X.$$

□

We also present the following lemma for later use (cf. [Cesa-Bianchi and Lugosi, 2006, Lemma 2.3]).

Lemma 2. *For any $\rho_1, \dots, \rho_N \in (-\infty, \infty)$ and $0 < a \le b$, we have*

$$\frac{1}{a} \ln \sum_{i \in [N]} e^{-a\rho_i} - \frac{1}{b} \ln \sum_{i \in [N]} e^{-b\rho_i} \le \left(\frac{1}{a} - \frac{1}{b} \right) \ln N.$$

Different from [Cesa-Bianchi and Lugosi, 2006, Lemma 2.3], $\rho_1, \dots, \rho_N$ are allowed to be negative, which is actually important in the proof of bandit feedback setting (see the proof of Lemma 3). Lemma 2 is proved using Hölder's inequality as follows.

Proof. By Hölder's inequality (or the property of the p-norm), $\|\boldsymbol{x}\|_p \le N^{\frac{1}{p} - \frac{1}{q}} \|\boldsymbol{x}\|_q$ holds for any $\boldsymbol{x} \in \mathbb{R}^N$ and $1 \le p \le q$. Thus, by letting $p = 1$ and $q = b/a$, we obtain

$$\sum_{i \in [N]} e^{-a\rho_i} \le N^{1 - \frac{a}{b}} \left(\sum_{i \in [N]} \left(e^{-b\rho_i \frac{a}{b}} \right)^{\frac{b}{a}} \right)^{\frac{a}{b}}.$$

Hence we have

$$\left(\sum_{i\in[N]} e^{-a\rho_i}\right)^{\frac{1}{a}} \leq N^{\frac{1}{a}-\frac{1}{b}} \left(\sum_{i\in[N]} \left(e^{-b\rho_i}\right)^{\frac{a}{b}}\right)^{\frac{a}{b}\frac{1}{a}} = N^{\frac{1}{a}-\frac{1}{b}} \left(\sum_{i\in[N]} e^{-b\rho_i}\right)^{\frac{1}{b}}.$$

The proof is completed by taking the natural logarithm of both sides and rearranging the terms. □

3.6.3 Proof for the High-probability Regret Bound

We show the complete proof of Theorem 1. Below is a detailed statement of the theorem.

Theorem 3. *Given any $X \in \mathcal{C}$ and any fixed $T \in [n]$, the sequence of actions $\{X_t\}_{t\in[T]}$ obtained by* CombD$(\alpha = 3, \mathcal{C})$ *satisfies the following inequality with a probability of at least $1-\delta$:*

$$\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \boldsymbol{\ell}_t^\top \mathbf{1}_X) \leq \left(\frac{3d(e-2)\lambda}{4D^2} + \frac{3}{2} + D\sqrt{\frac{7}{\lambda}\ln\frac{K+2}{\delta}}\right)T^{2/3} + o(T^{2/3}).$$

Let $\widetilde{\boldsymbol{x}}_t \coloneqq \sum_{X\in\mathcal{C}} q_t(X)\mathbf{1}_X$. As in [Braun and Pokutta, 2016], the proof is obtained by bounding each term on the right hand side of the following equation:

$$\begin{aligned}
&\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \boldsymbol{\ell}_t^\top \mathbf{1}_X) \\
&= \sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t) + \sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) + \sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X),
\end{aligned}$$

where $X \in \mathcal{C}$ is an arbitrary action. To bound them, we prove the following three lemmas.

Lemma 3. *The following inequality holds with a probability of at least $1-\delta$:*

$$\begin{aligned}
&\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) \\
&\leq \frac{\ln K}{\eta_T} + (e-2)\left(d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t} + \frac{D^2}{\lambda}\sqrt{\frac{1}{2}\ln\frac{1}{\delta}\sum_{t=1}^{T}\frac{\eta_t^2}{\gamma_t^2(1-\gamma_t)^2}}\right).
\end{aligned}$$

The proof is based on the technique frequently used in the analysis of online optimization algorithms. Specifically, we define an appropriate *potential function* and evaluate the progress of the algorithm using it for each round; to do this we use Lemma 2. Finally we use the concentration inequalities to obtain high-probability bounds as in [Braun and Pokutta, 2016].

Proof. With the weight values $w_{t,i}$ $(i \in E)$ used in Algorithm 7, we define $w_t(X)$ and $W_t(X)$ for any $X \in \mathcal{C}$ and $t \in [T]$ as follows:

$$
\begin{aligned}
w_1(X) &\coloneqq 1, \\
w_t(X) &\coloneqq \prod_{i \in X} w_{t,i} = \prod_{i \in X} \exp\left(-\eta_t \sum_{t'=1}^{t-1} \widehat{\ell}_{t',i}\right) \quad \text{for } t \geq 2, \\
W_1 &\coloneqq K, \\
W_t &\coloneqq \sum_{X \in \mathcal{C}} w_t(X)^{\eta_{t-1}/\eta_t} = \sum_{X \in \mathcal{C}} \exp\left(-\eta_{t-1} \sum_{t'=1}^{t-1} \widehat{\boldsymbol{\ell}}_{t'}^{\top} \mathbf{1}_X\right) \quad \text{for } t \geq 2.
\end{aligned}
$$

By Lemma 2, we obtain

$$
\begin{aligned}
&\frac{1}{\eta_t} \ln \sum_{X \in \mathcal{C}} \exp\left(-\eta_t \sum_{t'=1}^{t-1} \widehat{\boldsymbol{\ell}}_{t'}^{\top} \mathbf{1}_X\right) - \frac{1}{\eta_{t-1}} \ln \sum_{X \in \mathcal{C}} \exp\left(-\eta_{t-1} \sum_{t'=1}^{t-1} \widehat{\boldsymbol{\ell}}_{t'}^{\top} \mathbf{1}_X\right) \\
&\leq \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) \ln K,
\end{aligned}
$$

which can be written as follows using the definitions of $w_t(X)$ and $W_t(X)$:

$$
\frac{1}{\eta_t} \ln \sum_{X \in \mathcal{C}} w_t(X) \leq \frac{1}{\eta_{t-1}} \ln W_t + \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) \ln K.
$$

Hence we can bound the difference of potential functions, $\ln W_{t+1}^{\eta_t^{-1}} - \ln W_t^{\eta_{t-1}^{-1}}$, as follows:

$$
\begin{aligned}
&\frac{1}{\eta_t} \ln W_{t+1} - \frac{1}{\eta_{t-1}} W_t - \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) \ln K \\
&\leq \frac{1}{\eta_t} \ln \frac{W_{t+1}}{\sum_{X' \in \mathcal{C}} w_t(X')} \\
&= \frac{1}{\eta_t} \ln \sum_{X \in \mathcal{C}} \frac{w_t(X) \exp(-\eta_t \widehat{\boldsymbol{\ell}}_t^{\top} \mathbf{1}_X)}{\sum_{X' \in \mathcal{C}} w_t(X')}
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{1}{\eta_t} \ln \sum_{X \in \mathcal{C}} q_t(X) \exp(-\eta_t \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) \\
&\leq \frac{1}{\eta_t} \ln \sum_{X \in \mathcal{C}} q_t(X) \left(1 - \eta_t \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X + (e-2)\eta_t^2 (\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X)^2\right) \\
&= \frac{1}{\eta_t} \ln \left(1 - \eta_t \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t + (e-2)\eta_t^2 \sum_{X \in \mathcal{C}} q_t(X) (\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X)^2\right) \\
&\leq -\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t + (e-2)\eta_t \sum_{X \in \mathcal{C}} q_t(X) (\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X)^2,
\end{aligned}
$$

where the second inequality comes from $e^{-x} \leq 1 - x + (e-2)x^2$ for any $|x| \leq 1$; note that η_t is defined to satisfy $\eta_t |\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X| \leq \eta_t D^2/(\gamma_t \lambda) = 1$. The third inequality is obtained by $\ln(1+x) \leq x$ for any $x \geq -1$. The second term on the right hand side is bounded from above as follows:

$$
\sum_{X \in \mathcal{C}} q_t(X) (\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X)^2 \leq \sum_{X \in \mathcal{C}} \frac{p_t(X)}{1-\gamma_t} (\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X)^2 \leq \frac{\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t}.
$$

Therefore, we have

$$
\frac{1}{\eta_t} \ln W_{t+1} - \frac{1}{\eta_{t-1}} \ln W_t \leq \left(\frac{1}{\eta_t} - \frac{1}{\eta_{t-1}}\right) \ln K - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t + (e-2)\eta_t \frac{\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t}.
$$

Summing up both sides of the above for $t = 1, \ldots, T$, we obtain the following inequality from $W_1 = K$:

$$
\frac{1}{\eta_T} \ln W_{T+1} \leq \frac{1}{\eta_T} \ln K - \sum_{t=1}^T \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t + (e-2) \sum_{t=1}^T \eta_t \frac{\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t}.
$$

On the other hand, we have

$$
\begin{aligned}
\frac{1}{\eta_T} \ln W_{T+1} &= \frac{1}{\eta_T} \ln \sum_{X' \in \mathcal{C}} \exp\left(-\eta_T \sum_{t'=1}^T \widehat{\boldsymbol{\ell}}_{t'}^\top \mathbf{1}_{X'}\right) \\
&\geq \frac{1}{\eta_T} \ln \exp\left(-\eta_T \sum_{t'=1}^T \widehat{\boldsymbol{\ell}}_{t'}^\top \mathbf{1}_X\right) \\
&= -\sum_{t=1}^T \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X.
\end{aligned}
$$

Therefore, we obtain

$$\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) \leq \frac{\ln K}{\eta_T} + (e-2)\sum_{t=1}^{T}\eta_t \frac{\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t}. \tag{3.6}$$

The second term on the right hand side can be bounded from above by using the Azuma–Hoeffding inequality (Proposition 2) for the sequence of martingale difference, $\frac{\eta_t}{1-\gamma_t}(\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t} - \mathbb{E}_t[\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}])$, as follows. First, note that we have

$$\mathbb{E}_t[\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}] \leq d \quad \text{and} \quad 0 \leq \frac{\eta_t \mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t} \leq \frac{\eta_t D^2}{(1-\gamma_t)\gamma_t \lambda}$$

by Lemma 1. Thus, by the Azuma-Hoeffding inequality, the following holds with a probability of at least $1-\delta$:

$$\sum_{t=1}^{T}\frac{\eta_t \mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t} \leq d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t} + \frac{D^2}{\lambda}\sqrt{\frac{\ln(1/\delta)}{2}\sum_{t=1}^{T}\frac{\eta_t^2}{\gamma_t^2(1-\gamma_t)^2}}.$$

Hence we have the following inequality with a probability of at least $1-\delta$:

$$\begin{aligned}
&\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) \\
&\leq \frac{\ln K}{\eta_T} + (e-2)\left(d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t} + \frac{D^2}{\lambda}\sqrt{\frac{\ln(1/\delta)}{2}\sum_{t=1}^{T}\frac{\eta_t^2}{\gamma_t^2(1-\gamma_t)^2}}\right).
\end{aligned}$$

□

Lemma 4 ([Braun and Pokutta, 2016])**.** *The following inequality holds with a probability of at least* $1-\delta$*:*

$$\begin{aligned}
\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t) \leq 2\sum_{t=1}^{T}\gamma_t + \frac{1}{3}\left(2 + \frac{D}{\sqrt{\lambda\gamma_T(1-\gamma_T)}}\right)\ln\frac{1}{\delta} \\
+ \sqrt{2\left(T + \frac{3D^2}{\lambda}\sum_{t=1}^{T}\frac{\gamma_t}{(1-\gamma_t)^2}\right)\ln\frac{1}{\delta}}.
\end{aligned}$$

Proof. Let $\boldsymbol{z} \coloneqq \sum_{X \in \mathcal{C}} \mu \mathbf{1}_X$ and $\overline{\boldsymbol{x}}_t \coloneqq \mathbb{E}_t[\mathbf{1}_{X_t}] = (1-\gamma_t)\widetilde{\boldsymbol{x}}_t + \gamma_t \boldsymbol{z}$. We obtain the proof by using Bennett's inequality (Proposition 3) for the martingale difference sequence

$$\begin{aligned} Y_t &\coloneqq \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t] \\ &= \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \boldsymbol{\ell}_t^\top \overline{\boldsymbol{x}}_t + \boldsymbol{\ell}_t^\top \widetilde{\boldsymbol{x}}_t \\ &= \boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t + \gamma_t \boldsymbol{\ell}_t^\top (\widetilde{\boldsymbol{x}}_t - \boldsymbol{z}). \end{aligned}$$

We first bound the values of $|Y_t|$ and $\mathrm{Var}_t[Y_t]$. By $Q_t \preceq \frac{1}{1-\gamma_t} P_t$ and Jensen's inequality

$$(\boldsymbol{x}^\top \widetilde{\boldsymbol{x}}_t)^2 = \left(\sum_{X \in \mathcal{C}} q_t(X) \mathbf{1}_X^\top \boldsymbol{x} \right)^2 \leq \sum_{X \in \mathcal{C}} q_t(X) \left(\mathbf{1}_X^\top \boldsymbol{x} \right)^2 = \boldsymbol{x}^\top Q_t \boldsymbol{x} \quad \forall \boldsymbol{x} \in \mathbb{R}^d,$$

we have

$$(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t)^2 \leq c_t^2 \mathbf{1}_{X_t}^\top P_t^+ Q_t P_t^+ \mathbf{1}_{X_t} \leq \frac{\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t} \leq \frac{D^2}{\lambda \gamma_t (1-\gamma_t)},$$

and hence

$$|Y_t| \leq 1 + \frac{D}{\sqrt{\lambda \gamma_t (1-\gamma_t)}} + 2\gamma_t \leq 2 + \frac{D}{\sqrt{\lambda \gamma_T (1-\gamma_T)}}.$$

The variance of Y_t is bounded as follows:

$$\begin{aligned} \mathrm{Var}_t[Y_t] &\leq \mathbb{E}_t[(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t)^2] \\ &= \mathbb{E}_t[c_t^2 (1 - \mathbf{1}_{X_t}^\top P_t^+ \widetilde{\boldsymbol{x}}_t)^2] \leq \mathbb{E}_t[(1 - \mathbf{1}_{X_t}^\top P_t^+ \widetilde{\boldsymbol{x}}_t)^2] \\ &= \mathbb{E}_t[1 - 2\mathbf{1}_{X_t}^\top P_t^+ \widetilde{\boldsymbol{x}}_t + \widetilde{\boldsymbol{x}}_t^\top P_t^+ \mathbf{1}_{X_t} \mathbf{1}_{X_t}^\top P_t^+ \widetilde{\boldsymbol{x}}_t] \\ &= 1 - 2\overline{\boldsymbol{x}}_t^\top P_t^+ \widetilde{\boldsymbol{x}}_t + \widetilde{\boldsymbol{x}}_t^\top P_t^+ \widetilde{\boldsymbol{x}}_t \\ &= 1 - \frac{2}{1-\gamma_t} \overline{\boldsymbol{x}}_t^\top P_t^+ (\overline{\boldsymbol{x}}_t - \gamma_t \boldsymbol{z}) + \frac{1}{(1-\gamma_t)^2} (\overline{\boldsymbol{x}}_t - \gamma_t \boldsymbol{z})^\top P_t^+ (\overline{\boldsymbol{x}}_t - \gamma_t \boldsymbol{z}) \\ &= 1 - \frac{1-2\gamma_t}{(1-\gamma_t)^2} \overline{\boldsymbol{x}}_t^\top P_t^+ \overline{\boldsymbol{x}}_t + \frac{\gamma_t^2}{(1-\gamma_t)^2} (\boldsymbol{z} - 2\overline{\boldsymbol{x}}_t)^\top P_t^+ \boldsymbol{z} \\ &\leq 1 + \frac{\gamma_t^2}{(1-\gamma_t)^2} (\boldsymbol{z} - 2\overline{\boldsymbol{x}}_t)^\top P_t^+ \boldsymbol{z} \end{aligned}$$

$$\leq 1 + \frac{3\gamma_t D^2}{(1-\gamma_t)^2\lambda},$$

where the third inequality comes from $1 - 2\gamma_t \geq 0$, and the last inequality is obtained by Lemma 1 with $\|\boldsymbol{z}\| \leq D$ and $\|\overline{\boldsymbol{x}}_t\| \leq D$. Therefore, from Bennett's inequality, the following inequality holds with a probability of at least $1 - \delta$:

$$\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t + \gamma_t \boldsymbol{\ell}_t^\top(\widetilde{\boldsymbol{x}}_t - \boldsymbol{z})) \leq \frac{1}{3}\left(2 + \frac{D}{\sqrt{\lambda\gamma_T(1-\gamma_T)}}\right)\ln\frac{1}{\delta} + \sqrt{2\left(T + \frac{3D^2}{\lambda}\sum_{t=1}^{T}\frac{\gamma_t}{(1-\gamma_t)^2}\right)\ln\frac{1}{\delta}}.$$

The proof is completed by $|\boldsymbol{\ell}_t^\top(\widetilde{\boldsymbol{x}}_t - \boldsymbol{z})| \leq 2$. □

Lemma 5 ([Braun and Pokutta, 2016])**.** *The following inequality holds for all $X \in \mathcal{C}$ simultaneously with a probability of at least $1 - \delta$:*

$$\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X) \leq \frac{1}{3}\left(1 + \frac{D^2}{\gamma_T\lambda}\right)\ln\frac{K}{\delta} + \sqrt{\frac{2D^2}{\lambda}\ln\frac{K}{\delta}\sum_{t=1}^{T}\frac{1}{\gamma_t}}.$$

Proof. We fix $X \in \mathcal{C}$ arbitrarily. The proof is obtained by using Bennett's inequality for the martingale difference sequence $\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X$; note that $\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X] = 0$ holds by Lemma 1. First, the absolute value and variance of $\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X$ are bounded as follows:

$$|\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X| \leq 1 + \frac{D^2}{\gamma_t\lambda}$$

and

$$\begin{aligned}\mathrm{Var}_t[\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X] &\leq \mathbb{E}_t[(\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X)^2] \leq \mathbb{E}_t[\mathbf{1}_X^\top P_t^+ \mathbf{1}_{X_t}\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_X] \\ &\leq \mathbf{1}_X P_t^+ \mathbf{1}_X \leq \frac{D^2}{\gamma_t\lambda}.\end{aligned}$$

Hence, by Bennett's inequality, we have

$$\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X) \leq \frac{1}{3}\left(1 + \frac{D^2}{\gamma_T\lambda}\right)\ln\frac{K}{\delta} + \sqrt{\frac{2D^2}{\lambda}\ln\frac{K}{\delta}\sum_{t=1}^{T}\frac{1}{\gamma_t}}$$

with probability at least $1 - \delta/K$. Taking the union bound over all actions $X \in \mathcal{C}$, we obtain the claim. □

Using the above three lemmas, we prove Theorem 3 as follows.

Proof of Theorem 3. Note that we have $\gamma_t = \frac{t^{-1/3}}{2}$ and $\eta_t = \frac{\lambda}{D^2}\gamma_t = \frac{\lambda t^{-1/3}}{2D^2}$. By using Lemma 3, we obtain the following inequality with a probability of at least $1 - \delta/(K+2)$:

$$\begin{aligned}
&\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) \\
&\leq \frac{\ln K}{\eta_T} + (e-2)\left(d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t} + \frac{D^2}{\lambda}\sqrt{\frac{1}{2}\ln\frac{K+2}{\delta}\sum_{t=1}^{T}\frac{\eta_t^2}{\gamma_t^2(1-\gamma_t)^2}}\right) \\
&\leq \frac{2D^2\ln K}{\lambda}T^{1/3} + (e-2)\left(\frac{3d\lambda}{4D^2}(T^{2/3}+2T^{1/3}) + \sqrt{2T\ln\frac{K+2}{\delta}}\right).
\end{aligned}$$

We also obtain the following inequality with a probability of at least $1 - \delta/(K+2)$ by using Lemma 4:

$$\begin{aligned}
\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t) &\leq 2\sum_{t=1}^{T}\gamma_t + \frac{1}{3}\left(2 + \frac{D}{\sqrt{\lambda\gamma_T(1-\gamma_T)}}\right)\ln\frac{K+2}{\delta} \\
&\quad + \sqrt{2\left(T + \frac{3D^2}{\lambda}\sum_{t=1}^{T}\frac{\gamma_t}{(1-\gamma_t)^2}\right)\ln\frac{K+2}{\delta}} \\
&\leq \frac{3}{2}T^{2/3} + \frac{1}{3}\left(2 + \frac{\sqrt{2}D}{\sqrt{\lambda}}(T^{1/6}+T^{-1/6})\right)\ln\frac{K+2}{\delta} \\
&\quad + \sqrt{2\left(T + \frac{9D^2}{\lambda}T^{2/3}\right)\ln\frac{K+2}{\delta}}.
\end{aligned}$$

Furthermore, we have the following inequality with a probability of at least $1 - K\delta/(K+2)$ by using Lemma 5:

$$\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X - \boldsymbol{\ell}_t^\top \mathbf{1}_X)$$

$$\leq \frac{1}{3}\left(1+\frac{D^2}{\gamma_T\lambda}\right)\ln\frac{K+2}{\delta}+\sqrt{\frac{2D^2}{\lambda}\ln\frac{K+2}{\delta}\sum_{t=1}^{T}\frac{1}{\gamma_t}}$$
$$\leq \frac{1}{3}\left(1+\frac{2D^2}{\lambda}T^{1/3}\right)\ln\frac{K+2}{\delta}+\frac{\sqrt{3}D}{\sqrt{\lambda}}T^{2/3}\sqrt{\left(1+\frac{4}{3T}\right)\ln\frac{K+2}{\delta}}.$$

Summing up both sides of the three inequalities and taking the union bound, we obtain the theorem. □

3.6.4 Proof for the Expected Regret Bound

We now show the proof of Theorem 2; the detailed statement is as follows.

Theorem 4. *Given any $X \in \mathcal{C}$, the sequence of actions $\{X_t\}_{t\in[T]}$ obtained by* CombD$(\alpha = 2, \mathcal{C})$ *satisfies the following inequality for all $T \in [n]$:*

$$\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \boldsymbol{\ell}_t^\top \mathbf{1}_X)\right] \leq \left(\frac{2D^2\ln K}{\lambda}+\frac{(e-2)d\lambda}{D^2}+2\right)\sqrt{T}+o(\sqrt{T}).$$

Let $\widetilde{\boldsymbol{x}}_t \coloneqq \sum_{X\in\mathcal{C}} q_t(X)\mathbf{1}_X$. The proof is obtained by bounding each term on the right hand side of the following equation for any $X \in \mathcal{C}$:

$$\begin{aligned}
&\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \boldsymbol{\ell}_t^\top \mathbf{1}_X)\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \boldsymbol{\ell}_t^\top \mathbf{1}_X]\right] \\
&= \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t]\right] + \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X]\right], \qquad (3.7)
\end{aligned}$$

where the second equality comes from Lemma 1. To bound these terms, we prove the following two lemmas.

Lemma 6. *The following inequality holds:*

$$\mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X]\right] \leq \frac{\ln K}{\eta_T} + (e-2)d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t}.$$

Proof. As in the proof of Lemma 3, we have (3.6):

$$\sum_{t=1}^{T}(\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X) \leq \frac{\ln K}{\eta_T} + (e-2)\sum_{t=1}^{T}\eta_t \frac{\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}}{1-\gamma_t}.$$

Taking the expectation of both sides, we obtain

$$\begin{aligned}\mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X]\right] &\leq \frac{\ln K}{\eta_T} + (e-2)\mathbb{E}\left[\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t}\mathbb{E}_t[\mathbf{1}_{X_t}^\top P_t^+ \mathbf{1}_{X_t}]\right] \\ &\leq \frac{\ln K}{\eta_T} + (e-2)d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t},\end{aligned}$$

where the second inequality is obtained by Lemma 1. □

Lemma 7. *The following inequality holds:*

$$\mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t]\right] \leq 2\sum_{t=1}^{T}\gamma_t.$$

Proof. Since $|\boldsymbol{\ell}_t^\top \mathbf{1}_X| \leq 1$ holds for any $X \in \mathcal{C}$, we have

$$\begin{aligned}\mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t] = \mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t}] - \boldsymbol{\ell}_t^\top \widetilde{\boldsymbol{x}}_t &= \sum_{X\in\mathcal{C}} p_t(X)\boldsymbol{\ell}_t^\top \mathbf{1}_X - \sum_{X\in\mathcal{C}} q_t(X)\boldsymbol{\ell}_t^\top \mathbf{1}_X \\ &= \sum_{X\in\mathcal{C}} \gamma_t \mu \boldsymbol{\ell}_t^\top \mathbf{1}_X - \sum_{X\in\mathcal{C}} \gamma_t q_t(X)\boldsymbol{\ell}_t^\top \mathbf{1}_X \leq 2\gamma_t.\end{aligned}$$

Summing up both sides for $t = 1, \ldots, T$ and taking the expectation, we obtain the claim. □

We are now ready to prove Theorem 4 as follows.

Proof of Theorem 4. Recall that we have $\gamma_t = \frac{t^{-1/2}}{2}$ and $\eta_t = \frac{\lambda}{D^2}\gamma_t = \frac{\lambda t^{-1/2}}{2D^2}$. The proof is readily obtained by (3.7), Lemma 6 and Lemma 7 as follows:

$$\begin{aligned}&\mathbb{E}\left[\sum_{t=1}^{T}(\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \boldsymbol{\ell}_t^\top \mathbf{1}_X)\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\boldsymbol{\ell}_t^\top \mathbf{1}_{X_t} - \widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t]\right] + \mathbb{E}\left[\sum_{t=1}^{T}\mathbb{E}_t[\widehat{\boldsymbol{\ell}}_t^\top \widetilde{\boldsymbol{x}}_t - \widehat{\boldsymbol{\ell}}_t^\top \mathbf{1}_X]\right]\end{aligned}$$

$$\leq \frac{\ln K}{\eta_T} + (e-2)d\sum_{t=1}^{T}\frac{\eta_t}{1-\gamma_t} + 2\sum_{t=1}^{T}\gamma_t$$
$$\leq \frac{2D^2\ln K}{\lambda}\sqrt{T} + \frac{(e-2)d\lambda}{D^2}\left(\sqrt{T} + \frac{1}{2}\ln(2\sqrt{T}-1)\right) + 2\sqrt{T} - 1.$$

□

Chapter 4

Background to Submodular Function Maximization

This chapter presents the background to submodular function maximization. The readers who are familiar with this subject can skip this chapter.

4.1 Submodularity and Related Notions

Let V be a finite ground set and $f : 2^V \to \mathbb{R}$ be a set function. We say f is non-negative if $f(X) \geq 0$ for any $X \subseteq V$, normalized if $f(\emptyset) = 0$, monotone if $X \subseteq Y$ implies $f(X) \leq f(Y)$.

Submodularity of set function f is characterized by the following inequality:

$$f(X) + f(Y) \geq f(X \cup Y) + f(X \cap Y) \qquad \forall X, Y \subseteq V.$$

It is known that f is submodular if and only if f has the following diminishing return property:

$$f(v \mid X) \geq f(v \mid Y) \quad \forall X, Y \subseteq V \text{ such that } X \subseteq Y \text{ and } \forall v \notin Y.$$

Throughout this dissertation, we assume all set functions to be normalized, monotone, and submodular, unless otherwise specified.

As in [Conforti and Cornuéjols, 1984], the curvature $\kappa \in [0, 1]$ of monotone set function f is defined as

$$\kappa := 1 - \min_{v \in V} \frac{f(v \mid V \backslash \{v\})}{f(v)}.$$

A monotone submodular function with $\kappa = 0$ is modular, i.e., it can be written as $f(X) = \sum_{v \in X} w_v$ with some $w_v \geq 0$ ($v \in V$) for any $X \subseteq V$. Intuitively, the curvature is a measure that quantifies the deviation from being modular.

4.2 Literature Overview on Submodular Function Maximization

Given $\mathcal{C} \subseteq 2^V$ and normalized monotone submodular function $f : 2^V \to \mathbb{R}$, monotone submodular function maximization problems are formulated as

$$\underset{X \subseteq V}{\text{maximize}} \quad f(X) \qquad\qquad \text{subject to} \quad X \in \mathcal{C}.$$

Submodular function maximization problems of this form appear in many real-world scenarios: sensor placement [Krause *et al.*, 2008a,b], feature selection [Ko *et al.*, 1995], and document summarization [Lin and Bilmes, 2010]). Given an optimal solution, denoted by S^*, to the problem, we say an algorithm achieves an α-approximation guarantee if it is guaranteed to return solution S satisfying $f(S) \geq \alpha f(S^*)$; we sometimes say such a solution to be α-optimal.

The most well-known and easiest class of monotone submodular function maximization is the following problem with a cardinality constraint:

$$\underset{X \subseteq V}{\text{maximize}} \quad f(X) \qquad\qquad \text{subject to} \quad |X| \leq k,$$

where $k \in [|V|]$. Nemhauser *et al.* [1978] proved that the greedy algorithm, which, starting from $S = \emptyset$, iteratively adds $\operatorname{argmax}_{v \in V \setminus S} f(v \mid S)$ to S, can achieve a $(1 - 1/e)$-approximation guarantee. Nemhauser and Wolsey [1978] proved that no polynomial time algorithms that evaluate f values only polynomially many times can improve the approximation ratio in the value oracle model (i.e., f is given as a black-box oracle). Feige [1998] proved that the NP-hardness of a special case, called the max-k-cover problem. On the other hand, if the curvature κ of f is bounded by a constant smaller than 1, we can improve the approximation ratio. Conforti and Cornuéjols [1984] proved that the greedy algorithm achieves a $(1 - e^{-\kappa})/\kappa$-approximation guarantee, and Sviridenko *et al.* [2015] improved the result by proving that a $(1 - \kappa/e)$-approximation guarantee can be achieved by an algorithm based on the continuous greedy algorithm [Calinescu *et al.*, 2011]; this result holds for a more general case as will be mentioned later.

Submodular function maximization with a knapsack constraint, or the submodular knapsack problem, has also been extensively studied. Given a cost value $c_v \geq 0$ for each $v \in V$ and a budget value B, the problem is formulated as

$$\underset{X \subseteq V}{\text{maximize}} \quad f(X) \qquad\qquad \text{subject to} \quad \sum_{v \in X} c_v \leq B.$$

For a special case called the budgeted maximum coverage problem, Khuller *et al.* [1999] proved that the greedy algorithm achieves a $(1-1/\sqrt{e})$-approximation, and this result was extended to the case of general normalized monotone submodular functions by Lin and Bilmes [2010]. Khuller *et al.* [1999] also proved that a $(1-1/e)$-approximation guarantee for the budgeted maximum coverage problem can be achieved by a greedy-style algorithm that requires $O(|V|^5)$ times evaluations of f. Sviridenko [2004] proved that the same guarantee can be achieved for general normalized monotone submodular functions. Recently, Yoshida [2019] proved that a $(1-\kappa/e)$-approximation guarantee can be achieved by using the continuous greedy algorithm.

Another class of problems that have has been widely studied is submodular function maximization with a matroid constraint; we present the details of matroids in Chapter 7. For this case, the greedy algorithm achieves a 1/2-approximation guarantee [Fisher *et al.*, 1978]; if the curvature is bounded, the approximation ratio can be improved to $\frac{1}{1+\kappa}$ [Conforti and Cornuéjols, 1984]. Calinescu *et al.* [2011] proposed the continuous greedy algorithm, which can achieve a $(1-1/e)$-approximation guarantee. As mentioned above, Sviridenko *et al.* [2015] improved this result by presenting a $(1-\kappa/e)$-approximation algorithm, and they also proved that no polynomial-time algorithms can improve this result in the value oracle model.

Chapter 5

Submodular Function Maximization over Graphs with ZDDs

In this chapter, we consider monotone submodular function maximization problems whose constraints are specified with some graph structures. Although most previous studies have addressed submodular function maximization with cardinality, knapsack, or matroid constraints, more complex constraints often appear in practice. The problem class considered here includes such complex constraints (e.g., an s–t path constraint on a given graph). We propose a novel algorithm that takes advantage of ZDDs, which we use to store all feasible solutions compactly, thus enabling us to circumvent the difficulty of checking feasibility of given solutions. Theoretically, our algorithm is guaranteed to achieve a $(1 - \kappa)$-approximation guarantee, where κ is the curvature of submodular functions. Experiments with various instances show that our algorithm runs much faster than existing exact algorithms and finds better solutions than those obtained by an existing approximation algorithm. Notably, as regards all instances for which optimal solutions were obtained with the exact algorithm, we confirmed that our algorithm achieved better than a 90%-approximation. We finally consider extending set-function maximization over graphs to the case where objective functions are only approximately submodular.

5.1 Introduction

Submodular function maximization problems with some constraints, including cardinality, knapsack, and matroid constraints, are know to be approximately solved with greedy-style algorithms as in Chapter 4. However, it has remained hard to design efficient approximation algorithms for submodular function maximization with more complex constraints. A typical such problem is the *submodular orienteering problem* (SOP) [Chekuri and Pal, 2005]:

SOP. Given budget $B > 0$ and a graph with edge costs and two specified vertices s, t, we seek to find an s–t path whose cost is at most B to maximize a submodular objective function of the set of vertices visited by the s–t path. SOP can model various practical problems such as path-planning problems with robotic sensors [Singh *et al.*, 2009], travel planning [Zeng *et al.*, 2015], and door-to-door marketing [Zhang and Vorobeychik, 2016].

Submodular function maximization with complex constraints also arises in the context of wireless sensor network (WSN) design. One such example is a problem considered in [Krause *et al.*, 2006]: Given a graph that represents the connectability of sensor nodes, we seek a connected tree to maximize a submodular function that measures the informativeness of the obtained tree. Each edge in the graph has a cost, which represents the energy consumed when the two sensors placed on its ends communicate with each other, and the total cost of the obtained tree must be less than or equal to a given budget. In realistic WSN design problems, however, more complex constraints often appear. Below we list two such examples, which we call the *submodular Steiner tree problem* (SSTP) and the *submodular vertex-connected network problem* (SVCNP):

SSTP. As in [Ke *et al.*, 2011], many sensor placement problems have some critical areas that we are particularly interested in; a vertex placed in a critical area is called a *terminal*. In SSTP, we seek a *Steiner tree* that covers all terminals under the budget constraint to maximize an objective function.

SVCNP. We aim to obtain a robust WSN that has no *cut vertex*, which is a vertex whose removal breaks the connectivity of the network; we call a network with no cut vertex *vertex-connected* (VC) network. Since the failure

of a sensor placed on a cut vertex blocks data transmission, how to design a VC network has been an important research subject [Liu *et al.*, 2006; Xiong and Li, 2010]. In SVCNP, we seek a VC network under the budget constraint to maximize an objective function.

The difficulty of the above problems is that the constraints are quite hard to deal with. Specifically, given a subset of vertices, checking the feasibility of the subset in SOP and SSTP reduces to NP-complete problems: the *Hamilton path problem* and *Steiner tree problem*, respectively. The feasibility checking in SVCNP is known to be as hard as the *Hamilton cycle problem* if the edge costs are uniform, and otherwise it becomes more difficult, for which no existing algorithm has been developed; SVCNP is empirically more challenging than SOP and SSTP due to the difficult feasibility checking. If we apply a naive iterative algorithm, e.g., the greedy algorithm, to those problems, we need to solve an NP-complete problem every time a solution is updated. Therefore, those problems are currently awaiting a novel approach that can circumvent the difficulty of feasibility checking.

5.1.1 Our Contribution

We propose a novel approximation algorithm for submodular function maximization with graph-based constraints, which include many hard problems such as SOP, SSTP, and SVCNP. Formally, given graph $G = (V, E)$, where V and E are the vertex set and edge set, respectively, and normalized monotone submodular function $f : 2^V \to \mathbb{R}_{\geq}$, we consider the following problem:

$$\underset{X \subseteq E}{\text{maximize}} \;\; f(V(X)) \quad \text{subject to} \;\; X \in \mathcal{C}, \tag{5.1}$$

where $\mathcal{C} \subseteq 2^E$ is a feasible set that consists of edge subsets with certain substructures, and

$$V(X) \coloneqq \{v \in V \mid (u, v) \text{ or } (v, u) \in X \text{ for some } u \in V\}$$

is the set of ends of all $e \in X$. Problem (5.1) can express various constrained submodular function maximization problems. For example, an SOP can be written as problem (5.1) such that $\mathcal{C}$ is the set of all s–t paths whose cost is at most $B > 0$.

The constraint in problem (5.1) is generally quite complicated, and so we must devise a way to handle the constraint if we are to develop a truly

efficient algorithm. To do this, our algorithm takes advantage of ZDDs; more precisely, we use ZDDs with vertex indices (see, Section 2.4). By storing all $X \in \mathcal{C}$ with a ZDD, problem (5.1) can be reduced to submodular function maximization on a DAG, by which we circumvent the difficulty of checking feasibility for the rigid constraints. ZDD size is generally exponential in $|V|$, and so is the complexity of our algorithm. Fortunately, however, ZDDs that appear in practice are often of tractable size as confirmed in previous studies (e.g., [Minato, 1993]) and our experiments.

Our algorithm is proved to achieve a $(1 - \kappa)$-approximation guarantee, where $\kappa \in [0, 1]$ is the curvature of the submodular function [Conforti and Cornuéjols, 1984]. The curvature value is bounded from above for many practical submodular functions [Sharma *et al.*, 2015; Maehara *et al.*, 2017]; in such cases the approximation ratio, $1 - \kappa$, of our algorithm is also bounded.

In the experiments, we apply our algorithm to three sensing tasks formulated as SOP, SSTP, and SVCNP, respectively; notably, to the best of our knowledge, our algorithm is the first practical approximation algorithm that is applicable to SVCNP. We show that our algorithm achieves at least a 90%-approximation in all instances for which optimal values are available, outperforming an existing approximation algorithm in most cases. Our algorithm is also shown to be much faster than exact algorithms.

5.1.2 Related Work

SOP has been extensively studied because of its applicability to various problems [Singh *et al.*, 2009; Zeng *et al.*, 2015; Zhang and Vorobeychik, 2016]. SOP was proposed by Chekuri and Pal [2005] as an extension of the *orienteering problem*, a classical combinatorial optimization problem. For SOP, they proposed a quasi-polynomial time approximation algorithm, called the *recursive greedy algorithm*. Singh *et al.* [2009] scaled up the recursive greedy algorithm and applied it to path-planning problems for robot-based environment sensing. Recently, a faster bi-criterion approximation algorithm for SOP called the *generalized cost-benefit greedy algorithm* (GCB) was proposed by Zhang and Vorobeychik [2016]; it uses the cost-benefit greedy strategy, where the cost values are computed approximately. A special case of SOP was considered by Zeng *et al.* [2015]; they studied the travel route search problem of maximizing the satisfaction of a traveler's preference, which is an SOP with a *keyword coverage* (KC) objective function. They proposed an A* algorithm with a heuristic function that is specialized for the KC function.

While their algorithm is exact (i.e., it always finds an optimal solution), its time complexity is generally exponential in $|E|$.

Submodular function maximization that has budget and tree constraints is studied in [Krause *et al.*, 2006], which is an important problem in WSN design. For this problem they proposed an efficient approximation algorithm called *pSPIEL*. Their problem formulation is, however, sometimes insufficient for modeling realistic settings. Below we present two such examples and previous studies on them.

In many real-world WSN design problems, our interest is rarely spread over the whole target areas; rather we have some critical areas that are of particular interest. Such a situation is modeled as a network design problem with a Steiner tree constraint [Ke *et al.*, 2011]. However, they did not consider the submodularity of objective functions, which is known to be essential in sensor placement problems [Krause *et al.*, 2008b]. Taking the above into account, we consider SSTP, which is an submodular function maximization problem with budget and Steiner tree constraints. This problem is very hard due to its rigid constraints, and thus no algorithms for this problem have been studied.

SVCNP is also a variant of the problem considered in [Krause *et al.*, 2006]. In many real-world scenarios, it is important to obtain a robust WSN, and thus finding VC networks has been an attractive research subject [Liu *et al.*, 2006; Xiong and Li, 2010]. Although many existing approximation algorithms consider computing the smallest VC network on graphs with uniform edge costs [Garg *et al.*, 1993; Heeger and Vygen, 2017], no algorithm considers non-uniform edge costs, which frequently appear in practice. This implies no existing algorithm can check the feasibility of a given vertex subset efficiently in SVCNPs, and so we are currently missing practical algorithms for SVCNPs.

5.2 ZDD-based Algorithm

This section presents the approximation algorithm for problem (5.1). We briefly review ZDDs used here, and then we describe the algorithm that searches for an approximate solution on the ZDDs.

Algorithm 8 GreedyDP($\mathtt{Z}_{\mathcal{C}}, f$)

$\mathtt{U} = \mathtt{N} \backslash \{\mathtt{r}\}$.
Create a topological order of all $\mathtt{n} \in \mathtt{U}$.
$\mathtt{R}_{\mathtt{r},\mathtt{n}} = \emptyset$ for all $\mathtt{n} \in \mathtt{N}$.
while $\mathtt{U} \neq \emptyset$ **do**
    Let $\mathtt{n} \in \mathtt{U}$ be the first node in the topological order.
    $(\mathtt{m}^*, \mathtt{n}) = \underset{(\mathtt{m},\mathtt{n}) \in \mathtt{A}_\mathtt{n}}{\operatorname{argmax}} f(\mathtt{R}_{\mathtt{r},\mathtt{m}} \cup (\mathtt{m}, \mathtt{n}))$.
    $\mathtt{R}_{\mathtt{r},\mathtt{n}} = \mathtt{R}_{\mathtt{r},\mathtt{m}^*} \cup (\mathtt{m}^*, \mathtt{n})$.
    $\mathtt{U} = \mathtt{U} \backslash \{\mathtt{n}\}$.
end while
return $\mathtt{R}_{\mathtt{r},1}$.

5.2.1 ZDDs for Submodular Maximization over Graphs

Given $\mathcal{C} \subseteq 2^E$, we construct ZDD $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ with vertex indices that stores all $X \in \mathcal{C}$. As in Section 2.4, we define $I \coloneqq E \cup V$ As the set of all edges and vertices of G; all the elements in I are ordered appropriately by using the method shown in [Kawahara *et al.*, 2017b]. We define

$$E_{\mathtt{R}} \coloneqq \{l_{\mathtt{n}} \in E \mid \mathtt{a}^1_{\mathtt{n}} \in \mathtt{R}\} \qquad V_{\mathtt{R}} \coloneqq \{l_{\mathtt{n}} \in V \mid \mathtt{a}^1_{\mathtt{n}} \in \mathtt{R}\}.$$

Then $\mathtt{Z}_{\mathcal{C}}$ stores $\mathcal{C}$ as $\{E_{\mathtt{R}} \subseteq E \mid \mathtt{R} \in \mathcal{R}_{\mathtt{r},1}\}$. Furthermore, we always have $V_{\mathtt{R}} = V(E_{\mathtt{R}})$. For later use, we define $\mathcal{R}$ as a collection of all routes in $\mathtt{Z}_{\mathcal{C}}$; i.e.,

$$\mathcal{R} \coloneqq \{\mathtt{R} \in \mathcal{R}_{\mathtt{m},\mathtt{n}} \mid \exists \mathtt{m}, \mathtt{n} \in \mathtt{N} \text{ and } l_{\mathtt{m}} < l_{\mathtt{n}}\}.$$

Empirically, it is often the case that ZDD size $|\mathtt{N}|$ is much smaller than $|\mathcal{C}|$, thus making our algorithm efficient as described later. For example, as shown in Section 5.3.1, about 6.0×10^{16} feasible solutions of an SOP are store with a ZDD that has about 3.0×10^7 nodes, which allows our algorithm to find an approximate solution within 4.0×10^3 seconds.

5.2.2 GreedyDP

We present a ZDD-based $(1 - \kappa)$-approximation algorithm for problem (5.1). Let $\mathtt{Z}_{\mathcal{C}} = (\mathtt{N}, \mathtt{A})$ be a ZDD that stores $\mathcal{C}$. For convenience, we define $f(\mathtt{R}) \coloneqq f(V_{\mathtt{R}})$ for any $\mathtt{R} \in \mathcal{R}$. Thanks to the properties of ZDDs, problem (5.1) can

be written as the following submodular function maximization on $\mathtt{Z}_{\mathcal{C}}$:

$$\underset{\mathtt{R}\subseteq\mathtt{A}}{\text{maximize}}\ f(\mathtt{R}) \quad \text{subject to } \mathtt{R}\in\mathcal{R}_{\mathtt{r},1}. \tag{5.2}$$

Note that, if we obtain a $(1-\kappa)$-approximate solution $\mathtt{R}\in\mathcal{R}_{\mathtt{r},1}$ for problem (5.2), then $E_{\mathtt{R}}\subseteq E$ is a $(1-\kappa)$-approximate solution for problem (5.1). Below we discuss how to find an approximate solution for problem (5.2).

For every $\mathtt{n}\in\mathtt{N}\backslash\{\mathtt{r}\}$, we let $\mathrm{Parents}(\mathtt{n})\subseteq\mathtt{N}$ be the set of all parents of $\mathtt{n}$, and we define $\mathtt{A}_{\mathtt{n}}\coloneqq\{(\mathtt{m},\mathtt{n})\in\mathtt{A}\mid \mathtt{m}\in\mathrm{Parents}(\mathtt{n})\}$. Our algorithm is described in Algorithm 8, which finds a $(1-\kappa)$-approximate solution $\mathtt{R}\in\mathcal{R}_{\mathtt{r},1}$ for problem (5.2). We first creates a topological order for $\mathtt{U}=\mathtt{N}\backslash\{\mathtt{r}\}$ so that, for all $(\mathtt{m},\mathtt{n})\in\mathtt{A}$, $\mathtt{m}$ comes before $\mathtt{n}$. We then finds a route $\mathtt{R}_{\mathtt{r},\mathtt{n}}$ for each $\mathtt{n}\in\mathtt{N}\backslash\{\mathtt{r}\}$ in the topological order. At each iteration, $\mathtt{U}\subseteq\mathtt{N}\backslash\{\mathtt{r}\}$ represents the set of nodes that have not been visited yet. Thanks to the topological sorting, we have $\mathrm{Parents}(\mathtt{n})\subseteq\mathtt{N}\backslash\mathtt{U}$ for $\mathtt{n}\in\mathtt{N}$ chosen in Step 5. The search strategy of Algorithm 8 is based on the well-known dynamic programming (DP) approach that is used for finding the longest path on a DAG (see, e.g., [Sedgewick and Wayne, 2011]). On the other hand, Algorithm 8 is greedy in that, once we obtain a route, we do not change it, and a new route $\mathtt{R}_{\mathtt{r},\mathtt{n}}$ is obtained by the greedy rule over $\mathtt{A}_{\mathtt{n}}$ as in Steps 6 and 7. Thus we call it *GreedyDP*. We can easily check that Algorithm 8 requires at most $O(|\mathtt{N}|)$ function value evaluations. The following theorem provides the approximation guarantee for GreedyDP.

Theorem 5. *If $\mathtt{R}^*_{\mathtt{r},1}$ is an optimal route, GreedyDP finds a route $\mathtt{R}_{\mathtt{r},1}$ satisfying $f(\mathtt{R}_{\mathtt{r},1})\geq(1-\kappa)f(\mathtt{R}^*_{\mathtt{r},1})$.*

Proof. From the definition of curvature κ, we have the following inequality for any $T\subseteq V$ and $v\in V\backslash T$ if $f(v)>0$ holds:

$$1-\kappa=\min_{S\subsetneq V,\ v'\notin S}\frac{f(v'\mid S)}{f(v')}\leq\frac{f(v\mid T)}{f(v)},$$

which means $(1-\kappa)f(v)\leq f(v\mid T)$. If $f(v)=0$, then we have $f(v\mid T)=0$ by the submodularity. Thus $(1-\kappa)f(v)\leq f(v\mid T)$ holds in both cases. Since $f(v\mid S)\leq f(v)$ holds for any $S\subseteq V$ due to the submodularity, the following inequality holds for any $v\in V$ and $S,T\subseteq V$ satisfying $v\notin T$:

$$f(v\mid T)\geq(1-\kappa)f(v\mid S). \tag{5.3}$$

With this inequality we prove Theorem 5. Let

$$\mathtt{R}^*_{\mathtt{r},\mathtt{n}} := \arg\max_{\mathtt{R}\in\mathcal{R}_{\mathtt{r},\mathtt{n}}} f(\mathtt{R})$$

for all $\mathtt{n} \in \mathtt{N}$. The proof is obtained by induction on the size of $\mathtt{U}$, i.e., assuming

$$f(\mathtt{R}_{\mathtt{r},\mathtt{m}}) \geq (1-\kappa) f(\mathtt{R}^*_{\mathtt{r},\mathtt{m}}) \qquad \forall \mathtt{m} \in \mathtt{N}\backslash\mathtt{U}, \tag{5.4}$$

where $\mathtt{R}_{\mathtt{r},\mathtt{m}}$ is the route constructed with Algorithm 8, we prove $f(\mathtt{R}_{\mathtt{r},\mathtt{n}}) \geq (1-\kappa) f(\mathtt{R}^*_{\mathtt{r},\mathtt{n}})$ for node $\mathtt{n} \in \mathtt{U}$. Note that (5.4) holds if $\mathtt{U} = \mathtt{N}\backslash\{\mathtt{r}\}$.

Let $\mathtt{n} \in \mathtt{U}$ be a node chosen in Step 5 of Algorithm 8 and $\mathtt{m} \in \mathrm{Parents}(\mathtt{n})$ satisfy $(\mathtt{m},\mathtt{n}) \in \mathtt{R}^*_{\mathtt{r},\mathtt{n}}$, i.e., $\mathtt{m}$ is the tail of the last arc in $\mathtt{R}^*_{\mathtt{r},\mathtt{n}}$. Furthermore, let $(\mathtt{m}^*,\mathtt{n}) \in \mathtt{A}_{\mathtt{n}}\mathtt{n}$ be an arc chosen in Step 6. Note that $(\mathtt{m}^*,\mathtt{n})$ is chosen greedily and thus satisfies the following inequality:

$$f(\mathtt{R}_{\mathtt{r},\mathtt{n}}) = f(\mathtt{R}_{\mathtt{r},\mathtt{m}^*}) + f((\mathtt{m}^*,\mathtt{n}) \mid \mathtt{R}_{\mathtt{r},\mathtt{m}^*}) \geq f(\mathtt{R}_{\mathtt{r},\mathtt{m}}) + f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}_{\mathtt{r},\mathtt{m}}). \tag{5.5}$$

Since (5.4) holds for $\mathtt{m} \in \mathrm{Parents}(\mathtt{n}) \subseteq \mathtt{N}\backslash\mathtt{U}$ and $\mathtt{R}^*_{\mathtt{r},\mathtt{m}} \in \mathcal{R}_{\mathtt{r},\mathtt{m}}$ is an optimal route from $\mathtt{r}$ to $\mathtt{m}$, we get

$$f(\mathtt{R}_{\mathtt{r},\mathtt{m}}) \geq (1-\kappa) f(\mathtt{R}^*_{\mathtt{r},\mathtt{m}}) \geq (1-\kappa) f(\mathtt{R}^*_{\mathtt{r},\mathtt{n}}\backslash(\mathtt{m},\mathtt{n})). \tag{5.6}$$

Now we consider two cases as follows. If $(\mathtt{m},\mathtt{n})$ is 0-arc or $l_{\mathtt{m}} \in E$, then adding $(\mathtt{m},\mathtt{n})$ to a route $\mathtt{R} \in \mathcal{R}_{\mathtt{r},\mathtt{m}}$ does not increase the value of f. Thus we have

$$f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}_{\mathtt{r},\mathtt{m}}) = f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}^*_{\mathtt{r},\mathtt{n}}\backslash(\mathtt{m},\mathtt{n})) = 0.$$

If $(\mathtt{m},\mathtt{n})$ is 1-arc and $l_{\mathtt{m}} \in V$, then $l_{\mathtt{m}} \in V$ is added to the argument of $f(\cdot)$ by choosing $(\mathtt{m},\mathtt{n})$. Since $l_{\mathtt{m}}$ appears at most once in $\mathtt{R}_{\mathtt{r},\mathtt{m}} \cup (\mathtt{m},\mathtt{n})$ due to the property of ZDDs, we have $l_{\mathtt{m}} \notin V_{\mathtt{R}_{\mathtt{r},\mathtt{m}}}$. Hence we obtain the following inequality by (5.3):

$$f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}_{\mathtt{r},\mathtt{m}}) \geq (1-\kappa) f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}^*_{\mathtt{r},\mathtt{n}}\backslash(\mathtt{m},\mathtt{n})).$$

Therefore, in both cases, we have

$$f((\mathtt{m},\mathtt{n} \mid \mathtt{R}_{\mathtt{r},\mathtt{m}}) \geq (1-\kappa) f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}^*_{\mathtt{r},\mathtt{n}}\backslash(\mathtt{m},\mathtt{n})). \tag{5.7}$$

Substituting (5.6) and (5.7) into (5.5), we obtain

$$\begin{aligned} f(\mathtt{R}_{\mathtt{r},\mathtt{n}}) &\geq (1-\kappa)\left(f(\mathtt{R}^*_{\mathtt{r},\mathtt{n}}\backslash(\mathtt{m},\mathtt{n})) + f((\mathtt{m},\mathtt{n}) \mid \mathtt{R}^*_{\mathtt{r},\mathtt{n}}\backslash(\mathtt{m},\mathtt{n}))\right) \\ &= (1-\kappa) f(\mathtt{R}^*_{\mathtt{r},\mathtt{n}}), \end{aligned}$$

where $\mathtt{n} \in \mathtt{U}$. By induction on the size of $\mathtt{U}$, we obtain (5.4) for $U = \emptyset$, and thus $f(\mathtt{R}_{\mathtt{r},1}) \geq (1-\kappa) f(\mathtt{R}^*_{\mathtt{r},1})$ follows. □

Note that, once we have constructed $\mathtt{Z}_{\mathcal{C}}$ for a feasible set $\mathcal{C}$, it can be reused when maximizing any objective functions in the same $\mathcal{C}$ using GreedyDP. This makes our algorithm efficient particularly when solving submodular function maximization problems repeatedly in the same feasible set; for example, our algorithm is particularly efficient when solving path planning problems for a robotic sensor that must perform sensing on a daily basis or more frequently in the same target space.

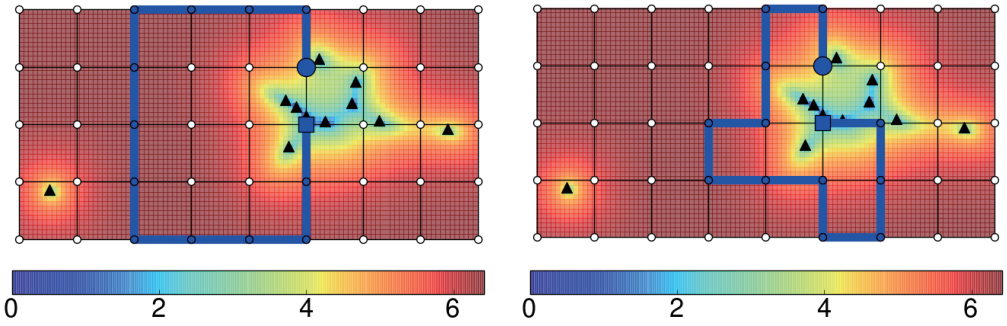
5.3 Experiments

We conduct experiments on three sensing tasks to assess the performance of our algorithm. One is the path planning problem for a robotic sensor, which can be written as an SOP, and the others are indoor WSN design problems, which can be formulated as an SSTP and SVCNP. All experiments were performed on a computer equipped with 128 GB RAM and Xeon E5 3.1 GHz CPU. Algorithm 8 and the alternative algorithms that we used for comparison were implemented in Python. We used the C++ library [Suzuki, 2016] to construct the ZDDs in our algorithm. Throughout the experiments, the limit of execution time is 10^4 seconds.

5.3.1 Path Planning for a Mobile Robotic Sensor

We consider the following path planning problem as considered in [Singh *et al.*, 2009; Zhang and Vorobeychik, 2016]. As shown in Figure 5.1, the 2D target space is discretized by the grid graph $G = (V, E)$. We seek a path in G that visits a subset of vertices optimally so that a mobile robot equipped with sensors can collect as much information as possible. Using a grid graph is a natural approach when obstacles, such as buildings, are placed regularly in the target space. We discretize the target space by using 5×9 and 7×13 grids. For simplicity, we suppose that the cost of traversing one edge is always 1, and the obtained path $X \subseteq E$ must satisfy the budget constraint $|X| \leq B$ for some $B > 0$. To assess the scalability of algorithms, we consider all feasible budget values: $B = 1, 3, \ldots, 43$ for the 5×9 grid and $B = 1, 3, \ldots, 89$ for the 7×13 grid, where $B = 43$ and $B = 89$ are the length of the longest s–t paths in the 5×9 and 7×13 grids, respectively.

This experiment uses air quality sensor data monitored at 11 stations in Shanghai, China [Zheng *et al.*, 2013], and we focus on PM2.5 data. Given the



(a) Our algorithm (b) GCB

Figure 5.1: s–t paths with $B = 13$ obtained by our algorithm and GCB, respectively. The vertices $s, t \in V$ are indicated by the blue filled circle and square, respectively. The 11 filled black triangles indicate the locations of the 11 stations. The heat map expresses the informativeness of locations before performing sensing with the robot; locations with a warmer (darker) color are more informative.

data obtained at the 11 stations, we consider collecting additional information using a mobile robot equipped with sensors. The robot must start from and return to specified locations.

To measure the informativeness of obtained paths, we employ the entropy function of the selected vertices, which is known to be a monotone submodular function [Krause *et al.*, 2008b]. Let V_0 be the set of locations of the 11 stations. Starting from $S = V_0$, if S is the current set of vertices that have already been visited, the marginal gain of visiting new vertex $v \notin V \backslash S$ is expressed by the conditional entropy

$$H(v \mid S) \coloneqq \frac{1}{2} \log(2\pi e \sigma^2_{v|S}),$$

where $\sigma^2_{v|S} \coloneqq K(v,v) - K(v,S)K(S,S)^{-1}K(S,v)$ is the conditional covariance. The matrices $K(S,T) \in \mathbb{R}^{|S| \times |T|}$ are defined by a kernel function; we here use an ordinary Gaussian kernel (see, e.g., [Bishop, 2006]). The parameters of $K(\cdot,\cdot)$ are fitted to the data obtained at the 11 stations.

In this experiment we use the following two algorithms to benchmark our algorithm:

CBG. A generalized cost-benefit greedy algorithm [Zhang and Vorobeychik, 2016]. Given current solution $S \subseteq V$, GCB considers adding new vertex $v \in V \backslash S$ sequentially in non-increasing order of the cost-benefit ratios; to do this we need an approximate cost of S, which we compute here using the nearest neighbor algorithm as in the original paper. We note that this method does not always find a simple s–t path as a solution; namely, it sometimes finds a walk that visits the same vertices twice or more.

A* algorithm. An exact A* algorithm based on [Zeng *et al.*, 2015]. Since heuristic function $h(\cdot)$ used in the original A* algorithm is designed for the KC function, it is not applicable to our setting. Thus we considered two alternatives of $h(\cdot)$. Given current vertex $v \in V$ and solution $X \subseteq E$, both heuristic function values $h(X)$ bound

$$\max_{S \subseteq W : |S| \leq B - |X|} f(S \mid V(X)) \tag{5.8}$$

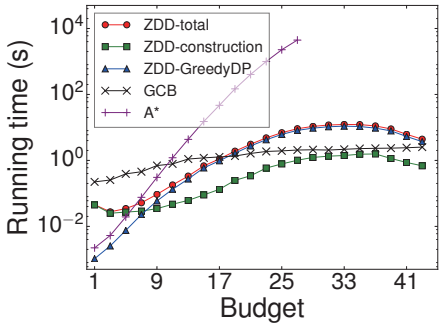
from above, where W is a set of all vertices that are reachable from v with at most $B - |X|$ cost; the A* algorithm is exact if $h(X)$ is always larger than or equal to (5.8). The first one is a natural generalization of the one used in [Zeng *et al.*, 2015]; letting $h'(X)$ be the objective value achieved by the greedy algorithm, which achieves a $(1 - 1/e)$-approximation for problem (5.8), we set

$$h(X) = h'(X)/(1 - 1/e).$$

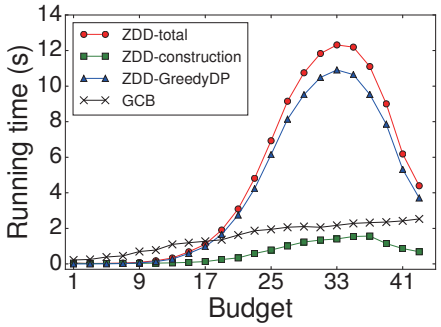
The second one is from [Chen *et al.*, 2015]; we let

$$h(X) = \max_{S \subseteq W : |S| \leq B - |X|} \sum_{v \in S} f(v \mid V(X)),$$

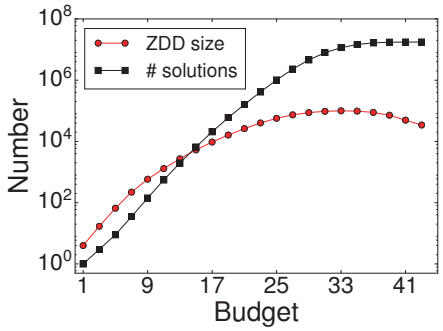
which is at least as large as (5.8) thanks to the submodularity. We experimentally observed that A* search with the second $h(\cdot)$ is much faster, and thus we employed it.



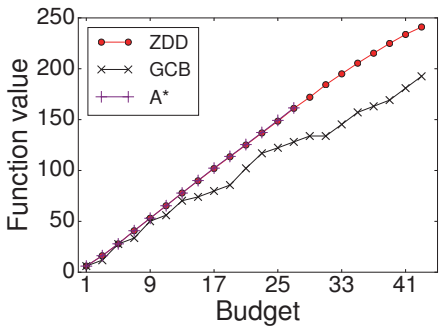
(a) Running times (semi-log)



(b) Running times



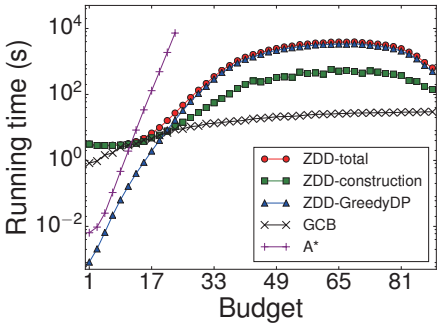
(c) ZDD size and # solutions



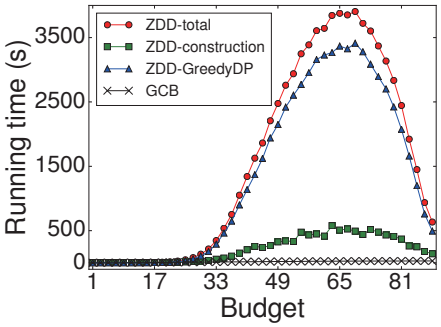
(d) Objective values

Figure 5.2: Results on SOPs with a 5×9 grid. (a) Semi-log plot of running times for our algorithm, GCB and the A* algorithm. The running times of our algorithm are shown for constructing a ZDD (ZDD-construction), executing GreedyDP (ZDD-GreedyDP) and their summation (ZDD-total). (b) Running times of our algorithm and GCB. (c) Semi-log plot of the ZDD size $|\mathtt{N}|$ and the number of feasible solutions $|\mathcal{C}|$. (d) Objective values achieved by our algorithm, GCB and the A* algorithm; those of the A* algorithm are always optimal.

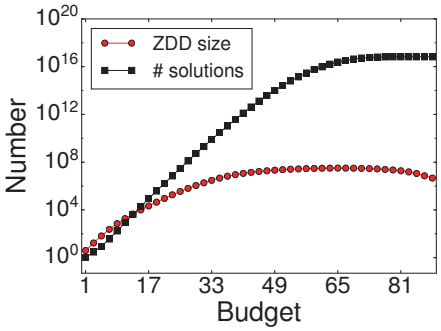
We also tried to apply an algorithm based on the recursive greedy algorithm [Singh *et al.*, 2009] to the problem. However, since the problem is not a Euclidean SOP, most of the techniques employed to accelerate the recursive greedy are not applicable. As a result, the approximation algorithm based on the recursive greedy took far longer than the exact A* algorithm; actually it took more than one day to find a solution for an SOP with the 5×9 grid and $B = 9$. Thus we omit comparison with the recursive greedy. We note that, even if the techniques for acceleration can be used, the recursive greedy



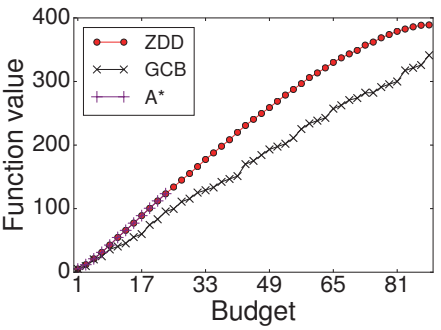
(a) Running times (semi-log)



(b) Running times



(c) ZDD size and # solutions



(d) Objective values

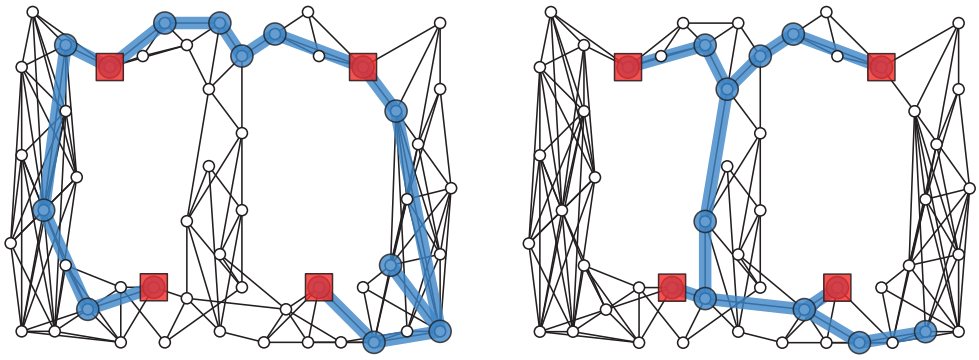
Figure 5.3: Results on SOPs with a 7×13 grid. (a)–(d) correspond to those of Figure 5.2.

becomes at most three orders of magnitude faster (see, [Singh *et al.*, 2009]). Thus it is still slower than our algorithm, which took only about 10^{-1} seconds for the SOP with the 5×9 grid and $B = 9$.

Figures 5.2 and 5.3 summarize the results of the 5×9 and 7×13 grids, respectively. We first consider the computation cost of the algorithms. Figures 5.2a and 5.3a show running times of our algorithm, GCB and the A* algorithm; the results of the A* algorithm with $B > 27$ and $B > 23$ are omitted in Figures 5.2a and 5.3a, respectively, since the time limit was exceeded. We see that our algorithm and GCB are much more scalable than the A* algorithm; our algorithm is about 5×10^2 times faster than the A* algorithm for the SOP on the 5×9 grid with $B = 27$. The detailed running time comparison of our algorithm and GCB is shown in Figures 5.2b and 5.3b. Although GCB is more scalable than our algorithm, both have reasonable

computation costs relative to those of the other algorithms. As examined in [Minato, 1993], ZDD size does not always increase with the value of B (Figures 5.2c and 5.3c), and neither does the running time of our algorithm. Figures 5.2c and 5.3c show that, in many instances, ZDD size is smaller than the number of feasible solutions by orders of magnitude, which makes our algorithm far more scalable than exact algorithms.

We now turn to the quality of solutions obtained by the algorithms. The objective values achieved the three algorithms are shown in Figures 5.2d and 5.3d, where those computed by the A* algorithm for $B \leq 27$ and $B \leq 23$, respectively, are optimal. Our algorithm significantly outperformed GCB; actually, it always achieved at least a 99%-approximation in the 26 instances to which the A* algorithm was applied, and found optimal solutions in 18 out of the 26 instances. The following two reasons explain why GCB had trouble in achieving high objective values. (1) The first reason is the difficulty of finding a feasible s–t path that covers the current solution of vertices efficiently. Every time GCB finds new solution $S \subseteq V$, we need to evaluate the cost of S by computing the minimum-cost s–t path that visits all vertices in S; this, however, is NP-hard in general. Thus GCB instead evaluates the cost of S inexactly using the nearest neighbor algorithm, which finds an s–t walk that visits all vertices in S. As a result, GCB sometimes returns an s–t walk that visits the same vertices time and again, which leads to the lower objective values. (2) The second reason is the characteristics of the cost-benefit greedy search strategy. To see this, we use Figure 5.1 as an explanatory example. The figure shows the solutions obtained by the two algorithms for the SOP on the 5×9 grid with $B = 13$. Here, our algorithm successfully found an optimal s–t path. GCB also finds an s–t path, but it fails to pass through the informative area (around the 3rd column from the left in the grid graph). Since GCB employs the cost-benefit greedy search strategy, it tends to choose vertices that are not far from s, t in the first a few iterations. Consequently, GCB fails to reach informative areas that are far from s, t. On the other hand, our algorithm seldom suffers these two problems. This is because (1) all feasible s–t paths are stored in a ZDD, and thus, (2) even if the informative areas are far from s, t, our algorithm considers s–t paths going through the areas as candidates.



(a) Our algorithm (b) GCB

Figure 5.4: Illustration of the problem and obtained solutions. The vertices represent the locations where sensors can be placed, and two sensors can communicate if and only if they are placed on vertices connected by an edge. The filled squares indicate terminals. Blue thick edges are the networks obtained with (a) our algorithm and (b) GCB for $B = 50$.

5.3.2 Indoor WSN Design with Critical Areas

We consider an indoor WSN design problem formulated as an SSTP. To collect as much information as possible, we aim to design a network that covers as wide an area as possible under the following four constraints. (1) Sensors must be placed at some specified locations (e.g, locations close to outlets); we denote the set of locations by V. (2) The sensors must form a connected network; sensors that are too far from each other, or even nearby sensors that are obstructed by, for example, walls or radiation from appliances, cannot communicate with each other. (3) The communication cost of the obtained network must be at most B; sensors consume electric power when communicating with each other, and the total power consumption is bounded by B. (4) There are some critical locations, which are called terminals, and a sensor must be placed on every terminal. To model this situation, we construct graph $G = (V, E)$ as in Figure 5.4. A sensor can be placed on a vertex, and two sensors can communicate if and only if they are placed on two vertices connected by an edge. The terminals are indicated by filled squares. We seek to find an optimal Steiner tree in G that includes all terminals.

This experiment uses temperature field data from the Intel Berkeley Research Laboratory [Madden, 2004]. To construct graph G, we use the

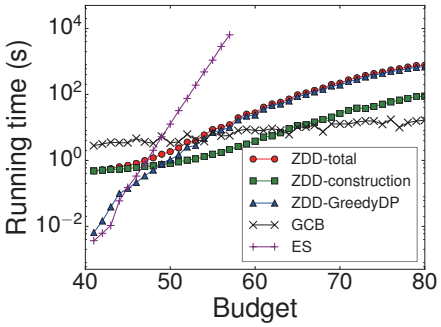
data on connectivity; two vertices are connected if and only if their average probability of successful communication is at least 40%. The resulting graph has 54 vertices and 144 edges. The edge costs are integers ranging from 1 to 5, which are computed from the success probability; the total cost of each obtained network must be at most B. As shown in [Krause *et al.*, 2008b], in the context of indoor sensor placements, mutual information

$$\mathrm{MI}(S) \coloneqq H(V \backslash S) - H(V \backslash S \mid S)$$

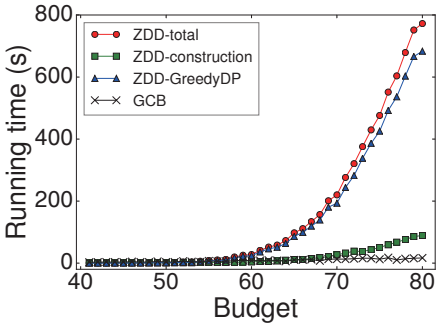
is suitable to measure the informativeness of given $S \subseteq V$, where $H(\cdot)$ is the entropy function used in the SOP experiments, and thus we employ it as an objective function. Unfortunately, the mutual information is non-monotone in general; this seems to be somewhat undesirable as a measure of informativeness, and it does not match the setting of problem (5.1). Fortunately, however, it is known to be approximately monotone if the number of deployed sensors is small enough compared to $|V|$. Thus we here make $B = 41, \ldots, 80$, so the number of deployable sensors is small enough; $B = 41$ is the cost of the minimum Steiner tree covering all terminals.

We implemented and applied our algorithm to the problem; its performance is compared with that of exhaustive search (ES) and a GCB variant that employs the following approximate cost computation. Given current solution $S \subseteq V$, the cost of S is evaluated by constructing a Steiner tree that covers S using the nearest neighbor method. More precisely, starting from $v_{\mathrm{init}} \in S$, if $T \subseteq S$ is a current connected tree, we connect the pair $u \in T$ and $v \in S \backslash T$ that has the minimum shortest-path length; this procedure is continued until T covers all vertices in S. We vary v_{init} among all vertices in S and take a Steiner tree with the smallest cost.

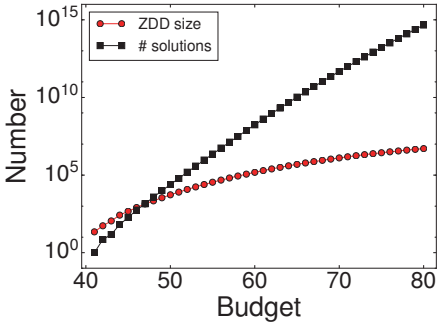
Figure 5.5a shows that our algorithm and GCB are far more scalable than ES; the results of ES for $B \geq 58$ are omitted since the time limit was exceeded. Compared with GCB, our algorithm performs poorly in terms of running time (Figure 5.5b). This, however, is not such a hardship in WSN design since we rarely face situations where we must design a network quickly. Figure 5.5c shows the numbers of feasible solutions $|\mathcal{C}|$ and ZDD sizes $|\mathtt{N}|$. We see that ZDD size grows much more slowly than $|\mathcal{C}|$, and this is why our algorithm is more scalable than ES. Figure 5.5d plots the objective values achieved by the two algorithms, where those obtained by ES for $B \leq 57$ are optimal. We see that our algorithm outperforms GCB in most instances and that our algorithm is almost optimal for $B \leq 57$; actually our algorithm achieved more



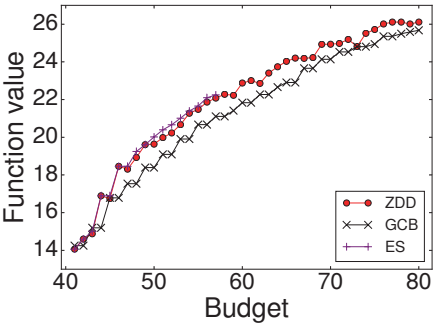
(a) Running times (semi-log)



(b) Running times



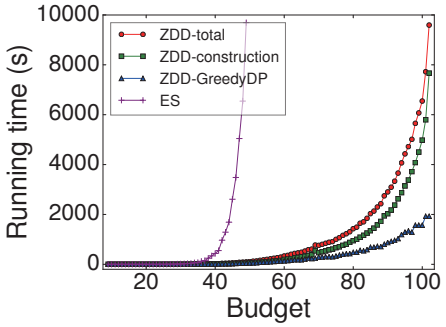
(c) ZDD sizes and # solutions



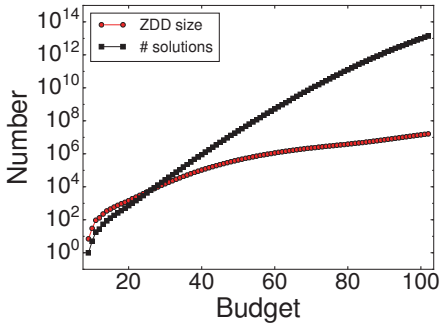
(d) Objective values

Figure 5.5: (a) Semi-log plot of running times of our algorithm, GCB, and ES. For our algorithm, the running time for each procedure is shown as in Figures 5.2 and 5.3. (b) Running times of our algorithm and GCB. (c) Numbers of feasible solutions and ZDD sizes. (d) Function values obtained by the three algorithms; those of ES for $B \leq 57$ are optimal.

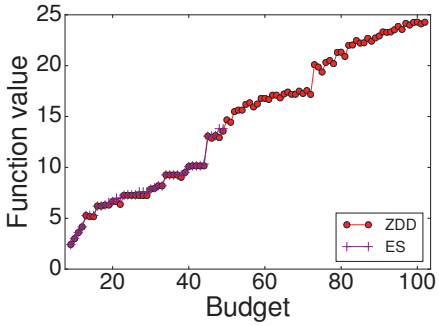
than a 97%-approximation in all 17 instances to which ES was applied, and found optimal solutions in 5 out of the 17 instances. Figures 5.4a and 5.4b illustrate the networks obtained with our algorithm and GCB, respectively, for $B = 50$. The network output by our algorithm covers a wider area than that output by GCB, showing the effectiveness of our algorithm for this problem. Similar to the results of the SOP experiments, GCB again suffers from the inherent characteristics of the cost-benefit greedy strategy. Namely, in the first a few iterations, GCB chooses vertices around the center of the target space, which have large cost-benefit ratios, but choosing such vertices greedily results in a WSN that cannot cover the left and right sides of the target space



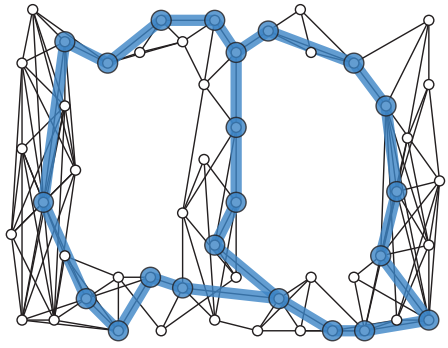
(a) Running times



(b) ZDD sizes and # solutions



(c) Objective values



(d) The graph and a solution

Figure 5.6: (a) Running times of our algorithm and ES. (b) Numbers of feasible solutions and ZDD sizes. (c) Function values attained by our algorithm and ES; those of ES for $B \leq 49$ are optimal. (d) Illustration of the graph and an obtained solution; blue thick edges are the network obtained with our algorithm for $B = 101$.

because of the budget constraint.

5.3.3 Robust Indoor WSN Design

We consider problems raised in finding robust WSNs, which we formulate as SVCNPs. We use the same graph and objective function as those in the SSTP experiments. In the SVCNP instances, we observed that the objective value increases with B even for large B, and thus we set $B = 9, \ldots, 102$; $B = 9$ is the cost of the minimum VC network (at $B = 103$ our algorithm exceeded the time limit).

When it comes to applying our algorithm to SVCNPs, a ZDD storing all VC networks cannot be obtained by the commonly used top-down construction method (see, e.g., [Kawahara *et al.*, 2017a]), and thus we constructed it using algebraic operations defined on a family of sets (see, [Knuth, 2011]), which can be performed on ZDDs. To the best of our knowledge, no existing algorithm is applicable to SVCNP since its constraint is too complex. Thus we employ only ES to benchmark our algorithm.

As in Figure 5.6a, our algorithm is much more scalable than ES by virtue of the fact that the ZDD size grows more slowly than the number of feasible solutions (Figure 5.6b); the running times of ES for $B \geq 50$ are omitted since the time limit was exceeded. Different from the results in SOPs and SSTPs, the ZDD-construction requires larger computation cost than ZDD-GreedyDP. This is because the ZDDs for SVCNPs are constructed using algebraic operations on ZDDs repetitively, which takes a somewhat long time. Figure 5.6c plots the objective values achieved by the two algorithms, where those obtained with ES for $B \leq 49$ are optimal. We observed that our algorithm attained at least a 91%-approximation in all 41 instances to which ES was applied, and found optimal solutions in 24 out of the 41 instances. Figure 5.6d illustrates a solution obtained with our algorithm for $B = 101$, showing that our algorithm successfully found a VC network that covers a wide area.

5.4 On Extension to Weakly Submodular Function Maximization

Set functions that appear in practice sometimes lack submodularity. Some of them, however, have a property that is close to submodularity; this is called weak submodularity [Das and Kempe, 2018]. Formally, monotone set function f is γ-weakly submodular if

$$f(S \mid T) \leq \gamma \sum_{v \in S} f(v \mid T)$$

holds for any disjoint S and T; We have $\gamma = 1$ if f is submodular, and larger $\gamma \leq 1$ implies that f is closer to being submodular. The γ value is know to be lower bounded if f originates from well-conditioned sparse optimization problems [Das and Kempe, 2018; Elenberg *et al.*, 2018].

As in the above sections, given a graph $G = (V, E)$, we consider set function f of vertices, and we assume it is γ-weakly submodular. As in [Hegde *et al.*, 2015], we aim to solve sparse optimization problems where solutions are expected to have a small number of connected components on the graph; we say such solutions to be contiguous. Such problems appear, for example, when we are to detect pollution on a water network. To obtain sparse and contiguous solutions, we use $C(S) = \lambda|S| + N(S)$ as a cost function, where $N(\cdot)$ counts the number of connected components and λ is a hyper-parameter that is set to be larger than the maximum degree in the graph, denoted by δ.

We apply the greedy algorithm to $\max_{C(S)\leq c} f(S)$. By iteratively choosing a vertex that keeps the solution feasible and has the maximum marginal cost-benefit ratio, we can obtain a solution S that satisfies

$$f(S) \geq \left(1 - \exp\left(-\frac{\gamma\lambda(1-\rho/c)}{\lambda+1}\right)\right) f(S^*)$$

for an optimal solution S^*, where ρ denotes the largest marginal gain of the cost function C (see, [Sakaue, 2019] for details). Therefore, for the above constraint that induces contiguity, we can prove the performance of the greedy algorithm. For future work, it will be interesting to consider developing a ZDD-based algorithm as discussed in the above sections for constrained weakly submodular maximization.

5.5 Conclusion

We proposed a novel approximation algorithm for submodular function maximization over graphs, which includes many important and practical problems such as SOPs, SSTPs, and SVCNPs. Our algorithm avoids the difficulty of checking feasibility by reducing the original problems to those of finding a route in a ZDD. We proved that our algorithm achieves $(1-\kappa)$-approximations where κ is the curvature of the objective submodular function. We observed the performance of our algorithm via numerical experiments involving three kinds of sensing tasks. The results showed that our algorithm finds better solutions than the alternative approximation algorithm in most instances, and runs much faster than the exact algorithms. Notably, our algorithm achieved more than a 90%-approximation in all instances for which optimal values were available.

In some experiments, we observed that the $(1-\kappa)$-approximation guarantee tends to be pessimistic. Actually, our algorithm found optimal solutions for SOP instances with $\kappa > 0.9$, implying that it might be possible to improve the $(1-\kappa)$-approximation guarantee. We believe that submodular optimization with DDs is a promising subject of study with much room for development, and so offers a significant advance in the field of constrained submodular optimization.

Chapter 6

Best-first Search Algorithm for Submodular Knapsack Problem

In this chapter, we develop an algorithm that can achieve arbitrary approximation guarantees for monotone submodular function maximization with a knapsack constraint. Although greedy-based algorithms are guaranteed to achieve constant-factor approximation guarantees for the problem, many applications require solutions with better guarantees; moreover, it is desirable to be able to control the trade-off between the computation time and approximation guarantee. Given this background, the *best-first search* (BFS) has been recently studied as a promising approach. However, existing BFS-based methods for submodular maximization sometimes suffer excessive computation cost since their *heuristic functions* are not well designed. We here present an accelerated BFS by introducing a new termination condition and developing a novel method for computing an upper-bound of the optimal value for submodular maximization, which enables us to use a better heuristic function. Experiments show that our accelerated BFS is far more efficient in terms of both time and space complexities than existing methods.

6.1 Introduction

This chapter considers the submodular knapsack problem (SKPs). The following discussion is related to two research subjects: submodular function maximization and search algorithms. Therefore, for convenience, we employ the following notation in this chapter. We let $U := [n]$ be a finite ground set,

and we denote the objective function by $g : 2^U \to \mathbb{R}$, which we assume to be normalized, monotone, and submodular. We define $X + Y := X \cup Y$ and $X - Y := X \backslash Y$ for any $X, Y \subseteq U$. We abuse notation and sometimes regard $v \in U$ as a subset of U; for instance, we let $X + v = X \cup \{v\}$.

Let $c_v \geq 0$ be a cost value defined for each $v \in U$. We define $c(X) := \sum_{v \in X} c_v$ and $c_X := \{c_v : v \in X\}$ for any $X \subseteq U$. Given a budget $B > 0$, we address the following SKP:

$$\underset{X \subseteq U}{\text{maximize}} \ g(X) \quad \text{subject to } c(X) \leq B. \tag{6.1}$$

In what follows, X^* denotes an optimal solution to SKP (6.1).

Although the greedy-style algorithms can efficiently compute approximate solutions to SKPs, their approximation guarantees are sometimes unsatisfying. In many important decision problems, we need to find solutions whose approximation guarantee is better than $1 - 1/\sqrt{e} \approx 0.39$ or $1 - 1/e \approx 0.63$, which are achieved by the greedy-style algorithms (see, Chapter 4). The *best-first search* (BFS), a general framework that includes the well-known A* search, is one of the most promising approaches that can achieve better approximation guarantees. The BFS-based approach for submodular maximization was established by Chen *et al.* [2015]; their original paper call the algorithm *filtered search* (FS). Although the time and space complexities of BFS are generally exponential in $|U|$, BFS can find α-approximate solutions for any $\alpha \in [0, 1]$, and we can control the trade-off between the computation cost and optimality by tuning the hyper-parameter, α. As in the standard A* search, BFS uses *heuristic function* $h(X)$ to measure how promising the current solution X is, and the performance of BFS strongly depends on how we design the heuristic function $h(\cdot)$. Unfortunately, as we will see in the experiments, BFSs with existing heuristic functions work poorly in some important SKP instances; this is partly because the existing heuristic function is not well designed.

6.1.1 Our Contribution

We propose an accelerated BFS for SKPs. The acceleration is attained by introducing a new termination condition to BFS and using a novel heuristic function. As we will see later, computing heuristic function values reduces to computing upper-bounds for optimal values of SKPs that appear as subproblems in BFS, and we propose a novel technique to compute such upper-bounds. The obtained upper-bound is empirically more accurate than those obtained

by existing techniques, thus enabling us to use a better heuristic function in BFS. Combining the upper-bound with the proposed termination condition substantially reduces the computation cost of BFS; our accelerated BFS is particularly effective when computing α-approximate solutions for $\alpha < 1$ (e.g., $\alpha = 0.7$ or 0.8). Experiments show that our algorithm outperforms existing methods in terms of both time and space complexities.

We note that obtaining accurate upper-bounds for SKPs is also beneficial in the empirical performance analysis of existing approximation algorithms for SKPs. Since SKP is NP-hard in general, computing optimal values for large SKPs is prohibitively expensive, making it almost impossible to evaluate the quality of obtained solutions for large SKPs. As shown later, our upper-bound computation method is as fast as the standard greedy algorithm, and it finds an upper-bound that is close to the optimal value. Thus the proposed upper-bound enables us to obtain empirical approximation guarantees that are typically better than $1 - 1/e$; this is helpful when evaluating the quality of given solutions for large SKPs.

6.1.2 Related Work

When it comes to obtaining approximation guarantees that are better than $1 - 1/e$, there are two major approaches: integer programming (IP) and BFS. The IP-based approach was first studied by Nemhauser and Wolsey [1981], who proposed a *branch-and-bound* algorithm. A more efficient IP-based algorithm using the *submodularity cut* is proposed in Kawahara *et al.* [2009]; although this method is applicable to non-monotone submodular maximization, it can only deal with the cardinality-constrained case. Unfortunately, IP-based methods are sometimes inefficient since they must solve subproblems too many times. The BFS-based methods are typically more efficient than the IP-based ones by virtue of their search strategy based on certain priorities, and so are attracting much attention recently. For submodular maximization with a specific objective function and an s–t path constraint, an A* search algorithm is proposed in [Zeng *et al.*, 2015]. Their algorithm employs the greedy algorithm for computing heuristic function values. For some submodular maximization problems including SKP, a BFS-based algorithm, called filtered search (FS), was proposed in [Chen *et al.*, 2015]. With their algorithm, heuristic function values are computed by solving relaxed knapsack problems (KPs). FS has a hyper-parameter, $\alpha \in [0, 1]$, that a user can control, and FS is guaranteed to find α-optimal solutions; this technique is studied as the

weighted A* search in the field of search problems [Pohl, 1970; Ebendt and Drechsler, 2009].

6.2 BFS for SKPs and Acceleration Techniques

We explain some basics of BFS for SKP (6.1) and additional techniques for its acceleration: *under estimation* and *termination condition*. The detail of BFS with these techniques is shown in Algorithm 9. We also provide a theoretical analysis on the proposed algorithm.

6.2.1 BFS for SKPs

We briefly review the procedures of BFS for SKPs, as detailed in [Chen *et al.*, 2015].

Let $\mathcal{F} := \{X \subseteq U : c(X) \leq B\}$ be the collection of all feasible solutions, and remember that elements in U are numbered by $1, \ldots, n$. We define a *state-space tree* $G = (\mathcal{F}, E)$, which is a directed tree whose root is $\emptyset \in \mathcal{F}$.[1] Each node corresponds to a feasible solution $X \in \mathcal{F}$, and is called a *state*; the pair $X, Y \in \mathcal{F}$ has a directed edge $(X, Y) \in E$ if and only if $X = Y - \max Y$ holds, where $\max Y$ is an element in Y with the largest number. For example, $X = \{2\}$ and $Y = \{2, 4\}$ have an edge (X, Y) since $Y - \max Y = \{2, 4\} - \{4\} = \{2\} = X$. We abuse the notation and suppose that the set U has the linear order $<$; for example, we use $\{2\} < \{4\}$. Although $|\mathcal{F}|$ can be exponential in $|U|$, BFS expands states in $\mathcal{F}$ on-demand, typically requiring the storage of a very small fraction of $\mathcal{F}$.

As in Algorithm 9, BFS employs a max heap (i.e., a priority queue) to manages $\langle$key, value$\rangle$ pairs. A key corresponds to state $X \in \mathcal{F}$ and the priority value of X is defined as

$$f(X) := g(X) + h(X),$$

where $h(\cdot)$ is a heuristic function. In each step of BFS, we pop the state with the biggest priority value from the heap and push its child states onto the heap. Let $T \in \mathcal{F}$ be a state popped from the heap. Then all child

[1] The original paper of FS uses a *state-space graph* and *closed list* so that each state is examined at most once. In practice, however, it is inefficient to use the closed list explicitly. Thus we employ here the state-space tree often used in *reverse search* [Avis and Fukuda, 1996].

states $S \in \mathcal{F}$ such that $(T, S) \in E$ (i.e., $S = T + v$ for all $v \in U$ such that $v > \max T$) are pushed onto the heap. These procedures are repeated until solution T such that $h(T) = 0$ is obtained. BFS is guaranteed to be optimal if the heuristic function is *admissible*, i.e., $h(X^*) = 0$ and $h(X) \geq g(X^*) - g(X)$ for any optimal solution X^* and $X \subseteq U$.

6.2.2 Under Estimation

While computing an optimal solution with BFS is sometimes too computationally expensive, an approximate solution can often be obtained efficiently by employing the under estimation technique as in [Chen *et al.*, 2015]; this approach is analogous to that of the weighted A* algorithm [Pohl, 1970; Ebendt and Drechsler, 2009]. Specifically, instead of using $f(T) = g(T) + h(T)$, we employ the under-estimated priority $f(T) = g(T) + \alpha h(T)$, where $\alpha \in [0, 1]$ is a controllable hyper parameter. From the non-negativity of $g(\cdot)$ and the admissibility of $h(\cdot)$, we have $f(T) \geq \alpha(g(T) + h(T)) \geq \alpha g(X^*)$, which guarantees the α-optimality of BFS.

6.2.3 Proposed Termination Condition

We now explain the proposed termination condition (Steps 12–16), which detects that an α-approximate solution has been already obtained; we refer to our algorithm as BFS with termination condition (BFSTC).

As we will show later, evaluating $h(S)$ requires to obtain subset $\hat{S} \subseteq U$ such that $S + \hat{S} \in \mathcal{F}$. If we can detect $g(S + \hat{S})/g(X^*) \geq \alpha$ without knowing the true value of $g(X^*)$, we can immediately stop the search and return $S + \hat{S}$ as an α-approximate solution. We use this idea to create BFSTC. In Steps 1 and 12, GetFeasible(S) computes $\hat{S}$; for example, we may use the standard greedy algorithm to obtain $\hat{S}$. How to compute $\hat{S}$ depends on the heuristic function used, and thus we show the details of GetFeasible later for each heuristic function. $S_{\max}$ maintained in Algorithm 9 always gives a feasible solution, and it is updated in Step 13 so that $S_{\max}$ is the current best solution. Furthermore, since we have $f(T) \geq \alpha g(X^*)$ thanks to the admissibility of $h(\cdot)$, g_{upper} maintained in Algorithm 9 always gives an upper-bound of $g(X^*)$; $g(S_{\max})/g(X^*) \geq g(S_{\max})/g_{\text{upper}}$ always holds. Namely, $g(S_{\max})/g_{\text{upper}} \geq \alpha$ implies $S_{\max}$ is an α-approximate solution, i.e., $g(S_{\max})/g(X^*) \geq \alpha$. As in the experiments, the termination condition reduces the search effort of our algorithm, particularly when computing α-approximate solutions for $\alpha \leq 0.7$.

Algorithm 9 BFSTC($U, g(\cdot), c_U, B, \alpha$)

$S_{\max} \leftarrow \text{GetFeasible}(\emptyset)$
$f(\emptyset) \leftarrow \alpha h(\emptyset)$
$g_{\text{upper}} \leftarrow f(\emptyset)/\alpha$
MaxHeap.push($\langle \emptyset, f(\emptyset) \rangle$)
while MaxHeap is not empty **do**
    $T \leftarrow$ MaxHeap.pop()
    if $h(T) = 0$ **then**
        return T
    end if
    $g_{\text{upper}} \leftarrow \min\{g_{\text{upper}}, f(T)/\alpha\}$
    for each S such that $(T, S) \in E$ **do**
        $\hat{S} \leftarrow \text{GetFeasible}(S)$
        $S_{\max} \leftarrow \text{argmax}_{X \in \{S+\hat{S}, S_{\max}\}} g(X)$
        if $g(S_{\max})/g_{\text{upper}} \geq \alpha$ **then**
            return $S_{\max}$
        end if
        $f(S) \leftarrow g(S) + \alpha h(S)$
        MaxHeap.push($\langle S, f(S) \rangle$)
    end for
end while

6.2.4 Theoretical Analysis

We here show a sufficient condition for BFSTC to achieve α-approximation; more specifically, we show a weaker version of the admissibility that suffices to obtain the α-optimality of BFSTC. The condition is a key to obtaining the proposed heuristic function. As shown by Chen *et al.* [2015], if heuristic function $h(\cdot)$ satisfies the following *critical admissibility* (CA),[2] then BFS is α-optimal. In what follows, we define

$$V_S := \{v \in U : v > \max S\}$$

for any given $S \subseteq U$; namely, V_S consists of the elements that are considered to be added to S in BFSTC.

[2] The above definition of CA is slightly different from the original one presented in Chen *et al.* [2015] since we employed the state-space tree instead of the state-space graph.

Definition 1 (critical admissibility)**.** *We say $h(\cdot)$ satisfies CA if the following conditions hold for any $S \subseteq U$:*

1. *$h(S) = 0$ if $g(v \mid S) \leq 0$ or $S + v \notin \mathcal{F}$ for all $v \in V_S$,*
2. *$h(S) \geq \max_{X \subseteq V_S : X+S \in \mathcal{F}} \sum_{v \in X} g(v \mid S)$ otherwise.*

We introduce here the *weak critical admissibility* (WCA). With WCA, we can design a novel heuristic function as shown later, which substantially enhances BFS performance.

Definition 2 (weak critical admissibility)**.** *We say $h(\cdot)$ satisfies WCA if the following conditions hold for any $S \subseteq U$:*

1. *$h(S) = 0$ if $g(v \mid S) \leq 0$ or $S + v \notin \mathcal{F}$ for all $v \in V_S$,*
2. *$h(S) \geq \max_{X \subseteq V_S : X+S \in \mathcal{F}} g(X \mid S)$ otherwise.*

We name this condition WCA since heuristic function $h(\cdot)$ satisfying CA always satisfies WCA, while the reverse is not true; this can be confirmed using the submodularity of g. The following theorem guarantees that BFSTC is α-optimal if its heuristic function satisfies WCA.

Theorem 6. *If $h(\cdot)$ satisfies WCA, then solution $R \subseteq U$ obtained by BFSTC satisfies $g(R) \geq \alpha g(X^*)$.*

Proof. Let X^* be an optimal solution and assume $R \neq X^*$, otherwise the claim holds trivially.

We first show that the max heap always contains a subset $P \subseteq X^*$ such that $\max P < \min X^* \backslash P$ until R is popped; we regard $\max \emptyset = 0$ and $\min \emptyset = \infty$. This is apparently true at the beginning since $\emptyset \subseteq X^*$ is in the heap. Whenever such P is popped, we have either

(i) $P = X^*$, or

(ii) all $P' \subseteq X^*$ such that $(P, P') \in E$ are pushed onto the heap, one of which satisfies $\max P' < \min X^* \backslash P'$.

If case (i) occurs, then we have $h(P) = 0$ since $h(\cdot)$ satisfies the first condition of WCA and $P = X^*$ is optimal. As a result, BFSTC returns the optimal solution $P = X^*$, which contradicts the assumption. Thus case (ii) continues

to hold until BFSTC returns R. Consequently, a subset $P \subseteq X^*$ such that $\max P < \min X^* \backslash P$ must be in the heap until R is popped.

We then prove $f(T) \geq \alpha g(X^*)$ for any T popped from the heap. As shown above, the heap always contains a subset $P \subseteq X^*$ with $\max P < \min X^* \backslash P$. For such P, we have

$$\begin{aligned} f(P) &= g(P) + \alpha h(P) \\ &\geq \alpha(g(P) + h(P)) \\ &\geq \alpha(g(P) + g(X^* \backslash P \mid P)) \\ &= \alpha g(X^*), \end{aligned}$$

where the second inequality comes from the fact that $h(\cdot)$ satisfies the second condition of WCA. Since the popped T has the largest f value in the max heap, we obtain $f(T) \geq f(P) \geq \alpha g(X^*)$. We note that, for some popped T, we always have $g_{\text{upper}} = f(T)/\alpha$. Hence g_{upper} always gives an upper-bound for $g(X^*)$, i.e., $g_{\text{upper}} \geq g(X^*)$.

We now consider two cases: R is obtained in Step 8 or in Step 15. If R is obtained in Step 15, we have $g(R) \geq \alpha g_{\text{upper}} \geq \alpha g(X^*)$. If R is obtained in Step 8, we have $h(R) = 0$, which leads to

$$g(R) = g(R) + \alpha h(R) = f(R) \geq \alpha g(X^*),$$

where $f(R) \geq \alpha g(X^*)$ comes from the fact that R is popped from the max heap. Therefore, $g(R) \geq \alpha g(X^*)$ holds in both cases, and thus the proof is completed. □

In the next section we discuss how to design $h(\cdot)$. The first condition of WCA (or CA) is easily satisfied by examining all $v \in V_S$, and thus we focus on the second condition. Given current solution S, heuristic function value $h(S)$ satisfying the second condition of WCA is obtained as an upper-bound of the optimal value of the following SKP:

$$\underset{Y \subseteq V}{\text{maximize}} \ g_S(Y) \quad \text{subject to } c(Y) \leq B_S, \tag{6.2}$$

where $B_S := B - c(S)$, $V := \{v \in V_S : c(v) \leq B_S\}$, and $g_S(Y) := g(S \cup Y) - g(S)$ $(\forall Y \subseteq V)$ is a monotone submodular function defined on $U - S$; $g_S(\cdot)$ is sometimes called the *contraction* of g on S (see, e.g., [Bach, 2013]) and satisfies $g_S(\emptyset) = 0$. Therefore, in the next section, we discuss how to obtain upper-bounds of the optimal value of SKP (6.2).

6.3 Upper-bound Computation for SKP

Algorithm 10 GreedySK$(V, g_S(\cdot), c_V, B_S)$

$X \leftarrow \text{GreedyAdd}(V, g_S(\cdot), c_V, B_S)$
$v_{\max} \leftarrow \max_{v \in V} g_S(v)$
$Z \leftarrow \text{argmax}_{Y \in \{v_{\max}, X\}} g_S(Y)$
return Z

Algorithm 11 GreedyAdd$(V, g_S(\cdot), c_V, B_S)$

$X \leftarrow \emptyset$
while $V \neq \emptyset$ **do**
    $\hat{v} \leftarrow \text{argmax}_{v \in V} g_S(v \mid X)/c_v$
    if $c(X + \hat{v}) \leq B_S$ **then**
        $X \leftarrow X + \hat{v}$
    end if
    $V \leftarrow V - \hat{v}$
end while
return X

As we have seen above, given solution S, heuristic function value $h(S)$ is obtained by computing an upper-bound of SKP (6.2). In this section, we let $Y^* \subseteq V$ denote an optimal solution for SKP (6.2), and we discuss how to compute an upper-bound of $g_S(Y^*)$, which we use as heuristic function value $h(S)$. In what follows, for a given ordered subset $X \subseteq V$, we let $X_{1:i}$ denote the set of the first i elements ($X_{1:i} = \emptyset$ if $i < 1$) and X_i denotes its i-th element.

For later use, we show the ordinary greedy algorithm for SKP (6.2) in Algorithm 10, which we call GreedySK. The algorithm computes two candidates, X and $v_{\max}$, as its output: X is obtained by adding the elements with the largest cost-benefit ratio sequentially, and $v_{\max}$ is a singleton with the maximum objective value. We refer to the algorithm for computing the first candidate X as GreedyAdd, and we describe it separately from GreedySK for later use.

In the rest of this section, we present three upper-bounds of $g_S(Y^*)$, which we name u_{app}, u_{mod}, and u_{dom}. The first two bounds are obtained with

existing techniques. The third one is the proposed upper-bound, and is used as heuristic function value $h(S)$ in our algorithm.

6.3.1 Upper-bound with Approximation Ratios ($u_{\mathbf{app}}$)

Given γ-approximate solution Z for SKP (6.2), the simplest way to obtain an upper-bound of $g_S(Y^*)$ is to compute $u_{\mathrm{app}} := g_S(Z)/\gamma$. As in [Lin and Bilmes, 2010], GreedySK for SKPs achieves at least $(1 - 1/\sqrt{e})$-approximation; more strictly, if X is the output of GreedyAdd, the approximation ratio achieved by GreedySK is at least $\gamma = 1 - \left(1 - \frac{1}{2|X|}\right)^{|X|}$. In the cardinality-constrained case, we can improve the upper-bound using the results in [Conforti and Cornuéjols, 1984] as follows: we compute the curvature $\kappa \in [0, 1]$ of submodular function g_S and we let $\gamma = \frac{1}{\kappa}\left(1 - (1 - \kappa/|X|)^{|X|}\right)$. The A* algorithm, whose heuristic function is a variant of the above u_{app}, is studied in [Zeng *et al.*, 2015] for a specific objective function.

We now consider the computational aspect of BFSTC that uses u_{app} as its heuristic function value. For current S, we can compute $h(S) = u_{\mathrm{app}}$ efficiently once we obtain $g_S(Z)$, where Z is the output of GreedySK$(V, g_S(\cdot), c_V, B_S)$. Therefore, in Steps 1 and 12 of BFSTC, we use GreedySK$(V, g_S(\cdot), c_V, B_S)$ as GetFeasible(S), with which we compute $g_S(Z)$ and let $h(S) = g_S(Z)/\gamma$.

6.3.2 Upper-bound with Modular Functions ($u_{\mathbf{mod}}$)

The second upper-bound u_{mod} is the one used in [Chen *et al.*, 2015], which gives a heuristic function satisfying CA. Thanks to the submodularity of g_S, we have $\sum_{v \in X} g_S(v) \geq g_S(X)$ for any $X \subseteq V$, and thus we obtain

$$\max_{X \subseteq V : c(X) \leq B_S} \sum_{v \in X} g_S(v) \geq \sum_{v \in Y^*} g_S(v) \geq g_S(Y^*) = g_S(Y^*). \tag{6.3}$$

Therefore, in order to bound $g_S(Y^*)$ from above, we compute an upper-bound of the left-hand side of eq. (6.3), which is an optimal value of KP. Such an upper-bound is easily obtained by considering the *fractional relaxation* of KP as follows. We sort V in the non-increasing order of $r_v := g_S(v)/c_v$; $V_{1:i}$ denotes the first i elements of the sorted set. If we have $r_v = \infty$ (i.e., $g_S(v) > 0$ and $c_v = 0$) for some $v \in V$, we place them at the beginning of V in arbitrary order. Furthermore, we define $l := \max\{i \in [|V|] : c(V_{1:l}) \leq B_S\}$.

If $l < |V|$, then the following value bounds the left-hand side of eq. (6.3) from above (see, [Dantzig, 1957]):

$$u_{\mathrm{mod}} := \sum_{v \in V_{1:l}} g_S(v) + (B_S - c(V_{1:l}))\, r_{V_{l+1}}.$$

If $l = |V|$, then $g_S(Y^*) = g_S(V)$ thanks to the monotonicity of g_S, and thus we let $u_{\mathrm{mod}} := g_S(V)$. Consequently, u_{mod} always bounds $g_S(Y^*)$ from above.

Given current solution S, we consider computing $h(S) = u_{\mathrm{mod}}$ efficiently in BFSTC. As discussed above, u_{mod} is obtained by solving the relaxed KP. Thus in Steps 1 and 12 of BFSTC, we let GetFeasible(S) compute $V_{1:l+1}$ and output $V_{1:l}$ if $l < |V|$; note that we have $V_{1:l} + S \in \mathcal{F}$. We then set $h(S)$ to the objective value that is obtained by solving the relaxed KP; namely $h(S) = \sum_{v \in V_{1:l}} g_S(v) + (B_S - c(V_{1:l}))\, g_S(V_{l+1})/c_{V_{l+1}}$. If $l = |V|$, we let GetFeasible(S) output V and set $h(S) = g_S(V)$.

For later use, given any $Y \subseteq V$, we define $u_{\mathrm{mod}}(Y)$ as an upper-bound of $\max_{X \subseteq V \setminus Y : c(X) \leq B_S} g_S(X \mid Y)$ that is obtained by solving its relaxed KP as shown above.

6.3.3 Upper-bound with Dominant Elements ($u_{\mathbf{dom}}$)

We here propose the new upper-bound u_{dom}. To compute u_{dom}, we use an output of GreedyAdd($V, g_S(\cdot), c_V, B_S$), which we denote $X_{1:k}$; $X_{1:i}$ is the set of the first i elements added by GreedyAdd for $i \in [k]$. We define

$$\beta_{1:k} := \begin{cases} 0 & \text{if } u_{\mathrm{mod}}(X_{1:i}) = 0 \text{ for some } i \in [k], \\ \prod_{i=1}^{k} \beta_i & \text{otherwise,} \end{cases}$$

$$\beta_i := 1 - \frac{g_S(X_i \mid X_{1:i-1})}{u_{\mathrm{mod}}(X_{1:i-1})} \quad \text{for } i \in [k].$$

Then we let $u_{\mathrm{dom}} := g_S(X_{1:k})/(1 - \beta_{1:k})$.

Theorem 7. *$u_{dom} \geq g_S(Y^*)$ holds for any given $S \subseteq U$.*

Proof. For $i = 0, \ldots, k$, we have

$$\begin{aligned} g_S(Y^*) &\leq g_S(Y^* + X_{1:i}) \\ &= g_S(X_{1:i}) + g_S(Y^* \mid X_{1:i}) \\ &\leq g_S(X_{1:i}) + \max_{X \subseteq V \setminus X_{1:i} : c(X) \leq B_S} g_S(X \mid X_{1:i}) \end{aligned}$$

$$\leq g_S(X_{1:i}) + u_{\mathrm{mod}}(X_{1:i}),$$

where the third inequality comes from the definition of $u_{\mathrm{mod}}(X_{1:i})$.

We first consider the case where $u_{\mathrm{mod}}(X_{1:i}) = 0$ holds for some $i \in [k]$. In this case we have

$$g_S(Y^*) \leq g_S(X_{1:i}) + u_{\mathrm{mod}}(X_{1:i}) = g_S(X_{1:i})$$

from inequality (**??**). Thanks to the monotonicity of g_S, we have $g_S(X_{1:k}) \geq g_S(X_{1:i})$, and hence $u_{\mathrm{dom}} = g_S(X_{1:k}) \geq g_S(Y^*)$ holds; more precisely, $u_{\mathrm{dom}} = g_S(X_{1:k}) = g_S(Y^*)$ holds since Y^* is optimal.

We then consider the case where $u_{\mathrm{mod}}(X_{1:i}) > 0$ holds for all $i \in [k]$. For $i \in [k]$, we have

$$\begin{aligned} g_S(Y^*) &\leq g_S(X_{1:i-1}) + u_{\mathrm{mod}}(X_{1:i-1}) \\ &= g_S(X_{1:i-1}) + \frac{u_{\mathrm{mod}}(X_{1:i-1})}{g_S(X_i \mid X_{1:i-1})}(g_S(X_{1:i}) - g_S(X_{1:i-1})) \\ &= g_S(X_{1:i-1}) + \frac{1}{1-\beta_i}(g_S(X_{1:i}) - g_S(X_{1:i-1})) \end{aligned}$$

from inequality (**??**). Rearranging the terms yields

$$g_S(X_{1:i}) \geq \beta_i g_S(X_{1:i-1}) + (1-\beta_i) g_S(Y^*),$$

and thus the following inequality holds for $i \in [k]$:

$$g_S(Y^*) - g_S(X_{1:i}) \leq \beta_i (g_S(Y^*) - g_S(X_{1:i-1})).$$

Therefore, from $g_S(\emptyset) = 0$, we obtain

$$g_S(X_{1:k}) \geq \left(1 - \prod_{i=1}^{k} \beta_i\right) g_S(Y^*) = (1 - \beta_{1:k})\, g_S(Y^*).$$

Hence $u_{\mathrm{dom}} = g_S(X_{1:k})/(1-\beta_{1:k}) \geq g_S(Y^*)$ holds. □

The above proof is analogous to the well-known proof of $(1 - 1/e)$-approximation for cardinality-constrained submodular maximization. In the cardinality-constrained case, we have $\beta_i = 1 - 1/k$ in the worst case, which leads to the $(1 - 1/e)$-approximation guarantee; from this fact, the simple upper-bound $g_S(X_{1:k})/(1 - 1/e) \geq g_S(Y^*)$ can be obtained. In most

cases, however, we have $\beta_i < 1 - 1/k$, and hence u_{dom} is expected to be closer to $g_S(Y^*)$ than $g_S(X_{1:k})/(1 - 1/e)$.

We discuss when u_{dom} gives an upper-bound that is close to $g_S(Y^*)$. From the definition of u_{dom}, we see that u_{dom} is likely to overestimate $g_S(Y^*)$ if $\beta_{1:k} \in [0, 1]$ is close to 1; namely $\beta_{1:k} \approx 0$ is desirable, which holds if we have $\beta_i \approx 0$ for some $i \in [k]$. Such small β_i can be obtained if we have $g_S(X_i \mid X_{1:i-1})/u_{\mathrm{mod}}(X_{1:i-1}) \approx 1$, which occurs if X_i is has a dominant marginal gain compared to $v \in V - X_{1:i}$. In other words, if $g_S(X_i \mid X_{1:i-1}) \gg g_S(v \mid X_{1:i-1})$ holds for all $v \in V - X_{1:i}$, then we have $\beta_i \approx 0$, and consequently we obtain $\beta_{1:k} \approx 0$. Namely, the accuracy of u_{dom} depends on the dominance of the elements selected by GreedyAdd.

We consider computing $h(S) = u_{\mathrm{dom}}$ efficiently in BFSTC for current solution S. As shown above, u_{dom} can be computed easily if we have $g_S(X_{1:k})$ and the marginal gains $g_S(v \mid X_{1:i-1})$ for all $v \in V - X_{1:i-1}$ and $i \in [k]$, which can be obtained once GreedyAdd$(V, g_S(\cdot), c_V, B_S)$ is executed. Therefore, in Steps 1 and 12 of BFSTC, we get $\hat{S}$ by using GreedySK$(V, g_S(\cdot), c_V, B_S)$, which includes GreedyAdd$(V, g_S(\cdot), c_V, B_S)$ as its building block. Then we can compute u_{dom} without additional function evaluation.

6.4 Experiments

All experiments were conducted on a 64-bit Cent7.3.1611 machine with Xeon 5E-2697 v3 2.6 GHz CPUs and 256 GB of RAM. Our algorithm is BFSTC with the heuristic function given by u_{dom}, which we call DOM. To benchmark our algorithm, we use BFSTC with the heuristic functions given by u_{app} and u_{mod}, which we call APP and MOD, respectively. Note that APP and MOD are improved variants of those in [Zeng *et al.*, 2015] and [Chen *et al.*, 2015], respectively. All instances considered are formulated as SKPs. For the objective functions, we use the following *weighted coverage* (COV) function, *facility location* (LOC) function, and *bipartite influence* (INF) function.

Weighted Coverage (COV). The first one is the well-known weighted coverage function (see, e.g., [Krause and Golovin, 2014]). Let $[m]$ be a set of m items. We define $w_i \geq 0$ as the weight of i-th item for each $i \in [m]$. Each $v \in U$ covers some items, and we let $I_v \subseteq [m]$ indicate the set of items covered

by v. The following weighted coverage function is monotone and submodular:

$$g(X) := \sum_{i \in \bigcup_{v \in X} I_v} w_i.$$

This function often appears in the context of itemset mining (e.g., [Kumar *et al.*, 2015]). Here c_v is the cost of choosing v.

Facility Location (LOC). The second one is the *facility location* function (see, e.g., [Krause and Golovin, 2014]). We regard U as a set of locations and consider selecting some locations at which to build certain facilities. Let $[m]$ be a set of m clients. We define $q_{i,v} \geq 0$ as the benefit i-th client gains from the facility built at v. Given $X \subseteq U$, which represents the set of locations where the facilities are built, each client gains benefit from the most beneficial facility, and thus the total benefit for the clients is defined as

$$g(X) = \sum_{i \in [m]} \max_{v \in X} q_{i,v},$$

which is known to be monotone and submodular. Here c_v represents the cost of building a facility at v.

Bipartite Influence (INF). The third one represents an influence model on bipartite graphs, which is a special case of the problem studied in [Alon *et al.*, 2012]. Let U be a set of items. We regard $[m]$ as a set of m targets. Given bipartite graph $G = (U, [m]; A)$, where $A \subseteq U \times [m]$ is a set of directed edges, we consider an influence maximization problem on G. The probability that i-th target gets activated by items $X \subseteq U$ is $1 - \prod_{v \in X:(v,i) \in A}(1 - p_v)$, where $p_v \in [0, 1]$ is the activation probability of item v. Objective function $g(X)$ is the expected number of targets that are activated by items in X, which can be written as

$$g(X) = \sum_{i \in [m]} \left(1 - \prod_{v \in X:(v,i) \in A} (1 - p_v) \right).$$

In this setting, c_v represents the cost of choosing v.

6.4.1 Artificial Instances

We randomly generated 100 instances for COV, LOC, and INF, and we compared the performance of the three algorithms. In all instances, we set $|U| = 100$ and $B = 1$. The costs c_v ($v \in U$) are drawn randomly from a uniform distribution on $[0, 1]$; we denote a uniform distribution over $[a, b]$ by $u_{[a,b]}$. We let $m = 1000$ for each m that appears in the definitions of COV, LOC, and INF. In COV instances, w_i are drawn from $u_{[0,1]}$, and each $v \in U$ randomly covers each $w \in W$ with probability 0.3. In LOC instances, all benefits, $q_{i,v}$, are drawn from $u_{[0,1]}$. In INF instances, all p_v are drawn from $u_{[0,1]}$, and each pair $(v, i) \in U \times [m]$ is connected randomly with probability 0.3. In all instances, we set the time limit to one hour.

The results are summarized in Table 6.1 for various approximation ratios $\alpha = 0.4, \ldots, 1.0$, which can be controlled by the user; since the greedy algorithm for SKP achieves $1 - 1/\sqrt{e} \approx 0.39$ approximation, we consider only the case $\alpha > 0.39$. For each method, we observed and compared the number of instances solved without exceeding the time limit, number of nodes pushed onto the heap (averaged over all 100 instances), and running time. In Table 6.2, we also present the results on the number of nodes pushed onto the heap averaged over solved instances. The running times in Table 6.1 is compared for two pairs (DOM, MOD) and (DOM, APP), both of which are averaged over all instances solved by the both methods. Our method DOM substantially outperforms the other methods in all aspects. Notably, in the case $\alpha \leq 0.7$, DOM pushes only one node onto the heap, thus requiring dramatically less time and space complexities than the other methods. This result is thanks to the high accuracy of u_{dom} and the proposed termination condition; our upper-bound u_{dom} is so accurate that DOM detected the α-optimality of the solution obtained by GreedySK at the beginning, and thus it terminated in Step 15 of Algorithm 9 without pushing additional nodes onto the heap.

Table 6.1: Numerical results for artificial instances. '# solved' is the number of instances solved without exceeding the time limit. '# nodes' is the number of nodes pushed onto the heap averaged over all instances including those unsolved. 'A–B time' compares the running times for the two methods (A, B) = (DOM, MOD) and (DOM, APP); the running times are averaged over instances solved by both A and B. For each value of α the best results are given in bold.

	Approximation ratio α		0.4	0.5	0.6	0.7	0.8	0.9	1.0
COV	# solved	DOM	**100**	**100**	**100**	**100**	**100**	**96**	**84**
		MOD	**100**	97	91	84	73	64	62
		APP	**100**	83	59	47	27	24	20
	# nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**3.98e+3**	**5.96e+5**	**1.55e+6**
		MOD	1.13e+5	4.66e+5	9.05e+5	1.39e+6	1.98e+6	2.56e+6	2.92e+6
		APP	1.35e+4	1.43e+6	2.88e+6	4.05e+6	7.29e+6	8.16e+6	9.67e+6
	DOM–MOD time (s)	DOM	**2.47e-2**	**2.61e-2**	**2.60e-2**	**2.35e-2**	**2.92e+0**	**6.32e+1**	**1.95e+2**
		MOD	2.72e+1	1.24e+2	2.79e+2	4.13e+2	4.83e+2	4.80e+2	8.75e+2
	DOM–APP time (s)	DOM	**2.47e-2**	**2.49e-2**	**2.49e-2**	**2.19e-2**	**2.07e+0**	**2.30e+1**	**4.75e+1**
		APP	3.42e+0	4.28e+2	7.19e+2	1.36e+3	1.09e+3	1.05e+3	8.04e+2
LOC	# solved	DOM	**100**	**100**	**100**	**100**	**100**	**100**	**100**
		MOD	**100**	**100**	**100**	98	93	87	76
		APP	**100**	98	77	60	45	38	32
	# nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**2.08e+3**	**1.03e+5**	**5.38e+5**
		MOD	2.27e+4	8.83e+4	2.53e+5	5.40e+5	9.25e+5	1.37e+6	1.91e+6
		APP	4.99e+2	5.13e+5	1.79e+6	2.88e+6	4.12e+6	5.10e+6	6.18e+6
	DOM–MOD time (s)	DOM	**1.72e-2**	**1.70e-2**	**1.74e-2**	**1.70e-2**	**1.55e+0**	**3.17e+1**	**7.74e+1**
		MOD	3.67e+0	1.38e+1	6.74e+1	1.50e+2	2.49e+2	4.48e+2	4.33e+2
	DOM–APP time (s)	DOM	**1.72e-2**	**1.68e-2**	**1.64e-2**	**1.57e-2**	**1.25e+0**	**1.27e+1**	**2.62e+1**
		APP	1.90e-1	1.20e+2	4.96e+2	7.74e+2	6.84e+2	9.41e+2	8.35e+2
INF	# solved	DOM	**100**	**100**	**100**	**100**	**100**	**100**	**100**
		MOD	**100**	**100**	**100**	**100**	97	93	89
		APP	**100**	99	87	69	48	38	29
	# nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**8.76e+2**	**6.02e+4**	**3.52e+5**
		MOD	5.59e+2	9.62e+3	5.81e+4	2.11e+5	5.05e+5	7.99e+5	1.11e+6
		APP	2.85e+2	4.25e+5	1.24e+6	2.23e+6	3.71e+6	5.04e+6	6.78e+6
	DOM–MOD time (s)	DOM	**1.59e-3**	**1.65e-3**	**1.66e-3**	**1.82e-3**	**8.14e-2**	**2.97e+0**	**1.61e+1**
		MOD	2.25e-2	2.83e-1	1.93e+0	2.86e+1	1.26e+2	1.88e+2	3.01e+2
	DOM–APP time (s)	DOM	**1.59e-3**	**1.65e-3**	**1.60e-3**	**1.71e-3**	**6.70e-2**	**1.20e+0**	**3.20e+0**
		APP	1.45e-2	1.43e+2	4.12e+2	8.00e+2	6.99e+2	8.49e+2	8.04e+2

Table 6.2: Results on the number of nodes pushed onto the heap. '# nodes (solved)' is the number of pushed nodes averaged over instances solved by each method. 'A–B # nodes' is the number of pushed nodes averaged over all instances solved by both methods (A, B) = (DOM, MOD) or (DOM, APP). For each value of α the best results given in bold.

	Approximation ratio α		0.4	0.5	0.6	0.7	0.8	0.9	1.0
COV	# nodes (solved)	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**3.98e+3**	**4.29e+5**	**8.66e+5**
		MOD	1.13e+5	3.37e+5	5.40e+5	7.38e+5	7.86e+5	8.21e+5	1.02e+6
		APP	1.35e+4	7.47e+5	9.08e+5	1.06e+6	8.39e+5	7.66e+5	6.50e+5
	DOM–MOD # nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**4.35e+3**	**1.69e+5**	**4.71e+5**
		MOD	1.13e+5	3.37e+5	5.40e+5	7.38e+5	7.86e+5	8.21e+5	1.02e+6
	DOM–APP # nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**4.05e+3**	**6.77e+4**	**1.50e+5**
		APP	1.35e+4	7.47e+5	9.08e+5	1.06e+6	8.39e+5	7.66e+5	6.50e+5
LOC	# nodes (solved)	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**2.08e+3**	**1.03e+5**	**5.38e+5**
		MOD	2.27e+4	8.83e+4	2.53e+5	4.64e+5	6.34e+5	8.24e+5	8.23e+5
		APP	4.99e+2	4.41e+5	8.13e+5	9.27e+5	7.73e+5	8.34e+5	7.43e+5
	DOM–MOD # nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**2.20e+3**	**7.32e+4**	**2.45e+5**
		MOD	2.27e+4	8.83e+4	2.53e+5	4.64e+5	6.34e+5	8.24e+5	8.23e+5
	DOM–APP # nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.99e+3**	**3.31e+4**	**9.85e+4**
		APP	4.99e+2	4.41e+5	8.13e+5	9.27e+5	7.73e+5	8.34e+5	7.43e+5
INF	# nodes (solved)	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**8.76e+2**	**6.02e+4**	**3.52e+5**
		MOD	5.59e+2	9.62e+3	5.81e+4	2.11e+5	3.86e+5	5.18e+5	6.80e+5
		APP	2.85e+2	3.88e+5	7.59e+5	1.01e+6	9.51e+5	9.34e+5	8.45e+5
	DOM–MOD # nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**8.86e+2**	**4.94e+4**	**2.13e+5**
		MOD	5.59e+2	9.62e+3	5.81e+4	2.11e+5	3.86e+5	5.18e+5	6.80e+5
	DOM–APP # nodes	DOM	**1.00e+0**	**1.00e+0**	**1.00e+0**	**1.00e+0**	**7.94e+2**	**2.33e+4**	**7.48e+4**
		APP	2.85e+2	3.88e+5	7.59e+5	1.01e+6	9.51e+5	9.34e+5	8.45e+5

6.4.2 Real-world Instances

We applied the three algorithms to COV, LOC, and INF instances that are generated with real-world data. Since the data did not include information on cost values c_v, we drew them from $u_{[0.1,1]}$ in all instances; in practice it is rare for any item v to have cost of $c_v \approx 0$, and thus we set the lower-bound of cost to 0.1. The budget value, B, is set for each instance so that the resulting instance does not become computationally too demanding; we set B to a small value if $|U|$ is large. In all instances, the time limit is one day (86,400 seconds). Below we detail the experimental settings:

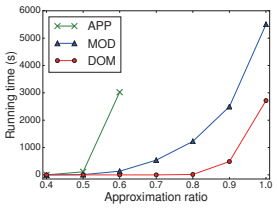
COV. The COV instance uses a dataset on messages exchanged by 899 users on 522 topics [Opsahl, 2013]. Each user posts messages on some topics with which he/she is familiar; we regard a user covers a topic if he/she posts messages to the topic. Each topic is weighted based on the number of posted messages; namely, the importance of topics is measured by the number of posted messages. We consider selecting some users so that the total weights of covered topics is maximized. Here we let $B = 1.25$.

LOC. The LOC instance uses a dataset on 473 subway stations in New York City (NYC) [Roest and Mashariki, 2015]. We regard the stations as clients, and consider building several facilities so that people at any station can easily reach one of the facilities. The candidate locations, on which facilities can be built, are given by a 10×10 grid that discretizes the NYC map, and we select some locations from the 100 candidate locations. The benefit a client (station) gains from a facility is computed based on the distance between them. In this instance we let $B = 1.5$.

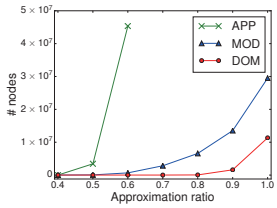
INF. The INF instance uses the MovieLens 100K dataset [Harper and Konstan, 2015]. The dataset contains $100,000$ ratings (1–5) by 943 users on 1682 movies, and we consider selecting some influential movies. The activation probability of each movie is computed from the ratings, and a movie can activate a user if he/she has rated the movie. Here we let $B = 1$.

Figures 6.1 (a)–(i) summarize the results; some results of APP for COV and INF instances are omitted since they exceeded the time limit for COV

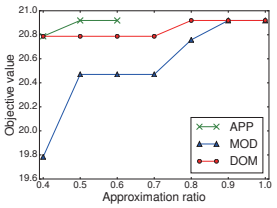
instances, and memory shortages occurred with in INF instances. Similar to the results on artificial instances, DOM outperforms the other methods both in time and space complexities. MOD and APP tend to require too much computational effort even for small α, achieving slightly higher objective values than DOM in some instances. This implies that MOD and APP are poor at using the hyper-parameter α to control the trade-off between the complexity and optimality. On the other hand, DOM successfully controlled the trade-off, and reduced the complexities when computing α-approximate solutions for small α; in particular DOM is efficient when $\alpha \leq 0.7$.



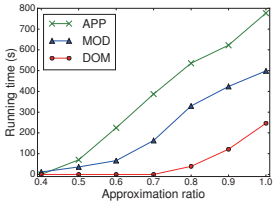
(a) Running times (COV)



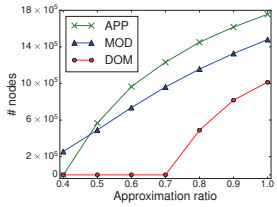
(b) # nodes (COV)



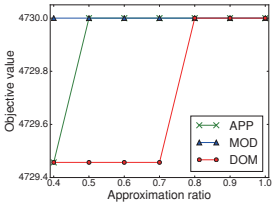
(c) Obj. values (COV)



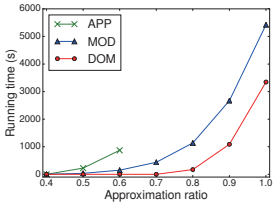
(d) Running times (LOC)



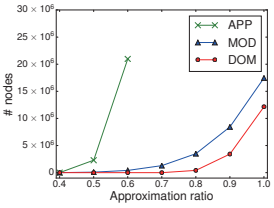
(e) # nodes (LOC)



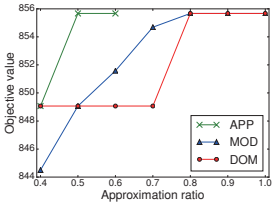
(f) Obj. values (LOC)



(g) Running times (INF)



(h) # nodes (INF)



(i) Obj. values (INF)

Figure 6.1: Numerical results on real-world instances. Figures (a)–(c), (d)–(f), and (g)–(i) show results for COV, LOC, and INF instances, respectively. (a), (d), (g): Running times of the three algorithms: DOM, MOD, and APP. (b), (e), (h): Number of nodes pushed onto the heap. (c), (f), (i): Achieved objective values.

6.5 Conclusion

We proposed an accelerated BFS for SKP. The acceleration is achieved by introducing a new termination condition and developing a novel method for computing an upper-bound of the optimal value of the SKP. For any given $\alpha \in [0, 1]$, our algorithm is proved to find an α-optimal solution. Experiments showed that our algorithm finds approximate solutions with substantially less time and space complexities than the existing methods.

In this chapter, as an application of the proposed upper-bound computation method, we focused on accelerating BFS. However, we believe that computing an upper-bound accurately is beneficial to many other algorithms that include submodular maximization as a subroutine. In future work we will try to develop more powerful variants of the upper-bound computation methods, which will improve the performance of various algorithms.

Chapter 7

k-submodular Function Maximization under Matroid Constraint

In this chapter, we consider monotone k-submodular function maximization under a matroid constraint. The k-submodular function is an extension of the submodular function such that its input is given by k disjoint subsets instead of a single subset. For unconstrained non-negative k-submodular maximization, Ward and Živný [2016] proposed a constant-factor approximation algorithm, and this was improved by the recent work [Iwata *et al.*, 2016], which presents a 1/2-approximation algorithm. Iwata *et al.* [2016] also provided a $k/(2k-1)$-approximation algorithm for non-negative monotone k-submodular maximization and proved that the approximation ratio is asymptotically tight. Ohsaka and Yoshida [2015] proposed constant-factor approximation algorithms for non-negative monotone k-submodular maximization with some cardinality constraints.

While the unconstrained and cardinality-constrained cases have been studied as above, no approximation guarantee has been obtained for k-submodular maximization with more general constraints. Here, we prove a 1/2-approximation guarantee of the greedy algorithm for non-negative monotone k-submodular maximization with a matroid constraint. The algorithm runs in $O(M|E|(\mathrm{IO} + k\mathrm{EO}))$ time, where M is the size of a maximal optimal solution, $|E|$ is the size of the ground set, and IO and EO represent the time for the independence oracle of the matroid and the evaluation oracle of the k-submodular function, respectively.

7.1 Introduction

The k-submodular function has been proposed by Huber and Kolmogorov [2012], which express the submodularity of choosing k disjoint sets of elements, instead of a single set. Let

$$(k+1)^E \coloneqq \{(X_1, \ldots, X_k) \mid X_i \subseteq E \ (i = 1, \ldots, k),\ X_i \cap X_j = \emptyset \ (i \neq j)\}.$$

A function $f : (k+1)^E \to \mathbb{R}$ is called *k-submodular* if, for any $\boldsymbol{x} = (X_1, \ldots, X_k)$ and $\boldsymbol{y} = (Y_1, \ldots, Y_k)$ in $(k+1)^E$, we have

$$f(\boldsymbol{x}) + f(\boldsymbol{y}) \geq f(\boldsymbol{x} \sqcup \boldsymbol{y}) + f(\boldsymbol{x} \sqcap \boldsymbol{y})$$

where

$$\boldsymbol{x} \sqcap \boldsymbol{y} \coloneqq (X_1 \cap Y_1, \ldots, X_k \cap Y_k),$$
$$\boldsymbol{x} \sqcup \boldsymbol{y} \coloneqq \left(X_1 \cup Y_1 \setminus \Big(\bigcup_{i \neq 1} X_i \cup Y_i\Big), \ldots, X_k \cup Y_k \setminus \Big(\bigcup_{i \neq k} X_i \cup Y_i\Big)\right).$$

For an input $\boldsymbol{x} = (X_1, \ldots, X_k)$ of a k-submodular function, we define the size of $\boldsymbol{x}$ by $\big|\bigcup_{i \in \{1,\ldots,k\}} X_i\big|$. We say f is *monotone* if $f(\boldsymbol{x}) \leq f(\boldsymbol{y})$ holds for any $\boldsymbol{x} = (X_1, \ldots, X_k)$ and $\boldsymbol{y} = (Y_1, \ldots, Y_k)$ satisfying $X_i \subseteq Y_i$ for $i = 1, \ldots, k$.

k-submodular functions appear in various scenarios: they are used when designing placement of k types of sensors and relaxing some NP-hard problems. Therefore, the k-submodular function is recently a popular subject of study [Gridchyn and Kolmogorov, 2013; Hirai and Iwamasa, 2016]. If $k = 1$, the above definition is equivalent to that of submodular functions. If $k = 2$, the k-submodular function is equivalent to the so-called *bisubmodular function*, for which maximization algorithms have been widely studied [Iwata *et al.*, 2013; Ward and Živný, 2016]. For unconstrained non-negative k-submodular maximization, Ward and Živný [2016] proposed a $\max\{1/3, 1/(1+a)\}$-approximation algorithm, where $a = \max\{1, \sqrt{(k-1)/4}\}$. Iwata *et al.* [2016] improved the approximation ratio to $1/2$. They also proposed a $k/(2k-1)$-approximation algorithm for non-negative monotone k-submodular maximization, and proved that, for any $\varepsilon > 0$, a $((k+1)/2k + \varepsilon)$-approximation algorithm for maximizing monotone k-submodular functions requires exponentially many oracle queries. This means their approximation ratio is asymptotically tight. Ohsaka and Yoshida [2015] proposed a $1/2$-approximation algorithm for non-negative monotone k-submodular maximization with a total

size constraint (i.e., $\left|\bigcup_{i\in\{1,\dots,k\}} X_i\right| \leq N$ for a non-negative integer N) and a 1/3-approximation algorithm for that with individual size constraints (i.e., $|X_i| \leq N_i$ for $i = 1, \dots, k$ with associated non-negative integers $N_1, \dots, N_k$).

We prove that the greedy algorithm achieves a 1/2-approximation for non-negative monotone k-submodular maximization with a matroid constraint. This approximation ratio is asymptotically tight due to the aforementioned hardness result by Iwata *et al.* [2016]. Given $\mathcal{F} \subseteq 2^E$, we say a system $(E, \mathcal{F})$ is *matroid* if the following conditions hold:

(M1) $\emptyset \in \mathcal{F}$,

(M2) If $A \subseteq B \in \mathcal{F}$ then $A \in \mathcal{F}$,

(M3) If $A, B \in \mathcal{F}$ and $|A| < |B|$ then there exists $e \in B \backslash A$ such that $A \cup \{e\} \in \mathcal{F}$.

If $A \in \mathcal{F}$, we say A is *independent*, and we say $A \in \mathcal{F}$ is *maximal* if no $B \in \mathcal{F}$ satisfies $A \subsetneq B$. Matroids include various systems $(E, \mathcal{F})$; below we list some examples.

(a) E is a finite set and

$$\mathcal{F} \coloneqq \{F \subseteq E \mid |F| \leq N\},$$

where N is a non-negative integer.

(b) E is the set of columns of a matrix over some field, and

$$\mathcal{F} \coloneqq \{F \subseteq E \mid \text{Columns in } F \text{ are linearly independent on the field}\}.$$

(c) E is the set of edges of a undirected graph G with a vertex set V, and

$$\mathcal{F} \coloneqq \{F \subseteq E \mid (V, F) \text{ forms a forest}\}.$$

(d) E is a finite set partitioned into ℓ subsets $E_1, \dots, E_\ell$ with associated non-negative integers $N_1, \dots, N_\ell$, and

$$\mathcal{F} \coloneqq \{F \subseteq E \mid |F \cap E_i| \leq N_i \text{ for } i = 1, \dots, \ell\}.$$

The total size constraint corresponds to **(a)**, which is called a *uniform matroid*. Since submodular functions and matroids are capable of modeling various problems, approximation algorithms for submodular function maximization (i.e., $k = 1$) with a matroid constraint have attracted much attention [Fisher *et al.*, 1978; Lee *et al.*, 2010; Calinescu *et al.*, 2011; Filmus and Ward, 2014]. However, to the best of our knowledge, no approximation algorithm has been studied for k-submodular maximization with a matroid constraint.

We here prove that the greedy algorithm achieves a 1/2-approximation guarantee for the following non-negative monotone k-submodular maximization with a matroid constraint:

$$\underset{\boldsymbol{x}\in(k+1)^E}{\text{maximize}}\ f(\boldsymbol{x}) \qquad \text{subject to} \bigcup_{\ell\in\{1,\dots,k\}} X_\ell \in \mathcal{F}, \tag{7.1}$$

where $\boldsymbol{x} = (X_1, \dots, X_k)$. We also show that the algorithm incurs $O(M|E|(\mathrm{IO}+k\mathrm{EO}))$ computation cost, where M is the size of a maximal optimal solution, and IO and EO represent the time for the independence oracle of the matroid and the evaluation oracle of the k-submodular function, respectively. As in Section 7.2, all maximal optimal solutions for problem (7.1) have equal size, and we denote the size by M throughout this chapter.

The rest of this chapter is organized as follows. Section 7.2 reviews some basics of k-submodular functions and matroids. Section 7.3 discusses the greedy algorithm for problem (7.1) and proves the 1/2-approximation. Section 7.4 observes why it is hard to apply the existing techniques shown in [Iwata *et al.*, 2016], which are used to prove $k/(2k-1)$-approximation for the unconstrained case, to the case with a matroid constraint. We conclude this chapter in Section 7.5.

7.2 Notation and Definitions

We elucidate some properties of a k-submodular function f where $k \in \mathbb{Z}_{>0}$. For $\boldsymbol{x} = (X_1, \dots, X_k)$ and $\boldsymbol{y} = (Y_1, \dots, Y_k)$ in $(k+1)^E$, we define a partial order $\preceq$ such that $\boldsymbol{x} \preceq \boldsymbol{y}$ if $X_i \subseteq Y_i$ for all $i \in [k]$. For $\boldsymbol{x}, \boldsymbol{y} \in (k+1)^E$ satisfying $\boldsymbol{x} \preceq \boldsymbol{y}$, we use $\boldsymbol{x} \prec \boldsymbol{y}$ if $X_i \subsetneq Y_i$ holds for some $i \in [k]$. We also define

$$\Delta_{e,i} f(\boldsymbol{x}) \coloneqq f(X_1, \dots, X_i \cup \{e\}, \dots, X_k) - f(X_1, \dots, X_k)$$

for $\boldsymbol{x} \in (k+1)^E$, $e \notin \bigcup_{\ell \in [k]} X_\ell$ and $i \in [k]$, which is a marginal gain when adding $e \in E$ to the i-th set of $\boldsymbol{x} \in (k+1)^E$. It is not hard to see that the k-submodularity implies the *orthant submodularity* [Ward and Živný, 2016]

$$\Delta_{e,i} f(\boldsymbol{x}) \geq \Delta_{e,i} f(\boldsymbol{y})$$

for any $\boldsymbol{x}, \boldsymbol{y} \in (k+1)^E$ with $\boldsymbol{x} \preceq \boldsymbol{y}$, $e \notin \bigcup_{j \in [k]} Y_j$, and $i \in [k]$, and the *pairwise monotonicity*

$$\Delta_{e,i} f(\boldsymbol{x}) + \Delta_{e,j} f(\boldsymbol{x}) \geq 0$$

for any $\boldsymbol{x} \in (k+1)^E$, $e \notin \bigcup_{\ell \in [k]} X_\ell$, and $i, j \in [k]$ with $i \neq j$. Actually, these properties characterize k-submodular functions:

Theorem 8 (Ward and Živný [2016])**.** *A function $f : (k+1)^E \to \mathbb{R}$ is k-submodular if and only if f is orthant submodular and pairwise monotone.*

For notational ease, we identify $(k+1)^E$ with $\{0, 1, \ldots, k\}^E$, that is, we associate $(X_1, \ldots, X_k) \in (k+1)^E$ with $\boldsymbol{x} \in \{0, 1, \ldots, k\}^E$ by $X_i = \{e \in E \mid \boldsymbol{x}(e) = i\}$ for $i \in [k]$, where $\boldsymbol{x}(e)$ is the entry of $\boldsymbol{x}$ whose index corresponds to $e \in E$. We sometimes abuse the notation, and simply write $\boldsymbol{x} = (X_1 \ldots, X_k)$ by regarding a vector $\boldsymbol{x}$ as k disjoint subsets of E. For $\boldsymbol{x} \in \{0, 1, \ldots, k\}^E$, we define $\mathrm{supp}(\boldsymbol{x}) \coloneqq \{e \in E \mid \boldsymbol{x}(e) \neq 0\}$; the size of $\boldsymbol{x}$ can be written as $|\mathrm{supp}(\boldsymbol{x})|$. Let $\mathbf{0}$ be the zero vector in $\{0, 1, \ldots, k\}^E$. In what follows, we assume monotone k-submodular functions f to be normalized, i.e., $f(\mathbf{0}) = 0$; if $f(\mathbf{0}) \neq 0$, we redefine $f(\boldsymbol{x}) \coloneqq f(\boldsymbol{x}) - f(\mathbf{0})$ where $\boldsymbol{x} \in (k+1)^E$.

We now turn to some properties of matroid $(E, \mathcal{F})$. An independent set $A \in \mathcal{F}$ is called a *basis* if it is a maximal independent set. We denote the set of all bases by $\mathcal{B}$. It is known that each element in $\mathcal{B}$ has the same size (see, e.g., [Korte and Vygen, 2012, Theorem 13.5]); the size is denoted by M in what follows. Thus, we have the following lemma for the size of the maximal optimal solutions for problem (7.1).

Lemma 8. *The size of any maximal optimal solution for problem* (7.1) *is M.*

Proof. Assume there is a maximal optimal solution $\boldsymbol{o}$ such that $|\mathrm{supp}(\boldsymbol{o})| < M$. Let $\boldsymbol{x} \in (k+1)^E$ be an arbitrary vector such that $\mathrm{supp}(\boldsymbol{x}) \in \mathcal{B}$. Then, by **(M3)**, there exists $e \in \mathrm{supp}(\boldsymbol{x}) \backslash \mathrm{supp}(\boldsymbol{o})$ such that $\mathrm{supp}(\boldsymbol{o}) \cup \{e\} \in \mathcal{F}$. Since f is monotone, by assigning arbitrary $i \in [k]$ to $\boldsymbol{o}(e)$, we get $\Delta_{e,i} f(\boldsymbol{o}) \geq 0$; more precisely, $\Delta_{e,i} f(\boldsymbol{o}) = 0$ since $\boldsymbol{o}$ is an optimal solution. This contradicts the assumption that $\boldsymbol{o}$ is a maximal optimal solution. □

We also introduce the following lemma for later use.

Lemma 9. *Suppose $A \in \mathcal{F}$ and $B \in \mathcal{B}$ satisfy $A \subsetneq B$. Then, for any $e \notin A$ satisfying $A \cup \{e\} \in \mathcal{F}$, there exists $e' \in B \backslash A$ such that $\{B \backslash \{e'\}\} \cup \{e\} \in \mathcal{B}$.*

Proof. The case $e \in B$ is trivial as one can take $e' = e$ and then $\{B \backslash \{e'\}\} \cup \{e\} = B \in \mathcal{B}$. Thus we suppose $e \notin B$ in what follows.

If $|B| - |A| = 1$, by defining $e' = B \backslash A$, we get $\{B \backslash \{e'\}\} \cup \{e\} = A \cup \{e\} \in \mathcal{F}$. Since $|A \cup \{e\}| = |B|$, we have $\{B \backslash \{e'\}\} \cup \{e\} \in \mathcal{B}$.

If $|B| - |A| \geq 2$, then $|A \cup \{e\}| < |B|$. Thus, by applying **(M3)** iteratively, we can obtain $|B| - |A| - 1$ elements $e_1, \ldots, e_{|B|-|A|-1} \in B \backslash \{A \cup \{e\}\}$ such that

$$\{A \cup \{e\}\} \cup \{e_1\} \cup \cdots \cup \{e_{|B|-|A|-1}\} \in \mathcal{B}.$$

Therefore, defining $e' = B \backslash \{A \cup \{e_1\} \cup \cdots \cup \{e_{|B|-|A|-1}\}\}$, we get

$$\{B \backslash \{e'\}\} \cup \{e\} = \{A \cup \{e\}\} \cup \{e_1\} \cup \cdots \cup \{e_{|B|-|A|-1}\} \in \mathcal{B}.$$

This completes the proof. □

7.3 Main Results

We first present a greedy algorithm for problem (7.1); it runs in $O(M|E|(\mathrm{IO} + k\mathrm{EO}))$ time where IO and EO stand for the time for the independence oracle of matroid and the evaluation oracle of k-submodular function, respectively. We then prove that the greedy algorithm outputs a 1/2-approximate solution for problem (7.1). In summary, this section proves the following theorem:

Theorem 9. *For problem (7.1), a 1/2-approximate solution can be obtained with a greedy algorithm that runs in $O(M|E|(\mathrm{IO} + k\mathrm{EO}))$ time.*

7.3.1 Greedy Algorithm and Complexity Analysis

For a given vector $\boldsymbol{s} \in (k+1)^E$, we define

$$E(\boldsymbol{s}) \coloneqq \{e \in E \backslash \mathrm{supp}(\boldsymbol{s}) \mid \mathrm{supp}(\boldsymbol{s}) \cup \{e\} \in \mathcal{F}\},$$

where $\mathcal{F}$ is the independent set of problem (7.1). We now consider applying Algorithm 12 to problem (7.1).

Algorithm 12 A greedy algorithm for k-submodular maximization with a matroid constraint

Require: a monotone k-submodular function $f : (k+1)^E \to \mathbb{R}_{\geq 0}$ and a matroid $(E, \mathcal{F})$, which are given by the evaluation and independence oracles, respectively.

Ensure: a vector $\boldsymbol{s}$ satisfying $\mathrm{supp}(\boldsymbol{s}) \in \mathcal{B}$.

1: $\boldsymbol{s} \leftarrow \boldsymbol{0}$.
2: **for** $j = 1$ to M **do**
3: Construct $E(\boldsymbol{s})$ using the independence oracle.
4: $(e, i) \leftarrow \arg\max_{e \in E(\boldsymbol{s}), i \in [k]} \Delta_{e,i} f(\boldsymbol{s})$.
5: $\boldsymbol{s}(e) \leftarrow i$.
6: **end for**
7: **return** $\boldsymbol{s}$.

We make a remark on using Algorithm 12 in practice. In Step 2, the algorithm requires the value of M, the size of a maximal independent set. However, in practice, we need not calculate the value of M beforehand. Instead, we continue the iteration while $E(\boldsymbol{s})$ is nonempty, which we check in Step 3. We can confirm that this modification does not change the output as follows. As long as $|\mathrm{supp}(\boldsymbol{s})| < M$, exactly one element is added to $\mathrm{supp}(\boldsymbol{s})$ at each iteration due to the monotonicity and **(M3)**, and, if $|\mathrm{supp}(\boldsymbol{s})| = M$, the iteration stops since $\mathrm{supp}(\boldsymbol{s})$ is a maximal independent set. Algorithm 12 is described using M to make it easy to understand in the subsequent discussions. Note that, defining $\boldsymbol{s}^{(j)}$ as the solution obtained after the j-th iteration, we have $|\mathrm{supp}(\boldsymbol{s}^{(j)})| = j$ for $j \in [M]$.

We now examine the time complexity of Algorithm 12. Let EO be the time for the evaluation oracle of the k-submodular function f, and IO be the time for the independence oracle of the matroid $(E, \mathcal{F})$. At the j-th iteration, the independence oracle is used at most $|E|$ times in Step 3, and the evaluation oracle is used at most $k|E|$ times in Step 4. Thus, the time complexity of Algorithm 12 is given by $O(M|E|(\mathrm{IO} + k\mathrm{EO}))$.

7.3.2 Proof for 1/2-approximation

We now prove that Algorithm 12 gives a 1/2-approximate solution for problem (7.1). To prove this, we define a sequence of vectors $\boldsymbol{o}^{(0)}, \boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(M)}$ as in [Ohsaka and Yoshida, 2015; Ward and Živný, 2016; Iwata *et al.*, 2016].

Let $(e^{(j)}, i^{(j)})$ be the pair chosen greedily at the j-th iteration, and $\boldsymbol{s}^{(j)}$ be the solution after the j-th iteration; we let $\boldsymbol{s}^{(0)} = \mathbf{0}$ and $\boldsymbol{s} = \boldsymbol{s}^{(M)}$ be the output of Algorithm 12. We also define $S^{(j)} \coloneqq \mathrm{supp}(\boldsymbol{s}^{(j)})$ for each $j \in [M]$. Let $\boldsymbol{o}$ be a maximal optimal solution. In what follows, we show how to construct a sequence of vectors $\boldsymbol{o}^{(0)} = \boldsymbol{o}, \boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(M-1)}, \boldsymbol{o}^{(M)} = \boldsymbol{s}$ satisfying the following conditions, where $O^{(j)} \coloneqq \mathrm{supp}(\boldsymbol{o}^{(j)})$ for each $j \in [M]$:

$$\boldsymbol{s}^{(j)} \prec \boldsymbol{o}^{(j)} \text{ if } j = 0, 1, \ldots, M-1, \text{ and } \boldsymbol{s}^{(j)} = \boldsymbol{o}^{(j)} = \boldsymbol{s} \text{ if } j = M. \quad (7.2)$$

$$O^{(j)} \in \mathcal{B} \text{ for } j = 0, 1, \ldots, M. \quad (7.3)$$

Below we see how to obtain $\boldsymbol{o}^{(j)}$ from $\boldsymbol{o}^{(j-1)}$ satisfying (7.2) and (7.3). Note that $\boldsymbol{s}^{(0)} = \mathbf{0}$ and $\boldsymbol{o}^{(0)} = \boldsymbol{o}$ satisfy (7.2) and (7.3).

We assume that $\boldsymbol{o}^{(j-1)}$ satisfies

$$\boldsymbol{s}^{(j-1)} \prec \boldsymbol{o}^{(j-1)}, \text{ and } O^{(j-1)} \in \mathcal{B},$$

and we consider obtaining $\boldsymbol{o}^{(j)}$ from $\boldsymbol{o}^{(j-1)}$. Since $\boldsymbol{s}^{(j-1)} \prec \boldsymbol{o}^{(j-1)}$ means $S^{(j-1)} \subsetneq O^{(j-1)}$, and $e^{(j)}$ is chosen to satisfy $S^{(j-1)} \cup \{e^{(j)}\} \in \mathcal{F}$, we see from Lemma 9 that there exists $e' \in O^{(j-1)} \backslash S^{(j-1)}$ satisfying $\{O^{(j-1)} \backslash \{e'\}\} \cup \{e^{(j)}\} \in \mathcal{B}$. We let $o^{(j)} = e'$ and define $\boldsymbol{o}^{(j-1/2)}$ as the vector obtained by assigning 0 to the $o^{(j)}$-th element of $\boldsymbol{o}^{(j-1)}$. We then define $\boldsymbol{o}^{(j)}$ as the vector obtained from $\boldsymbol{o}^{(j-1/2)}$ by assigning $i^{(j)}$ to the $e^{(j)}$-th element. The vector thus constructed, $\boldsymbol{o}^{(j)}$, satisfies

$$O^{(j)} = \{O^{(j-1)} \backslash \{o^{(j)}\}\} \cup \{e^{(j)}\} \in \mathcal{B}. \quad (7.4)$$

Furthermore, since $\boldsymbol{o}^{(j-1/2)}$ satisfies

$$\boldsymbol{s}^{(j-1)} \preceq \boldsymbol{o}^{(j-1/2)},$$

we have the following property for $\boldsymbol{o}^{(j)}$:

$$\boldsymbol{s}^{(j)} \prec \boldsymbol{o}^{(j)} \text{ if } j = 1, \ldots, M-1, \text{ and } \boldsymbol{s}^{(j)} = \boldsymbol{o}^{(j)} = \boldsymbol{s} \text{ if } j = M, \quad (7.5)$$

where the strictness of the inclusion for $j \in [M-1]$ can be easily confirmed from $|S^{(j)}| = j < M = |O^{(j)}|$. Applying the above discussion for $j = 1, \ldots, M$ iteratively, we see from (7.4) and (7.5) that the obtained sequence of vectors $\boldsymbol{o}^{(0)}, \boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(M)}$ satisfies (7.2) and (7.3).

We now prove the following inequality for $j \in [M]$:

$$f(\boldsymbol{s}^{(j)}) - f(\boldsymbol{s}^{(j-1)}) \geq f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)}). \quad (7.6)$$

From $S^{(j-1)} \subsetneq O^{(j-1)}$ and $o^{(j)} \in O^{(j-1)} \backslash S^{(j-1)}$, we get $S^{(j-1)} \cup \{o^{(j)}\} \subseteq O^{(j-1)} \in \mathcal{B}$ for each $j \in [M]$. Thus we obtain the following inclusion from **(M2)** for each $j \in [M]$:

$$S^{(j-1)} \cup \{o^{(j)}\} \in \mathcal{F}.$$

Hence, from $o^{(j)} \notin S^{(j-1)}$, we get $o^{(j)} \in E(\boldsymbol{s}^{(j-1)})$. Therefore, for the pair $(e^{(j)}, i^{(j)})$, which is chosen greedily, we have

$$\Delta_{e^{(j)},i^{(j)}} f(\boldsymbol{s}^{(j-1)}) \geq \Delta_{o^{(j)},\boldsymbol{o}^{(j-1)}(o^{(j)})} f(\boldsymbol{s}^{(j-1)}). \tag{7.7}$$

Furthermore, since $\boldsymbol{s}^{(j-1)} \preceq \boldsymbol{o}^{(j-1/2)}$ holds, orthant submodularity implies

$$\Delta_{o^{(j)},\boldsymbol{o}^{(j-1)}(o^{(j)})} f(\boldsymbol{s}^{(j-1)}) \geq \Delta_{o^{(j)},\boldsymbol{o}^{(j-1)}(o^{(j)})} f(\boldsymbol{o}^{(j-1/2)}). \tag{7.8}$$

Using (7.7) and (7.8), we get (7.6) as follows:

$$\begin{aligned}
f(\boldsymbol{s}^{(j)}) - f(\boldsymbol{s}^{(j-1)}) =& \Delta_{e^{(j)},i^{(j)}} f(\boldsymbol{s}^{(j-1)}) \\
\geq& \Delta_{o^{(j)},\boldsymbol{o}^{(j-1)}(o^{(j)})} f(\boldsymbol{s}^{(j-1)}) \\
\geq& \Delta_{o^{(j)},\boldsymbol{o}^{(j-1)}(o^{(j)})} f(\boldsymbol{o}^{(j-1/2)}) \\
\geq& \Delta_{o^{(j)},\boldsymbol{o}^{(j-1)}(o^{(j)})} f(\boldsymbol{o}^{(j-1/2)}) - \Delta_{e^{(j)},i^{(j)}} f(\boldsymbol{o}^{(j-1/2)}) \\
=& f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)}),
\end{aligned}$$

where the third inequality is due to the monotonicity: $\Delta_{e^{(j)},i^{(j)}} f(\boldsymbol{o}^{(j-1/2)}) \geq 0$.

By (7.6), we have

$$\begin{aligned}
f(\boldsymbol{o}) - f(\boldsymbol{s}) &= \sum_{j=1}^{M} (f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)})) \\
&\leq \sum_{j=1}^{M} (f(\boldsymbol{s}^{(j)}) - f(\boldsymbol{s}^{(j-1)})) = f(\boldsymbol{s}) - f(\boldsymbol{0}) = f(\boldsymbol{s}),
\end{aligned}$$

which means $f(\boldsymbol{s}) \geq f(\boldsymbol{o})/2$.

7.4 Further Discussion

The above 1/2-approximation is the best possible for large k as confirmed by the hardness result of [Iwata *et al.*, 2016]. On the other hand, they also proved

Algorithm 13 An algorithm framework for the unconstrained case

Require: a monotone k-submodular function $f : (k+1)^E \to \mathbb{R}_{\geq 0}$.
Ensure: a vector $\boldsymbol{s}$.
1: $\boldsymbol{s} \leftarrow \mathbf{0}$.
2: **for** each $e \in E$ **do**
3: Set probability distribution p_i for $i \in [k]$.
4: Choose $\boldsymbol{s}(e) \in [k]$ randomly, with $\Pr[\boldsymbol{s}(e) = i] = p_i$ for $i \in [k]$.
5: **end forreturn** $\boldsymbol{s}$.

that the $k/(2k-1)$-approximation is possible in expectation for unconstrained monotone k-submodular maximization, and thus it seems possible to design an algorithm for problem (7.1) whose approximation ratio is better than $1/2$ by using the framework of [Iwata *et al.*, 2016]. Unfortunately, however, it is not so easy; below we elucidate the difficulties.

First, we review the existing techniques. Algorithm 13 is the framework for algorithms presented in [Ward and Živný, 2016; Iwata *et al.*, 2016]. As in Section 7.3.2, the proofs in [Ward and Živný, 2016; Iwata *et al.*, 2016] are done by constructing a sequence of vectors $\boldsymbol{o}^{(0)} = \boldsymbol{o}, \boldsymbol{o}^{(1)}, \ldots, \boldsymbol{o}^{(|E|)} = \boldsymbol{s}$, which is defined as follows. Let $e^{(j)}$ be the element considered at the j-th iteration and $\boldsymbol{s}^{(j)}$ be the solution obtained after the j-th iteration ($\boldsymbol{s}^{(0)} = \mathbf{0}$). We define $\boldsymbol{o}^{(j-1/2)}$ from $\boldsymbol{o}^{(j-1)}$ as the vector obtained by assigning 0 to $\boldsymbol{o}^{(j-1)}(e^{(j)})$, and then we define $\boldsymbol{o}^{(j)}$ from $\boldsymbol{o}^{(j-1/2)}$ by assigning $\boldsymbol{s}^{(j)}(e^{(j)})$ to $\boldsymbol{o}^{(j-1/2)}(e^{(j)})$. Note that, in the unconstrained case, $\boldsymbol{o}^{(j)}$ can be obtained from $\boldsymbol{o}^{(j-1)}$ by changing only the $e^{(j)}$-th element. For notational ease, we define $y_i^{(j)} \coloneqq \Delta_{e^{(j)},i} f(\boldsymbol{s}^{(j-1)})$ and $a_i^{(j)} \coloneqq \Delta_{e^{(j)},i} f(\boldsymbol{o}^{(j-1/2)})$. The following lemma is crucial in the proof for the unconstrained case.

Lemma 10 (Iwata *et al.* [2016])**.** *Let $c \in \mathbb{R}_{\geq 0}$ and $p_i^{(j)}$ be the probability distribution set at the j-th iteration for $i \in [k]$. Conditioning on $\boldsymbol{s}^{(j-1)}$, suppose that*

$$\sum_{i \in [k]} \left(a_{i^*}^{(j)} - a_i^{(j)}\right) p_i^{(j)} \leq c \left(\sum_{i \in [k]} y_i^{(j)} p_i^{(j)} \right) \tag{7.9}$$

holds for each $j \in \{1, \ldots, |E|\}$, where $i^* = \boldsymbol{o}^{(j-1)}(e^{(j)})$. Then Algorithm 13 returns a solution $\boldsymbol{s}$ satisfying $\mathbb{E}[f(\boldsymbol{s})] \geq \frac{1}{1+c} f(\boldsymbol{o})$.

The left-hand side and right-hand side of (7.9) are actually equivalent to

$\mathbb{E}[f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)})]$ and $\mathbb{E}[f(\boldsymbol{s}^{(j)}) - f(\boldsymbol{s}^{(j-1)})]$, respectively, and so

$$\mathbb{E}[f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)})] \leq c\mathbb{E}[f(\boldsymbol{s}^{(j)}) - f(\boldsymbol{s}^{(j-1)})]$$

holds from (7.9), which plays a role similar to (7.6) in the proof of Lemma 10. Iwata *et al.* [2016] proved that (7.9) holds for $c = 1 - 1/k$ by designing $p_i^{(j)}$ $(i \in [k])$ appropriately, which leads to the $k/(2k-1)$-approximation thanks to Lemma 10.

We now see the difference between the unconstrained problem and problem (7.1). To show (7.9) with $c = 1 - 1/k$ for the unconstrained case, Iwata *et al.* [2016] bounded the left-hand side of (7.9) from above as follows:

$$\begin{aligned}\sum_{i\in[k]} \big(a_{i^*}^{(j)} - a_i^{(j)}\big)p_i^{(j)} &= \sum_{i\neq i^*} \big(a_{i^*}^{(j)} - a_i^{(j)}\big)p_i^{(j)} \\ &\leq \frac{1}{\gamma} \cdot \gamma y_{i^*}^{(j)} \sum_{i\neq i^*} p_i^{(j)} \\ &\leq \frac{1}{\gamma}\Bigg((1-\theta)(\gamma y_{i^*}^{(j)})^{\frac{1}{1-\theta}} + \theta\Big(\sum_{i\neq i^*} p_i^{(j)}\Big)^{\frac{1}{\theta}}\Bigg),\end{aligned} \tag{7.10}$$

where $\gamma > 0$ and $\theta \in (0,1)$. The first inequality comes from $a_i^{(j)} \geq 0$ and $y_{i^*}^{(j)} \geq a_{i^*}^{(j)}$, and the second is by the weighted AM-GM inequality. Actually, as in (7.10), separating the term indexed by i^* from the others indexed by $i \in [k]\backslash\{i^*\}$ is the key to proving (7.9) with $c = 1 - 1/k$ (see [Iwata *et al.*, 2016] for details).

Now we turn to applying similar techniques to the case of problem (7.1). At the j-th iteration $(j \in [M])$, we consider choosing element $e^{(j)}$ and letting $i = \boldsymbol{s}^{(j-1)}(e^{(j)})$ with probability $p_i^{(j)}$ $(i \in [k])$ to obtain $\boldsymbol{s}^{(j)}$ (how to choose $e^{(j)}$ is unclear, but we here assume $e^{(j)}$ is chosen somehow). Recall that, as in Section 7.3.2, $\boldsymbol{o}^{(j)}$ is obtained from $\boldsymbol{o}^{(j-1)}$ by assigning 0 and $i^{(j)}$ to $\boldsymbol{o}^{(j-1)}(o^{(j)})$ and $\boldsymbol{o}^{(j-1)}(e^{(j)})$, respectively. Thus, conditioning on $\boldsymbol{s}^{(j-1)}$, $\mathbb{E}[f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)})]$ can be written as follows:

$$\begin{aligned}&\mathbb{E}[f(\boldsymbol{o}^{(j-1)}) - f(\boldsymbol{o}^{(j)})] \\ &= \sum_{i\in[k]} \big(\Delta_{o^{(j)},i^*} f(\boldsymbol{o}^{(j-1/2)}) - \Delta_{e^{(j)},i} f(\boldsymbol{o}^{(j-1/2)})\big)p_i^{(j)},\end{aligned} \tag{7.11}$$

where $i^* = \boldsymbol{o}^{(j-1)}(o^{(j)})$. Note that $o^{(j)} \neq e^{(j)}$ can happen even in the case of a total size constraint. Therefore, it is hard to prove that (7.11) reduces

to the sum of $k-1$ terms as in the first equality of (7.10), and thus we cannot separate the term indexed by i^* from those indexed by $i \in [k]\backslash\{i^*\}$ as in (7.10). This is the cause of the discrepancy between the unconstrained problem and problem (7.1).

From the above observation, one can obtain some clues that may lead to algorithms with better approximation guarantees for problem (7.1). For example, if one could prove that

$$\Delta_{o^{(j)},i^*} f(\boldsymbol{o}^{(j-1/2)}) p_{i^*}^{(j)} \leq \sum_{i\in[k]} \Delta_{e^{(j)},i} f(\boldsymbol{o}^{(j-1/2)}) p_i^{(j)}$$

holds by appropriately choosing $e^{(j)}$ and designing $p_i^{(j)}$ for $i \in [k]$, then (7.11) is bounded from above by the sum of $k-1$ terms as follows:

$$\begin{aligned}
&\sum_{i\in[k]} \big(\Delta_{o^{(j)},i^*} f(\boldsymbol{o}^{(j-1/2)}) - \Delta_{e^{(j)},i} f(\boldsymbol{o}^{(j-1/2)})\big) p_i^{(j)} \\
&\leq \Delta_{o^{(j)},i^*} f(\boldsymbol{o}^{(j-1/2)}) \sum_{i\neq i^*} p_i^{(j)} + \Delta_{o^{(j)},i^*} f(\boldsymbol{o}^{(j-1/2)}) p_{i^*}^{(j)} - \sum_{i\in[k]} \Delta_{e^{(j)},i} f(\boldsymbol{o}^{(j-1/2)}) p_i^{(j)} \\
&\leq \Delta_{o^{(j)},i^*} f(\boldsymbol{o}^{(j-1/2)}) \sum_{i\neq i^*} p_i^{(j)},
\end{aligned}$$

to which the techniques similar to those in [Iwata *et al.*, 2016] may be applicable. Furthermore, instead of considering a single $e^{(j)}$ at each iteration, we can consider setting $e_i^{(j)}$ for each $i \in [k]$ as candidates to be chosen. However, designing a better algorithm from these observation is so intricate as to go beyond the scope of this paper, and thus we leave it as an open problem.

7.5 Conclusion

We proved that a 1/2-approximate solution can be obtained for non-negative monotone k-submodular maximization with a matroid constraint via a greedy algorithm. Our approach follows the techniques shown in [Ohsaka and Yoshida, 2015; Ward and Živný, 2016; Iwata *et al.*, 2016]. The proved approximation ratio is asymptotically tight due to the hardness result shown in [Iwata *et al.*, 2016]. We showed that the proposed algorithm incurs $O(M|E|(\mathrm{IO} + k\mathrm{EO}))$ computation cost. We also observed why it is difficult to apply the existing

techniques shown in [Iwata *et al.*, 2016], which are used to prove $k/(2k-1)$-approximation for the unconstrained problem, to the problem with a matroid constraint.

Chapter 8

Exact SDP Reformulation of Binary Polynomial Optimization via Graph Representation

In this chapter, we consider binary polynomial optimization problems (POPs), which are hard non-convex optimization problems. We show that the problems can be reformulated as convex optimization problems of larger size by utilizing graph representations of original binary POPs. Specifically, for binary POPs of degree d with n variables, we prove that the $\lceil (n+d-1)/2 \rceil$th semidefinite programming (SDP) relaxation in Lasserre's hierarchy provides an exact reformulation. If binary POPs involve only even-degree monomials, we show that it can be further reduced to $\lceil (n+d-2)/2 \rceil$. This upper bound on the relaxation order coincides with the conjecture by Laurent [2003], which was recently proved by Fawzi *et al.* [2015], on binary quadratic optimization problems where $d = 2$. We also numerically confirm that the bounds are tight. More precisely, we present instances of binary POPs that require solving at least the $\lceil (n+d-1)/2 \rceil$th SDP relaxation for general binary POPs and the $\lceil (n+d-2)/2 \rceil$th SDP relaxation for even-degree binary POPs to obtain exact optimal values.

8.1 Introduction

We consider binary polynomial optimization problems (POPs) formulated as

$$\underset{x\in\{-1,1\}^n}{\text{minimize}} \quad f(x), \tag{8.1}$$

where f is a polynomial of degree d and n is the number of variables. In this chapter, we often use vectors as superscripts and subscripts; therefore, for notational brevity, we do not use bold letters to denote vectors.

Binary POPs of form (8.1) arise in various scenarios. For example, the maximum cut problem [Goemans and Williamson, 1995], the maximum stable set problems [Hertz, 1995], and the quadratic assignment problems [Rendl and Wolkowicz, 1997] are formulated as binary quadratic optimization problems (QOPs), which are special binary POPs with $d = 2$. The maximum satisfiability problems [Anjos, 2005] and the maximum weighted independent set problem [He *et al.*, 2013] can be represented as binary POPs.

Since binary POPs have complicated non-convex structures, we often consider relaxing them to convex optimization problems. Lasserre's hierarchy of semidefinite programming (SDP) relaxations [Lasserre, 2001b] is one of the most studied relaxation methods for general POPs including binary POPs (8.1). The hierarchy consists of SDPs whose size increases with a parameter so-called *relaxation order* ω. Intuitively, the larger ω is, the closer the minimum of SDP can be to the optimal value of the original POP. Binary POPs have a finite number of SDP relaxations to be solved in the hierarchy for computing their optimal values. Lasserre [2001a] showed that any binary POP could equivalently be transformed into an SDP with at most $2^n - 1$ variables, or the nth SDP relaxation in the hierarchy ($\omega = n$). Kurpisz *et al.* [2017] presented one of the hardest binary POP instances such that its objective function has degree $d = n$ and the $(n-1)$th SDP relaxation in Lasserre's hierarchy is not tight.

For the case of binary QOPs of form (8.1), Laurent [2003] conjectured that the exact optimal value can be obtained by solving the SDP relaxation with relaxation order $\omega = \lceil n/2 \rceil$ in the hierarchy. Recently, Fawzi *et al.* [2015] proved the conjecture by applying their results related to optimization problems on a finite abelian group to binary QOPs. As regards general binary POP (8.1), while the nth SDP relaxation is sufficient for obtaining an exact reformulation thanks to the result of [Lasserre, 2001a], which means $\omega = n$ is an upper bound, any smaller upper bounds of ω value have not been proposed nor proved.

In this chapter we prove that relaxation order $\omega = \lceil (n+d-1)/2 \rceil$ is sufficient for obtaining the optimal value of (8.1), and it can be reduced to $\lceil (n+d-2)/2 \rceil$ if all monomials of f are even degree. For the binary QOPs where $d = 2$, our result coincides with $\lceil n/2 \rceil$ as shown in [Fawzi *et al.*, 2015]. Since $d \leq n$ always holds for the degree of (8.1) from $x_i^2 = 1$ $(i = 1, \ldots, n)$, we see that $\lceil (n+d-1)/2 \rceil$ is always bounded by n, which is compatible with the result of [Lasserre, 2001a]. In [Kurpisz *et al.*, 2017, Theorem 3], it is shown that, if an instance of binary POP has the integrality gap at $\omega = n-1$, then the degree of the objective function is n. Our upper bound $\omega = \lceil (n+d-1)/2 \rceil$ is also compatible with their result since $\lceil (n+d-1)/2 \rceil = n$ if and only if $d = n$. They also showed that $\omega = n$ is the lower bound of the relaxation order required for the exact SDP relaxation of some binary POPs with $d = n$, which implies that our bound is sharp in the case of $d = n$. More recently, Kurpisz *et al.* [2016] proved that $\omega = \lceil (n + 2\widetilde{d} - 1)/2 \rceil$ is the lower bound of the relaxation order for some binary POPs such that $d = 2\widetilde{d}$ and n is odd. This implies the sharpness of our upper bound $\omega = \lceil (n+d-1)/2 \rceil$ when d is even and n is odd.

Our proof is based on a graph-representation technique, which is an extension of what is used in [Fawzi *et al.*, 2015, Theorem 1D]. Specifically, we construct a chordal cover for the Cayley graph, which is obtained from the monomials of binary POPs (8.1), and then we find its Fourier support. The main idea for finding a chordal cover of the Cayley graph is to *paste cliques* along the complete graphs. We utilize a result from the graph theory, which states that a graph is chordal if and only if, starting from complete graphs, it can be constructed recursively by pasting along complete subgraphs. We then use the chordal cover to construct a Fourier support for the binary POP (8.1). As a result, we obtain the truncated moment matrix that corresponds to the $\lceil (n+d-1)/2 \rceil$th SDP relaxation in the hierarchy.

To empirically observe how large ω is required to obtain exact SDP relaxation for some instances of binary POPs, we conduct numerical experiments with random binary POP instances and binary POP instances whose objective polynomial functions have all-one coefficients. Regarding random instances, we randomly generated 1000 binary POPs for each (n, d) with $n = 3, 4, \ldots, 9$ and $d = 1, 2, \ldots, 9$ and confirmed that $\omega = \lceil (n+d-1)/2 \rceil$ is an upper bound for obtaining the optimal values of the instances. Although the results of all 1000 random instances show discrepancies from our theoretical upper bounds, some of the (even-degree) all-one coefficient instances require

$\omega = \lceil (n+d-1)/2 \rceil$ ($\omega = \lceil (n+d-2)/2 \rceil$) to obtain the exact optimal value. Namely, these results are consistent with the theoretical bounds.

8.1.1 Notation and Symbols

We list the notation and definitions used in this chapter.

- $\mathcal{S}^d \coloneqq \{S \in 2^{[n]} : |S| \leq d\}$.
- For given $S, T \in 2^{[n]}$, $S \Delta T \coloneqq \{S \backslash T\} \cup \{T \backslash S\}$. Note that $2^{[n]}$ forms a group with this operation; the identity element is $\emptyset$ and $S^{-1} = S$ for all $S \in 2^{[n]}$.
- For $\mathcal{S}, \mathcal{T} \subseteq 2^{[n]}$, $\mathcal{S} \Delta \mathcal{T} \coloneqq \{S \Delta T : \forall S \in \mathcal{S}, \forall T \in \mathcal{T}\}$.
- $x^\alpha \coloneqq x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$ for a vector of variables $x \in \mathbb{R}^n$ and a non-negative integer-valued vector $\alpha \in \mathbb{Z}^n_{\geq 0}$. Given set of vectors $\mathcal{D} \subseteq \mathbb{Z}^n_{\geq 0}$, $(x^\alpha)_{\alpha \in \mathcal{D}}$ denotes a $|\mathcal{D}|$-dimensional vector of monomials x^α for all $\alpha \in \mathcal{D}$. Given variables ℓ_δ for all $\delta \in \mathcal{D}$, we let $(\ell_\delta)_{\delta \in \mathcal{D}}$ be a vector of the variables indexed by $\delta \in \mathcal{D}$.
- $\mathcal{G} = (V, E)$ denotes a graph with the vertex set V and the edge set E. If two graphs $\mathcal{G}_1 = (V_1, E_1)$ and $\mathcal{G}_2 = (V_2, E_2)$ are subgraphs of the same graph, then $\mathcal{G}_1 \cup \mathcal{G}_2$ denotes a graph with the vertex set $V_1 \cup V_2$ and the edge set $E_1 \cup E_2$. Similarly, $\mathcal{G}_1 \cap \mathcal{G}_2$ denotes a graph with the vertex set $V_1 \cap V_2$ and the edge set $E_1 \cap E_2$.

Note that any d-dimensional polynomial f is expressed as

$$f(x) = \sum_{\alpha \in \mathbb{Z}^n_{\geq 0}, \|\alpha\|_1 \leq d} c_\alpha x^\alpha,$$

where c_α are coefficients and $\|\alpha\|_1 \coloneqq \sum_{i=1}^n \alpha_i$. Since we have $x \in \{-1, 1\}^n$ as in (8.1), we can assume $\alpha \in \{0, 1\}^n$. Thus, $\|\alpha\|_1$ denotes the number of nonzero elements in α. Note that $d \leq n$ holds and that the set of monomials $\{x^\alpha : \alpha \in \{0, 1\}^n, \|\alpha\|_1 \leq d\}$ forms a basis for real-valued polynomials of degree d on $\{-1, 1\}^n$. Hence, (8.1) can be expressed as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{\alpha \in \{0,1\}^n,\ \|\alpha\|_1 \leq d} c_\alpha x^\alpha \\ \text{subject to} \quad & x \in \{-1, 1\}^n. \end{aligned} \tag{8.2}$$

8.1.2 An Illustrative Example

The following example instance will be repetitively used throughout this chapter to illustrate our discussion.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 1 + x_1 + x_2 + x_3 + x_1x_2 + x_1x_3 + x_2x_3 \\ \text{subject to} \quad & x \in \{-1, 1\}^3. \end{aligned} \tag{8.3}$$

Since $n = 3$ and $d = 2$, we have $[n] = \{1, 2, 3\}$,

$$2^{[3]} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\},$$

and

$$\mathcal{S}^2 = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}.$$

We can apply our result to a binary POP of degree higher than 2, but it is too complicated to display the Cayley graph of such a binary POP and a chordal cover of the Cayley graph. Thus we use (8.3).

8.1.3 Organization

The rest of this chapter is organized as follows: In Section 8.2, we introduce basic concepts necessary for the subsequent discussion such as the group indicator vectors, the moment polytope, the truncated moment matrix, the Cayley graph, and the Fourier support. We use (8.3) to explain the concepts. Section 8.3 provides the main results. We describe the clique-pasting technique, which is used to construct a choral cover and a Fourier support. We also show how to obtain the truncated moment matrix. In Section 8.4, we focus on even-degree binary POPs. The method to obtain the reduced bound $\lceil (n + d - 2)/2 \rceil$ is discussed. In Section 8.5, we experimentally validate the obtained results with randomly generated binary POPs and binary POPs with all-one coefficients. Finally, we conclude this chapter in Section 8.6.

8.2 Preliminaries

We introduce the basics required in the subsequent discussion, and we review the results of [Fawzi *et al.*, 2015] for the case of $x \in \{-1, 1\}^n$.

8.2.1 The Group of Indicator Vectors

For a given $S \in 2^{[n]}$, let $\delta_S \in \{0,1\}^n$ be the indicator vector of S. We define $\mathcal{D} \coloneqq \{\delta_S : S \in 2^{[n]}\}$. Given set $\mathcal{S} \subseteq 2^{[n]}$, we define $\mathcal{D}_\mathcal{S} \coloneqq \{\delta_S : S \in \mathcal{S}\}$. We often use $\delta \in \mathcal{D}$ to express an element of $\mathcal{D}$, abbreviating its index. The product of $\delta_S, \delta_T \in \mathcal{D}$ is defined as

$$\delta_S \delta_T \coloneqq \delta_{S \Delta T}, \quad S, T \in 2^{[n]}.$$

For example, if $\delta_{\{1,2\}} = (1,1,0)$ and $\delta_{\{2,3\}} = (0,1,1)$, then $\delta_{\{1,2\}}\delta_{\{2,3\}} = \delta_{\{1,3\}} = (1,0,1)$. Note that the aforementioned operation returns the exclusive disjunction for each pair of entries. For simplicity, we frequently identify an indicator vector δ with its numeric string of length n, e.g., $\delta_{\{1,2\}} = 110$.

Note that $\mathcal{D}$ forms a group with the multiplication; the identity element is $\delta_\emptyset$ and $\delta^{-1} = \delta$. We also note that $\{x^\delta : \delta \in \mathcal{D}\}$ is the set of all monomials on $\{-1,1\}^n$ and that $\{x^\delta : \delta \in \mathcal{D}_{\mathcal{S}^d}\}$ is the set of monomials on $\{-1,1\}^n$ whose degree is less than or equal to d.

8.2.2 Moment Polytopes and Moment Matrices

If we linearize problem (8.2), we obtain

$$\begin{aligned} \underset{\ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^d}}}{\text{minimize}} \quad & \sum_{\delta \in \mathcal{D}_{\mathcal{S}^d}} c_\delta \ell_\delta \\ \text{subject to} \quad & (\ell_\delta)_{\delta \in \mathcal{D}_{\mathcal{S}^d}} \in \operatorname{conv}\left\{ (x^\delta)_{\delta \in \mathcal{D}_{\mathcal{S}^d}} \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^d}} : x \in \{-1,1\}^n \right\}, \end{aligned} \tag{8.4}$$

where ℓ_δ $(\forall \delta \in \mathcal{D}_{\mathcal{S}^d})$ are decision variables. The feasible region can be characterized by the *moment polytope* defined as follows.

Definition 3. *The moment polytope $\mathcal{M}(\mathcal{S})$ for given $\mathcal{S} \subseteq 2^{[n]}$ is defined as the convex hull of vectors $(x^\delta)_{\delta \in \mathcal{D}_\mathcal{S}}$ for $x \in \{-1,1\}^n$, i.e.,*

$$\mathcal{M}(\mathcal{S}) \coloneqq \operatorname{conv}\left\{ (x^\delta)_{\delta \in \mathcal{D}_\mathcal{S}} \in \mathbb{R}^{\mathcal{D}_\mathcal{S}} : x \in \{-1,1\}^n \right\}.$$

Note that problem (8.4) can be expressed with the moment polytope:

$$\begin{aligned} \underset{\ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^d}}}{\text{minimize}} \quad & \sum_{\delta \in \mathcal{D}_{\mathcal{S}^d}} c_\delta \ell_\delta \\ \text{subject to} \quad & (\ell_\delta)_{\delta \in \mathcal{D}_{\mathcal{S}^d}} \in \mathcal{M}(\mathcal{S}^d). \end{aligned} \tag{8.5}$$

Definition 4. *For a given vector $\ell \in \mathbb{R}^{\mathcal{D}}$, moment matrix $M(\ell) \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}}$ is defined as a matrix whose (δ_S, δ_T)th element is given by $\ell_{\delta_S\delta_T} = \ell_{\delta_S \Delta T}$ for all $\delta_S, \delta_T \in \mathcal{D}$.*

The vector $\ell \in \mathbb{R}^{\mathcal{D}}$ and the moment matrix $M(\ell) \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}}$ are indexed by the elements in $\mathcal{D}$ as shown in the following example.

Example 1. For $n = 3$, the set of indicator vectors is

$$\mathcal{D} = \{000, 100, 010, 001, 110, 101, 011, 111\}.$$

Thus, for a given vector $\ell = (\ell_{000}\ \ell_{100}\ \ell_{010}\ \ell_{001}\ \ell_{110}\ \ell_{101}\ \ell_{011}\ \ell_{111})^\top \in \mathbb{R}^{\mathcal{D}}$, the moment matrix $M(\ell) \in \mathbb{R}^{\mathcal{D}\times\mathcal{D}}$ is defined as follows:

$$M(\ell) \coloneqq \begin{bmatrix} \ell_{000} & \ell_{100} & \ell_{010} & \ell_{001} & \ell_{110} & \ell_{101} & \ell_{011} & \ell_{111} \\ \ell_{100} & \ell_{000} & \ell_{110} & \ell_{101} & \ell_{010} & \ell_{001} & \ell_{111} & \ell_{011} \\ \ell_{010} & \ell_{110} & \ell_{000} & \ell_{011} & \ell_{100} & \ell_{111} & \ell_{001} & \ell_{101} \\ \ell_{001} & \ell_{101} & \ell_{011} & \ell_{000} & \ell_{111} & \ell_{100} & \ell_{010} & \ell_{110} \\ \ell_{110} & \ell_{010} & \ell_{100} & \ell_{111} & \ell_{000} & \ell_{011} & \ell_{101} & \ell_{001} \\ \ell_{101} & \ell_{001} & \ell_{111} & \ell_{100} & \ell_{011} & \ell_{000} & \ell_{110} & \ell_{010} \\ \ell_{011} & \ell_{111} & \ell_{001} & \ell_{010} & \ell_{101} & \ell_{110} & \ell_{000} & \ell_{100} \\ \ell_{111} & \ell_{011} & \ell_{101} & \ell_{110} & \ell_{001} & \ell_{010} & \ell_{100} & \ell_{000} \end{bmatrix}.$$

Definition 5. *Let $\mathcal{T} \subseteq 2^{[n]}$. For a given vector $\ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{T}\Delta\mathcal{T}}}$, truncated moment matrix $M_{\mathcal{T}}(\ell) \in \mathbb{R}^{\mathcal{D}_{\mathcal{T}}\times\mathcal{D}_{\mathcal{T}}}$ is defined as a matrix whose (δ_S, δ_T)th element is given by $\ell_{\delta_S\delta_T} = \ell_{\delta_S\Delta T}$ for all $S, T \in \mathcal{T}$.*

Example 2. If $\mathcal{T} \subseteq 2^{[3]}$ is given by

$$\mathcal{T} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\},$$

then

$$\mathcal{T}\Delta\mathcal{T} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}.$$

Hence

$$\mathcal{D}_{\mathcal{T}} = \{000, 100, 010, 001, 110, 101, 011\}$$

and

$$\mathcal{D}_{\mathcal{T}\Delta\mathcal{T}} = \{000, 100, 010, 001, 110, 101, 011, 111\}.$$

Thus, for a given vector $\ell = (\ell_{000}\ \ell_{100}\ \ell_{010}\ \ell_{001}\ \ell_{110}\ \ell_{101}\ \ell_{011}\ \ell_{111})^\top \in \mathbb{R}^{\mathcal{D}_\mathcal{T} \triangle \mathcal{T}}$, the truncated moment matrix $M_\mathcal{T}(\ell) \in \mathbb{R}^{\mathcal{D}_\mathcal{T} \times \mathcal{D}_\mathcal{T}}$ is defined as follows:

$$M_\mathcal{T}(\ell) \coloneqq \begin{bmatrix} \ell_{000} & \ell_{100} & \ell_{010} & \ell_{001} & \ell_{110} & \ell_{101} & \ell_{011} \\ \ell_{100} & \ell_{000} & \ell_{110} & \ell_{101} & \ell_{010} & \ell_{001} & \ell_{111} \\ \ell_{010} & \ell_{110} & \ell_{000} & \ell_{011} & \ell_{100} & \ell_{111} & \ell_{001} \\ \ell_{001} & \ell_{101} & \ell_{011} & \ell_{000} & \ell_{111} & \ell_{100} & \ell_{010} \\ \ell_{110} & \ell_{010} & \ell_{100} & \ell_{111} & \ell_{000} & \ell_{011} & \ell_{101} \\ \ell_{101} & \ell_{001} & \ell_{111} & \ell_{100} & \ell_{011} & \ell_{000} & \ell_{110} \\ \ell_{011} & \ell_{111} & \ell_{001} & \ell_{010} & \ell_{101} & \ell_{110} & \ell_{000} \end{bmatrix}.$$

Notice that for the given moment matrix $M(\ell)$, the truncated moment matrix $M_\mathcal{T}(\ell)$ is the submatrix of $M(\ell)$ such that the rows and columns are obtained from $M(\ell)$ according to $\mathcal{D}_\mathcal{T}$.

The moment polytope is characterized by the moment matrix as in the following proposition. For the proof, we refer the reader to [Fawzi *et al.*, 2015].

Proposition 4. *The moment polytope with respect to $\mathcal{S} \subseteq 2^{[n]}$ can be expressed as follows:*

$$\mathcal{M}(\mathcal{S}) = \{\ell \in \mathbb{R}^{\mathcal{D}_\mathcal{S}} : \exists y \in \mathbb{R}^{\mathcal{D}} \text{ s.t. } y_\delta = \ell_\delta \text{ for all } \delta \in \mathcal{D}_\mathcal{S},\ y_{\delta_\emptyset} = 1,\ M(y) \succeq 0\}.$$

8.2.3 Representing the Moment Polytope with the Truncated Moment Matrix

In Proposition 4, the moment polytope is represented with the moment matrix of size $|\mathcal{D}| \times |\mathcal{D}|$. To describe how the moment polytope can be expressed by a smaller-size truncated moment matrix, we briefly review the techniques of [Fawzi *et al.*, 2015, Theorem 1D]. We note that the *chordal completion theorem* (see, e.g., [Bapat, 2014, §12.3]) plays an important role in the proof of their theorem.

The graph $\mathcal{G} = (V, E)$ is *chordal* if any cycle of length at least four has a chord, which connects non-adjacent vertices. A *chordal cover* of $\mathcal{G}$ is a graph $\mathcal{G}' = (V, E')$ such that $E \subseteq E'$ and $\mathcal{G}'$ is chordal. A subset $\mathcal{C} \subseteq V$ is a *clique* in $\mathcal{G}$ if $\{i, j\} \in E$ for all $i, j \in \mathcal{C}$ such that $i \neq j$. The clique $\mathcal{C}$ is called *maximal* in $\mathcal{G}$ if it is not a strict subset of another clique $\mathcal{C}'$ in $\mathcal{G}$.

For a given set $\mathcal{S} \subseteq 2^{[n]}$, we define the *Cayley graph* $\mathrm{Cay}(\mathcal{S})$ as follows: The vertex set is $2^{[n]}$ and two distinct vertices $S, T \in 2^{[n]}$ are connected by an

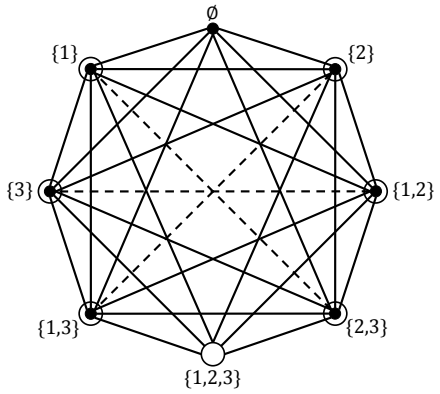


Figure 8.1: For the case $n = 3$ and $\mathcal{S}^2$, the solid lines represent the edges of the Cayley graph $\mathrm{Cay}(\mathcal{S}^2)$, and its chordal cover Γ is obtained by adding the dashed lines to the edge set. Note that Γ does not contain the edge between $\emptyset$ and $\{1,2,3\}$. The vertex set $\mathcal{T} = \{T \in 2^{[3]} : |T| \leq 2\}$ is the Fourier support (indicated by the filled circles) of Γ since, for all maximal cliques, $\mathcal{C}_0 = \{S \in 2^{[3]} : |S| \leq 2\}$ (indicated by the filled circles) and $\mathcal{C}_1 = \{S \in 2^{[3]} : |S| \geq 1\}$ (indicated by the open circles) of Γ, we have $\emptyset \Delta \mathcal{C}_0 \subseteq \mathcal{T}$ and $\{1,2,3\}\Delta\mathcal{C}_1 \subseteq \mathcal{T}$.

edge if and only if $S\Delta T \in \mathcal{S}$. Given graph Γ with vertex set $2^{[n]}$, $\mathcal{T} \subseteq 2^{[n]}$ is called a *Fourier support* of Γ if, for any maximal clique $\mathcal{C} \subseteq 2^{[n]}$ of Γ, there exists a node $S_{\mathcal{C}} \in 2^{[n]}$ such that $S_{\mathcal{C}}\Delta\mathcal{C} \subseteq \mathcal{T}$.

Example 3. For $n = 3$ and $\mathcal{S}^2 = \{S \in 2^{[3]} : |S| \leq 2\}$, Figure 8.1 displays the Cayley graph $\mathrm{Cay}(\mathcal{S}^2)$, the chordal cover Γ, and the Fourier support $\mathcal{T}$ of Γ. $\mathrm{Cay}(\mathcal{S}^2)$ consists of all eight vertices of $2^{[3]}$ and edges shown by solid lines. The graph obtained by adding three dashed lines to $\mathrm{Cay}(\mathcal{S}^2)$ is a chordal cover Γ of $\mathrm{Cay}(\mathcal{S}^2)$. The vertex set $\mathcal{T} = \{T \in 2^{[3]} : |T| \leq 2\}$ is a Fourier support of Γ. Indeed, the chordal graph Γ has two maximal cliques $\mathcal{C}_0 = \{S \in 2^{[3]} : |S| \leq 2\}$ and $\mathcal{C}_1 = \{S \in 2^{[3]} : |S| \geq 1\}$, for which $S_{\mathcal{C}_0} = \emptyset$ and $S_{\mathcal{C}_1} = \{1,2,3\}$ satisfy $S_{\mathcal{C}_0}\Delta\mathcal{C}_0 \subseteq \mathcal{T}$ and $S_{\mathcal{C}_1}\Delta\mathcal{C}_1 \subseteq \mathcal{T}$, respectively.

Now we describe [Fawzi *et al.*, 2015, Theorem 1D], which enables us to represent the moment polytope with an appropriately truncated moment matrix. That is, the feasible set of problem (8.5) can be expressed with a smaller moment matrix. For the proof, see [Fawzi *et al.*, 2015].

Theorem 10 (Fawzi *et al.* [2015])**.** *Let $\mathcal{S}$ be a given subset of $2^{[n]}$. If $\mathrm{Cay}(\mathcal{S})$ has a chordal cover Γ with Fourier support $\mathcal{T} \subseteq 2^{[n]}$, then the moment polytope can be expressed as follows:*

$$\mathcal{M}(\mathcal{S}) = \left\{ \ell \in \mathbb{R}^{\mathcal{D}_\mathcal{S}} \ : \ \exists y \in \mathbb{R}^{\mathcal{D}_\mathcal{T} \Delta \mathcal{T}} \ \text{s.t.} \ \begin{array}{l} y_\delta = \ell_\delta \text{ for all } \delta \in \mathcal{D}_\mathcal{S}, \\ y_{\delta_\emptyset} = 1, \text{ and } M_\mathcal{T}(y) \succeq 0 \end{array} \right\}.$$

8.3 Truncation of the Moment Matrix for Binary POPs

Remember that $\mathcal{M}(\mathcal{S}^d)$ in Definition 3 provides the feasible set of problem (8.5). In what follows, we focus on finding an appropriate Fourier support $\mathcal{T}$ of a chordal graph Γ that covers $\mathrm{Cay}(\mathcal{S}^d)$. If such $\mathcal{T}$ is obtained, then we can characterize the moment polytope $\mathcal{M}(\mathcal{S}^d)$ with a truncated moment matrix $M_\mathcal{T}(y)$ thanks to Theorem 10.

8.3.1 Constructing a Chordal Cover

If $\mathcal{G}$ is a graph with subgraphs $\mathcal{G}_1, \mathcal{G}_2$ such that $\mathcal{G} = \mathcal{G}_1 \cup \mathcal{G}_2$, we say that $\mathcal{G}$ arises from $\mathcal{G}_1$ and $\mathcal{G}_2$ by *pasting* these graphs together along $\mathcal{H} \coloneqq \mathcal{G}_1 \cap \mathcal{G}_2$. We have the following proposition on constructing a chordal graph (see, [Diestel, 2000, Proposition 5.5.1] for the proof).

Proposition 5. *A graph is chordal if and only if it can be constructed recursively by pasting along complete subgraphs, starting from complete graphs.*

For example, in Figure 8.1, the chordal cover Γ is obtained by pasting the two complete graphs $\mathcal{C}_0 = \{S \in 2^{[3]} : |S| \leq 2\}$ and $\mathcal{C}_1 = \{S \in 2^{[3]} : |S| \geq 1\}$ along the complete graph $\mathcal{C}_0 \cap \mathcal{C}_1$, whose vertex set is $\{S \in 2^{[3]} : 1 \leq |S| \leq 2\}$.

For simplicity, we let

$$\mathcal{T}_k \coloneqq \{S \in 2^{[n]} : |S| = k\}, \quad k = 0, 1, \ldots, n. \tag{8.6}$$

Note that $2^{[n]} = \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_n$. We define $n - d + 1$ cliques whose vertex sets are given by

$$\mathcal{C}_k \coloneqq \mathcal{T}_k \cup \mathcal{T}_{k+1} \cup \cdots \cup \mathcal{T}_{k+d}, \quad k = 0, 1, \ldots, n - d. \tag{8.7}$$

Figure 8.2 shows the construction of cliques. We sometimes use $\mathcal{C}_k$ to express the clique itself whose vertex set is given by (8.7).

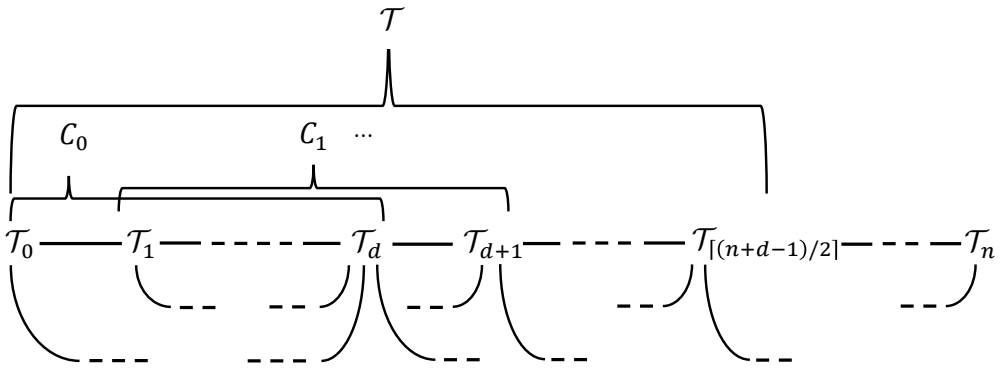


Figure 8.2: A sketch of the Cayley graph $\mathrm{Cay}(\mathcal{S}^d)$, the chordal cover Γ, and the Fourier support $\mathcal{T}$. In $\mathrm{Cay}(\mathcal{S}^d)$, there exists a pair of connected nodes $S \in \mathcal{T}_i$ and $T \in \mathcal{T}_j$ if and only if $|i-j| \leq d$. The chordal cover Γ is obtained by pasting the cliques $\mathcal{C}_k$ and $\mathcal{C}_{k+1}$ along $\mathcal{C}_k \cap \mathcal{C}_{k+1}$ for $k = 0, 1, \ldots, n-d-1$ and the Fourier support of Γ is $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_{\lceil (n+d-1)/2 \rceil}$, which is shown in Section 8.3.2.

We now describe how to construct a chordal cover Γ of $\mathrm{Cay}(\mathcal{S}^d)$. Let Γ be a graph obtained by pasting $\mathcal{C}_k$ and $\mathcal{C}_{k+1}$ along the complete graph $\mathcal{C}_k \cap \mathcal{C}_{k+1}$ for $k = 0, 1, \ldots, n-d-1$. Observe that the vertex set of Γ is $2^{[n]}$ and that $S, T \in 2^{[n]}$ are connected in Γ if and only if

$$||S| - |T|| \leq d$$

holds. Then, Γ is chordal by Proposition 5 since Γ is obtained by pasting the complete graphs $\mathcal{C}_k$ $(k = 0, 1, \ldots, n-d)$ along the complete graph $\mathcal{C}_k \cap \mathcal{C}_{k+1}$. Moreover, Γ covers $\mathrm{Cay}(\mathcal{S}^d)$ since all connected vertices $S, T \in 2^{[n]}$ in $\mathrm{Cay}(\mathcal{S}^d)$ satisfy $|S \Delta T| \leq d$, which implies $||S| - |T|| \leq d$.

8.3.2 A Fourier Support of the Chordal Cover

Let $\mathcal{T} \subseteq 2^{[n]}$ be a vertex set defined as

$$\mathcal{T} \coloneqq \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_{\lceil \frac{n+d-1}{2} \rceil}. \tag{8.8}$$

We show that $\mathcal{T}$ is a Fourier support of Γ defined in Section 8.3.1.

We first prove that any clique in Γ is contained in one of the cliques $\mathcal{C}_k$ $(k = 0, 1, \ldots, n-d)$, which implies all maximal cliques in Γ are given by $\mathcal{C}_k$ $(k = 0, 1, \ldots, n-d)$. Take an arbitrary clique $\mathcal{C}$ in Γ, and let S be a vertex with the smallest cardinality in $\mathcal{C}$. Then the cardinality of any vertex in $\mathcal{C}$ is at most $|S| + d$, and so the following holds:

- If $|S| \leq n - d$, then $\mathcal{C} \subseteq \mathcal{C}_{|S|}$.
- If $|S| > n - d$, then $\mathcal{C} \subseteq \mathcal{C}_{n-d}$.

Thus, any clique $\mathcal{C}$ is contained in one of $\mathcal{C}_k$ $(k = 0, 1, \ldots, n-d)$.

We next prove that $\mathcal{T}$ defined in (8.8) is a Fourier support of Γ, i.e., for all maximal cliques $\mathcal{C}_k$ $(k = 0, 1, \ldots, n-d)$, there exists $S_{\mathcal{C}_k} \in 2^{[n]}$ such that $S_{\mathcal{C}_k} \Delta \mathcal{C}_k \subseteq \mathcal{T}$. Choose $S_{\mathcal{C}_k}$ for $\mathcal{C}_k$ $(k = 0, 1, \ldots, n-d)$ as follows:

- If $k \leq \left\lceil \frac{n+d-1}{2} \right\rceil - d$, then $\mathcal{C}_k \subseteq \mathcal{T}$. Thus $S_{\mathcal{C}_k} = \emptyset$.
- If $k \geq \left\lceil \frac{n+d-1}{2} \right\rceil - d + 1$, then

$$n - k \leq n + d - 1 - \left\lceil \frac{n+d-1}{2} \right\rceil \leq \left\lceil \frac{n+d-1}{2} \right\rceil.$$

  Hence,

$$[n] \Delta \mathcal{C}_k = \mathcal{T}_{n-(k+d)} \cup \mathcal{T}_{n-(k+d)+1} \cup \cdots \cup \mathcal{T}_{n-k} \subseteq \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+d-1}{2} \right\rceil} = \mathcal{T}$$

  holds. Thus, we let $S_{\mathcal{C}_k} = [n]$.

As a consequence, we obtain the following result by using Theorem 10.

Theorem 11. *Let $\mathcal{T}$ be the subset of $2^{[n]}$ defined by* (8.8)*. Then the moment polytope $\mathcal{M}(\mathcal{S}^d)$ can be expressed as follows:*

$$\mathcal{M}(\mathcal{S}^d) = \left\{ \ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^d}} \ : \ \exists y \in \mathbb{R}^{\mathcal{D}_{\mathcal{T}\Delta\mathcal{T}}} \ s.t. \ \begin{array}{l} y_\delta = \ell_\delta \text{ for all } \delta \in \mathcal{D}_{\mathcal{S}^d}, \\ y_{\delta_\emptyset} = 1, \text{ and } M_{\mathcal{T}}(y) \succeq 0 \end{array} \right\}.$$

By Theorem 11, we have the exact SDP relaxation for binary POP (8.1) with the truncated moment matrix indexed by $\delta_S \in \mathcal{D}$ such that $|S| \leq \left\lceil \frac{n+d-1}{2} \right\rceil$.

8.3.3 Illustrating the Truncation with Example (8.3)

By linearizing the moment polytope of problem (8.3) in Section 8.1.2, we obtain the following problem with variables $\ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^2}}$:

$$\begin{aligned}
&\underset{\ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^2}}}{\text{minimize}} \quad \ell_{000} + \ell_{100} + \ell_{010} + \ell_{001} + \ell_{110} + \ell_{101} + \ell_{011} \\
&\text{subject to} \begin{pmatrix} \ell_{000} \\ \ell_{100} \\ \ell_{010} \\ \ell_{001} \\ \ell_{110} \\ \ell_{101} \\ \ell_{011} \end{pmatrix} \in \left\{ \ell \ : \ \exists y_{111} \quad \text{s.t.} \quad \ell_{000} = 1 \quad \text{and} \right. \\
&\qquad \left. \begin{bmatrix} 1 & \ell_{100} & \ell_{010} & \ell_{001} & \ell_{110} & \ell_{101} & \ell_{011} & y_{111} \\ \ell_{100} & 1 & \ell_{110} & \ell_{101} & \ell_{010} & \ell_{001} & y_{111} & \ell_{011} \\ \ell_{010} & \ell_{110} & 1 & \ell_{011} & \ell_{100} & y_{111} & \ell_{001} & \ell_{101} \\ \ell_{001} & \ell_{101} & \ell_{011} & 1 & y_{111} & \ell_{100} & \ell_{010} & \ell_{110} \\ \ell_{110} & \ell_{010} & \ell_{100} & y_{111} & 1 & \ell_{011} & \ell_{101} & \ell_{001} \\ \ell_{101} & \ell_{001} & y_{111} & \ell_{100} & \ell_{011} & 1 & \ell_{110} & \ell_{010} \\ \ell_{011} & y_{111} & \ell_{001} & \ell_{010} & \ell_{101} & \ell_{110} & 1 & \ell_{100} \\ y_{111} & \ell_{011} & \ell_{101} & \ell_{110} & \ell_{001} & \ell_{010} & \ell_{100} & 1 \end{bmatrix} \succeq 0 \right\}.
\end{aligned}$$

The Cayley graph, the chordal cover and the Fourier support are as shown in Example 3. Since $\left\lceil \frac{n+d-1}{2} \right\rceil = 2$ for $n = 3$ and $d = 2$, Theorem 11 indicates that the moment matrix can be truncated by

$$\mathcal{T} = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}\},$$

and so the above problem can be equivalently rewritten as

$$\begin{aligned}
&\underset{\ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^2}}}{\text{minimize}} \quad \ell_{000} + \ell_{100} + \ell_{010} + \ell_{001} + \ell_{110} + \ell_{101} + \ell_{011} \\
&\text{subject to} \begin{pmatrix} \ell_{000} \\ \ell_{100} \\ \ell_{010} \\ \ell_{001} \\ \ell_{110} \\ \ell_{101} \\ \ell_{011} \end{pmatrix} \in \left\{ \ell \ : \ \exists y_{111} \quad \text{s.t.} \quad \ell_{000} = 1 \quad \text{and} \right. \\
&\qquad \left. \begin{bmatrix} 1 & \ell_{100} & \ell_{010} & \ell_{001} & \ell_{110} & \ell_{101} & \ell_{011} \\ \ell_{100} & 1 & \ell_{110} & \ell_{101} & \ell_{010} & \ell_{001} & y_{111} \\ \ell_{010} & \ell_{110} & 1 & \ell_{011} & \ell_{100} & y_{111} & \ell_{001} \\ \ell_{001} & \ell_{101} & \ell_{011} & 1 & y_{111} & \ell_{100} & \ell_{010} \\ \ell_{110} & \ell_{010} & \ell_{100} & y_{111} & 1 & \ell_{011} & \ell_{101} \\ \ell_{101} & \ell_{001} & y_{111} & \ell_{100} & \ell_{011} & 1 & \ell_{110} \\ \ell_{011} & y_{111} & \ell_{001} & \ell_{010} & \ell_{101} & \ell_{110} & 1 \end{bmatrix} \succeq 0 \right\}.
\end{aligned}$$

Note that the size of the positive semidefinite matrix in this problem is reduced to 7 from 8 of the original one.

8.4 Further Truncating for Even-degree Binary POPs

This section focuses on even-degree binary POPs; i.e., only even-degree monomials appear in f of problem (8.1). Even-degree binary POPs, which includes the binary QOPs with f of degree 2, have been a popular subject of study. For instance, the SDP relaxation with the relaxation order $\omega = \lceil n/2 \rceil$ was proved to be an upper bound by Fawzi *et al.* [2015]. The biquadratic optimization and quartic POPs have been studied in [Ling *et al.*, 2009; Luo and Zhang, 2010; Yang and Yang, 2012], which are special cases of even-degree POPs. In particular, Yang and Yang [2012] proposed two approximation approaches for solving binary biquartic optimization problems.

For even-degree binary POPs, we can prove that the truncated moment matrix $M_{\mathcal{T}}(y)$ shown in Theorem 11 can be further reduced, i.e., the SDP relaxation with a smaller relaxation order can exactly solve an even-degree binary POP. When $d = 2$, the relaxation order of our result becomes $\omega = \lceil n/2 \rceil$, which coincides with the known upper bound for binary QOPs.

Let the degree of f be $2\widehat{d}$ throughout this section. For f with only even-degree monomials, we define $\mathcal{S}$ as $\mathcal{S}^{2\widehat{d}}_{\text{even}}$:

$$\mathcal{S}^{2\widehat{d}}_{\text{even}} := \{S \in 2^{[n]} : \ |S| = 0, 2, \ldots, 2\widehat{d}\}.$$

Then, we can obtain the following theorem, which will be proved in the subsequent sections.

Theorem 12. *Let $\mathcal{T}$ be the subset of $2^{[n]}$ defined as*

$$\mathcal{T} := \begin{cases} \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil} & \text{if } \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil \text{ is even,} \\ \mathcal{T}_1 \cup \mathcal{T}_3 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil} & \text{if } \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil \text{ is odd.} \end{cases}$$

Then, the moment polytope $\mathcal{M}(\mathcal{S}^{2\widehat{d}}_{even})$ can be expressed as

$$\mathcal{M}(\mathcal{S}^{2\widehat{d}}_{even}) = \left\{ \ell \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^{2\widehat{d}}_{even}}} \ : \ \exists y \in \mathbb{R}^{\mathcal{D}_{\mathcal{T}\Delta\mathcal{T}}} \ s.t. \ \begin{array}{l} y_\delta = \ell_\delta \text{ for all } \delta \in \mathcal{D}_{\mathcal{S}^{2\widehat{d}}_{even}}, \\ y_{\delta_\emptyset} = 1, \text{ and } M_{\mathcal{T}}(y) \succeq 0 \end{array} \right\}.$$

Thus, if the objective function f consists of only even monomials, problem (8.1) can be equivalently transformed into a smaller-size SDP whose moment matrix is truncated with

$$\mathcal{T} := \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil}$$

or

$$\mathcal{T} \coloneqq \mathcal{T}_1 \cup \mathcal{T}_3 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil}.$$

Note that for the binary QOPs with f having only even-degree monomials (i.e., $\widehat{d} = 1$), the above $\mathcal{T}$ becomes $\mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\lceil \frac{n}{2} \rceil}$ or $\mathcal{T}_1 \cup \mathcal{T}_3 \cup \cdots \cup \mathcal{T}_{\lceil \frac{n}{2} \rceil}$. This statement coincides with that of Laurent conjecture [Laurent, 2003], which was recently proved by Fawzi *et al.* [2015]; that is, the SDP relaxation with the relaxation order $\omega = \lceil n/2 \rceil$ in the hierarchy gives the exact relaxation for binary QOPs.

We briefly describe how to extract an optimal solution x^* of the even-degree binary POP from the optimal solution of SDP relaxation $\ell^* \in \mathbb{R}^{\mathcal{D}_{\mathcal{S}^{2\widehat{d}}_{\mathrm{even}}}}$; we multiply $\ell^*_{\{1\}}$ to each element of $\widehat{\ell} \coloneqq (\ell^*_{\delta_\emptyset}, \ell^*_{\delta_{\{1,2\}}}, \ell^*_{\delta_{\{1,3\}}}, \ldots, \ell^*_{\delta_{\{1,n\}}})$ so that $(\ell^*_{\delta_{\{1\}}}, \ell^*_{\delta_{\{2\}}}, \ell^*_{\delta_{\{3\}}}, \ldots, \ell^*_{\delta_{\{n\}}})$ can be obtained. As a result, the optimal solution is expressed as $x^*_1\widehat{\ell}$, which can be written as $\widehat{\ell}$ or $-\widehat{\ell}$. Since f is an even function, both of them are the optimal solutions to the original even-degree binary POP.

8.4.1 Cayley Graph for Even-degree Binary POPs

Observe that any two vertices $S, T \in 2^{[n]}$ are connected in $\mathrm{Cay}(\mathcal{S}^{2\widehat{d}}_{\mathrm{even}})$ if and only if $|S \Delta T|$ is even and $|S \Delta T| \leq 2\widehat{d}$ holds. Thus, $\mathrm{Cay}(\mathcal{S}^{2\widehat{d}}_{\mathrm{even}})$ has two connected components

$$\mathcal{T}_{\mathrm{even}} \coloneqq \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{2\lfloor n/2 \rfloor} \quad \text{and} \quad \mathcal{T}_{\mathrm{odd}} \coloneqq \mathcal{T}_1 \cup \mathcal{T}_3 \cup \cdots \cup \mathcal{T}_{2\lceil n/2 \rceil - 1},$$

where $\mathcal{T}_k$ $(k = 0, 1, \ldots, n)$ are defined by (8.6). More precisely, the Cayley graph $\mathrm{Cay}(\mathcal{S}^{2\widehat{d}}_{\mathrm{even}})$ has an edge between S and T if and only if

(a) $S, T \in \mathcal{T}_{\mathrm{even}}$ and $|S \Delta T| \leq 2\widehat{d}$, or

(b) $S, T \in \mathcal{T}_{\mathrm{odd}}$ and $|S \Delta T| \leq 2\widehat{d}$.

We define a map $\phi : 2^{[n]} \to 2^{[n]}$ by $\phi(S) \coloneqq \{1\} \Delta S$ for the succeeding discussion. Note that ϕ is an automorphism of $\mathrm{Cay}(\mathcal{S}^{2\widehat{d}}_{\mathrm{even}})$ that exchanges $\mathcal{T}_{\mathrm{even}}$ and $\mathcal{T}_{\mathrm{odd}}$. In addition, $\phi(S) \Delta \phi(T) = S \Delta T$ holds for all $S, T \in 2^{[n]}$.

Example 4. We show an example of the Cayley graph $\mathrm{Cay}(\mathcal{S}^{2\widehat{d}}_{\mathrm{even}})$ and the map ϕ for an even-degree binary POP with $n = 4$ and $\widehat{d} = 1$, which is

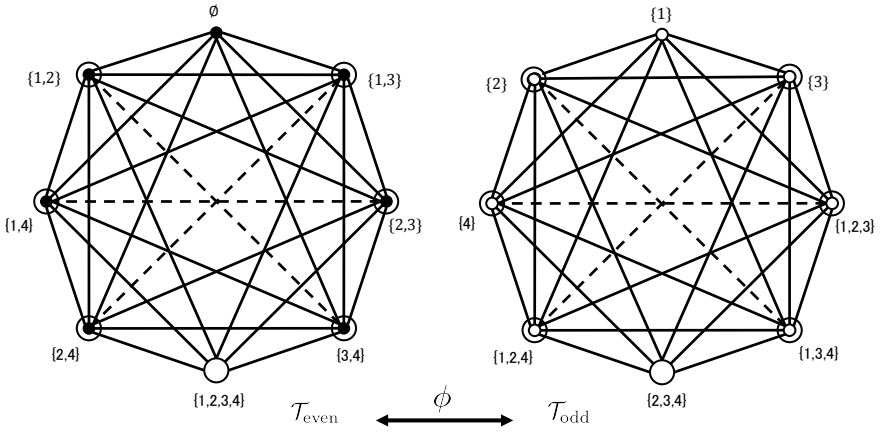


Figure 8.3: An example of the Cayley graph for the case of $n = 4$ and $\mathcal{S}^2_{\text{even}} = \{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\}$. Observe that $\phi(S) = \{1\}\Delta S$ is the automorphism of the Cayley graph that exchanges $\mathcal{T}_{\text{even}}$ and $\mathcal{T}_{\text{odd}}$. The graph generated by adding the dashed lines to $\text{Cay}(\mathcal{S}^2_{\text{even}})$ is a chordal cover Γ of $\text{Cay}(\mathcal{S}^2_{\text{even}})$ and the Fourier support of Γ is $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_2$ (indicated by filled circles).

a modified problem of (8.3) where all linear terms are multiplied by an additional variable x_4. The resulting problem is

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & 1 + x_1x_4 + x_2x_4 + x_3x_4 + x_1x_2 + x_1x_3 + x_2x_3 \\ \text{subject to} \quad & x \in \{-1,1\}^4. \end{aligned}$$

Figure 8.3 shows the Cayley graph $\text{Cay}(\mathcal{S}^2_{\text{even}})$ for the case of $n = 4$ and

$$\mathcal{S}^2_{\text{even}} = \{\emptyset, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}\}.$$

$\text{Cay}(\mathcal{S}^2_{\text{even}})$ consists of two connected components: $\mathcal{T}_{\text{even}}$ and $\mathcal{T}_{\text{odd}}$. The vertex set of $\text{Cay}(\mathcal{S}^2_{\text{even}})$ is $2^{[4]}$ and the edges are shown by the solid lines. Note that $\phi(S) = \{1\}\Delta S$ is the automorphism of the Cayley graph that exchanges $\mathcal{T}_{\text{even}}$ and $\mathcal{T}_{\text{odd}}$.

8.4.2 Constructing a Chordal Cover and its Fourier Support

To find a Fourier support $\mathcal{T}$ of a chordal graph Γ that covers $\mathrm{Cay}(\mathcal{S}_{\mathrm{even}}^{2\widehat{d}})$, we check whether $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil$ value of a given even-degree POP is even or odd. We first discuss the case where $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil$ is even in detail. We then briefly mention the required modification for the case where $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil$ is odd in Section 8.4.3.

Assume that $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil$ is even. For the construction of a chordal cover for $\mathrm{Cay}(\mathcal{S}_{\mathrm{even}}^{2\widehat{d}})$, we define $\left\lfloor \frac{n}{2} \right\rfloor - \widehat{d} + 1$ cliques whose vertex sets are given by

$$\mathcal{C}_k \coloneqq \mathcal{T}_k \cup \mathcal{T}_{k+2} \cup \cdots \cup \mathcal{T}_{k+2\widehat{d}}, \quad k = 0, 2, \ldots, 2\left\lfloor \frac{n}{2} \right\rfloor - 2\widehat{d}. \tag{8.9}$$

As above, we abuse the notation and let $\mathcal{C}_k$ to express the clique itself whose vertex set is given by (8.9). Note that the cliques $\mathcal{C}_k$ are indexed by even integer k. With the cliques, we construct a chordal cover Γ of $\mathrm{Cay}(\mathcal{S}_{\mathrm{even}}^{2\widehat{d}})$ as follows:

(i) Paste $\mathcal{C}_k$ and $\mathcal{C}_{k+2}$ along the complete graph $\mathcal{C}_k \cap \mathcal{C}_{k+2}$ for $k = 0, 2, \ldots, 2\left\lfloor \frac{n}{2} \right\rfloor - 2\widehat{d} - 2$. We denote the resulting connected component by Γ_{even}.

(ii) Paste $\phi(\mathcal{C}_k)$ and $\phi(\mathcal{C}_{k+2})$ along the complete graph $\phi(\mathcal{C}_k) \cap \phi(\mathcal{C}_{k+2})$ for $k = 0, 2, \ldots, 2\left\lfloor \frac{n}{2} \right\rfloor - 2\widehat{d} - 2$. The obtained connected component is denoted by Γ_{odd}.

Define $\Gamma \coloneqq \Gamma_{\mathrm{even}} \cup \Gamma_{\mathrm{odd}}$, which is chordal thanks to Proposition 5. Observe that Γ_{even} and Γ_{odd} satisfy the following conditions:

(A) The vertex set of Γ_{even} is $\mathcal{T}_{\mathrm{even}}$, and $S, T \in \mathcal{T}_{\mathrm{even}}$ are connected in Γ_{even} if and only if $||S| - |T|| \leq 2\widehat{d}$ holds.

(B) The vertex set of Γ_{odd} is $\mathcal{T}_{\mathrm{odd}}$, and $S, T \in \mathcal{T}_{\mathrm{odd}}$ are connected in Γ_{odd} if and only if $||\phi(S)| - |\phi(T)|| \leq 2\widehat{d}$ holds.

Considering conditions (a) and (b) in Section 8.4.1, we see that $\Gamma = \Gamma_{\mathrm{even}} \cup \Gamma_{\mathrm{odd}}$ covers $\mathrm{Cay}(\mathcal{S}_{\mathrm{even}}^{2\widehat{d}})$.

Let $\mathcal{T} \subseteq 2^{[n]}$ be a vertex set defined as

$$\mathcal{T} \coloneqq \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil}. \tag{8.10}$$

We see that $\mathcal{T}$ is a Fourier support of Γ.

First, we need to check that any clique in Γ is contained in one of the cliques $\mathcal{C}_k$ or $\phi(\mathcal{C}_k)$ for $k = 0, 2, \ldots, 2\left\lfloor \frac{n}{2} \right\rfloor - 2\widehat{d}$, which implies that all maximal cliques in Γ are given by $\mathcal{C}_k$ and $\phi(\mathcal{C}_k)$ for $k = 0, 2, \ldots, 2\left\lfloor \frac{n}{2} \right\rfloor - 2\widehat{d}$. This can be proved as in Section 8.3.2.

Next, we prove that $\mathcal{T}$ defined in (8.10) is a Fourier support of Γ. To show this, we examine whether there exists $S_{\mathcal{C}_k} \in 2^{[n]}$ such that $S_{\mathcal{C}_k} \Delta \mathcal{C}_k \subseteq \mathcal{T}$ for each $\mathcal{C}_k$ ($k = 0, 2, \ldots, 2\left\lfloor \frac{n}{2} \right\rfloor - 2\widehat{d}$). This is sufficient for guaranteeing that $\mathcal{T}$ is a Fourier support since $\phi(S_{\mathcal{C}_k}) \Delta \phi(\mathcal{C}_k) = S_{\mathcal{C}_k} \Delta \mathcal{C}_k \subseteq \mathcal{T}$ follows as regards the cliques $\phi(\mathcal{C}_k)$. An appropriate choice of $S_{\mathcal{C}_k}$ is given as follows:

- If $k \leq \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil - 2\widehat{d}$, then $\mathcal{C}_k \subseteq \mathcal{T}$. Consequently, $S_{\mathcal{C}_k} = \emptyset$.
- If $k \geq \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil - 2\widehat{d} + 2$ and n is even, then

  $$n - k \leq n + 2\widehat{d} - 2 - \left\lceil \frac{n + 2\widehat{d} - 2}{2} \right\rceil \leq \left\lceil \frac{n + 2\widehat{d} - 2}{2} \right\rceil.$$

  Hence,

  $$[n] \Delta \mathcal{C}_k = \mathcal{T}_{n-(k+2\widehat{d})} \cup \mathcal{T}_{n-(k+2\widehat{d})+2} \cup \cdots \cup \mathcal{T}_{n-k} \subseteq \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil} = \mathcal{T}.$$

  Therefore, we let $S_{\mathcal{C}_k} = [n]$.
- If $k \geq \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil - 2\widehat{d} + 2$ and n is odd, then we have $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil = \left\lceil \frac{n}{2} \right\rceil + \widehat{d} - 1 = n - \left\lceil \frac{n}{2} \right\rceil + \widehat{d}$, and thus

  $$n - k + 1 \leq n + 2\widehat{d} - 1 - \left\lceil \frac{n + 2\widehat{d} - 2}{2} \right\rceil = n - \left\lceil \frac{n}{2} \right\rceil + \widehat{d} = \left\lceil \frac{n + 2\widehat{d} - 2}{2} \right\rceil.$$

  Hence,

  $$\phi([n]) \Delta \mathcal{C}_k = [n] \Delta \phi(\mathcal{C}_k)$$

$$\begin{aligned}
&\subseteq [n]\Delta(\mathcal{T}_{k-1} \cup \mathcal{T}_{k+1} \cup \cdots \cup \mathcal{T}_{k+2\widehat{d}+1}) \\
&\subseteq \mathcal{T}_{n-k-2\widehat{d}-1} \cup \mathcal{T}_{n-k-2\widehat{d}+1} \cup \cdots \cup \mathcal{T}_{n-k+1} \\
&\subseteq \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_{\lceil (n+2\widehat{d}-2)/2 \rceil} \\
&= \mathcal{T}.
\end{aligned}$$

Therefore, we let $S_{\mathcal{C}_k} = \phi([n]) = \{2, 3, \ldots, n\}$.

As a consequence, $\mathcal{T}$ is a Fourier support of Γ that covers the Cayley graph $\mathrm{Cay}(\mathcal{S}^{2\widehat{d}}_{\mathrm{even}})$.

Example 5. As in Figure 8.3, if $n = 4$ and $\widehat{d} = 1$, then the graph obtained by adding the dashed lines to $\mathrm{Cay}(\mathcal{S}^2_{\mathrm{even}})$ is a chordal cover Γ of $\mathrm{Cay}(\mathcal{S}^2_{\mathrm{even}})$. The Fourier support $\mathcal{T}$ is $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_2$ (indicated by the filled circles). Note that there are four maximal cliques for Γ; $\mathcal{C}_0 \coloneqq \mathcal{T}_0 \cup \mathcal{T}_2$ (induced by the small filled circles) and $\mathcal{C}_2 \coloneqq \mathcal{T}_2 \cup \mathcal{T}_4$ (by the large open circles) in the left component, and $\phi(\mathcal{C}_0) \coloneqq \{1\}\Delta\mathcal{C}_0$ (by the small open circles) and $\phi(\mathcal{C}_2) \coloneqq \{1\}\Delta\mathcal{C}_2$ (by the large open circles) in the right component. We can confirm that $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_2$ is a Fourier support as follows:

- $\emptyset\Delta\mathcal{C}_0 \subseteq \mathcal{T}$,
- $[4]\Delta\mathcal{C}_2 \subseteq \mathcal{T}$,
- $\phi(\emptyset)\Delta\phi(\mathcal{C}_0) = \emptyset\Delta\mathcal{C}_0 \subseteq \mathcal{T}$, and
- $\phi([4])\Delta\phi(\mathcal{C}_2) = [4]\Delta\mathcal{C}_2 \subseteq \mathcal{T}$.

8.4.3 Modifications Required in the Odd Case

For the case where $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil$ is odd, we define the cliques

$$\mathcal{C}_k \coloneqq \mathcal{T}_k \cup \mathcal{T}_{k+2} \cup \cdots \cup \mathcal{T}_{k+2\widehat{d}}, \quad k = 1, 3, \ldots, 2\left\lceil \frac{n}{2} \right\rceil - 2\widehat{d} - 1.$$

Note that $\mathcal{C}_k$ are indexed by odd integers. We construct Γ_{even} and Γ_{odd} as follows:

- Construct Γ_{odd} by pasting $\mathcal{C}_k$ and $\mathcal{C}_{k+2}$ along $\mathcal{C}_k \cap \mathcal{C}_{k+2}$ for $k = 1, 3, \ldots, 2\left\lceil \frac{n}{2} \right\rceil - 2\widehat{d} - 3$.

- Construct Γ_{even} by pasting $\phi(\mathcal{C}_k)$ and $\phi(\mathcal{C}_{k+2})$ along $\phi(\mathcal{C}_k) \cap \phi(\mathcal{C}_{k+2})$ for $k = 1, 3, \ldots, 2\left\lceil \frac{n}{2} \right\rceil - 2\widehat{d} - 3$.

Thus we obtain the chordal cover $\Gamma \coloneqq \Gamma_{\text{odd}} \cup \Gamma_{\text{even}}$ for $\text{Cay}(\mathcal{S}^{2\widehat{d}}_{\text{even}})$. The maximal cliques of Γ are $\mathcal{C}_k$ for $k = 1, 3, \ldots, 2\left\lceil \frac{n}{2} \right\rceil - 2\widehat{d} - 1$ together with the corresponding $\phi(\mathcal{C}_k)$.

The Fourier support of Γ is given by

$$\mathcal{T} \coloneqq \mathcal{T}_1 \cup \mathcal{T}_3 \cup \cdots \cup \mathcal{T}_{\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil},$$

which can be confirmed by choosing $S_{\mathcal{C}_k}$ satisfying $S_{\mathcal{C}_k} \Delta \mathcal{C}_k \subseteq \mathcal{T}$ for each maximal clique $\mathcal{C}_k$ as follows:

- If $k \leq \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil - 2\widehat{d}$, then $S_{\mathcal{C}_k} = \emptyset$.
- If $k \geq \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil - 2\widehat{d} + 2$ and n is even, then $S_{\mathcal{C}_k} = [n]$.
- If $k \geq \left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil - 2\widehat{d} + 2$ and n is odd, then $S_{\mathcal{C}_k} = \phi([n])$.

Thus, the desired result for the case with odd $\left\lceil \frac{n+2\widehat{d}-2}{2} \right\rceil$ follows. By the discussion up to this point and Theorem 10, we have proved Theorem 12.

8.4.4 Discussion on Formulating General Binary POPs as Even-degree Binary POPs

Since general binary POPs can be expressed as even-degree binary POPs by adding a single variable, we may expect that Theorem 12 can yield better results for general POPs than Theorem 11. However, we show that the size of the SDPs is not further reduced with this approach.

Let $\widehat{c}_\alpha$ denote the coefficients of the monomials of even degree and $\widetilde{c}_\alpha$ the coefficients of the monomials of odd degree. Then, the objective function f of general binary POP (8.1) can be written as

$$f(x) = \sum_{||\alpha||_1 \leq d} c_\alpha x^\alpha = \sum_{||\alpha||_1 \leq d} \widehat{c}_\alpha x^\alpha + \sum_{||\alpha||_1 \leq d} \widetilde{c}_\alpha x^\alpha.$$

By introducing the additional variable $x_{n+1} \in \{-1, 1\}$, binary POP (8.1) can be expressed as the following *even-degree* binary POP on $\{-1, 1\}^{n+1}$, to which Theorem 12 can be applied:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \widetilde{f}(x) \coloneqq \sum_{||\alpha||_1 \le d} \widehat{c}_\alpha x^\alpha + \sum_{||\alpha||_1 \le d} \widetilde{c}_\alpha x^\alpha x_{n+1} \\ \text{subject to} \quad & x \in \{-1, 1\}^{n+1}. \end{aligned} \tag{8.11}$$

Let the degree of $\widetilde{f}$ be $\widetilde{d}$. Obviously, $\widetilde{d} = d$ if d is even, and $\widetilde{d} = d + 1$ if d is odd. Hence $d \le \widetilde{d}$.

If problem (8.11) has an optimal solution $x^* \in \{-1, 1\}^{n+1}$ with $x^*_{n+1} = 1$, then an optimal solution to problem (8.1) can be obtained as $(x^*_1, x^*_2, \ldots, x^*_n)$. We note that problem (8.11) always has an optimal solution with $x^*_{n+1} = 1$. More precisely, if problem (8.11) has an optimal solution x^* with $x^*_{n+1} = -1$, then $-x^*$ is also an optimal solution since $\widetilde{f}$ is an even function. Therefore, any general binary POP (8.1) with the constraint $x \in \{-1, 1\}^n$ can be formulated as an even-degree binary POP with the constraint $x \in \{-1, 1\}^{n+1}$.

However, applying Theorem 12 to problem (8.11) results in a larger truncated moment matrix than the one obtained by applying Theorem 11 to problem (8.1). To see this, we define $\mathcal{T}^n_k \coloneqq \{S \in 2^{[n]} : |S| = k\}$; note that the size of $[n]$ is included in the definition of $\mathcal{T}^n_k$. If Theorem 11 is applied to problem (8.1), the moment matrix is truncated by the following Fourier support:

$$\mathcal{T}^n \coloneqq \mathcal{T}^n_0 \cup \mathcal{T}^n_1 \cup \cdots \cup \mathcal{T}^n_{\lceil \frac{n+d-1}{2} \rceil}.$$

On the other hand, if Theorem 12 is applied to problem (8.11), the moment matrix is truncated by the Fourier support defined as follows:

$$\widetilde{\mathcal{T}}^{n+1} \coloneqq \begin{cases} \mathcal{T}^{n+1}_0 \cup \mathcal{T}^{n+1}_2 \cup \cdots \cup \mathcal{T}^{n+1}_{\lceil \frac{(n+1)+\widetilde{d}-2}{2} \rceil} & \text{if } \left\lceil \frac{(n+1)+\widetilde{d}-2}{2} \right\rceil \text{ is even,} \\ \mathcal{T}^{n+1}_1 \cup \mathcal{T}^{n+1}_3 \cup \cdots \cup \mathcal{T}^{n+1}_{\lceil \frac{(n+1)+\widetilde{d}-2}{2} \rceil} & \text{if } \left\lceil \frac{(n+1)+\widetilde{d}-2}{2} \right\rceil \text{ is odd.} \end{cases}$$

We now prove $|\mathcal{T}^n| \le |\widetilde{\mathcal{T}}^{n+1}|$. Let $\omega \coloneqq \lceil (n + d - 1)/2 \rceil$ and $\widetilde{\omega} \coloneqq \left\lceil (n + \widetilde{d} - 1)/2 \right\rceil$. Note that $\omega \le \widetilde{\omega}$ since $d \le \widetilde{d}$, and thus

$$|\mathcal{T}^n| = |\mathcal{T}^n_0| + |\mathcal{T}^n_1| + \cdots + |\mathcal{T}^n_\omega| = \sum_{i=0}^{\omega} \binom{n}{i} \le \sum_{i=0}^{\widetilde{\omega}} \binom{n}{i}. \tag{8.12}$$

We show that the right hand side of the inequality of (8.12) coincides with $|\widetilde{\mathcal{T}}^{n+1}|$ by considering two cases: $\widetilde{\omega}$ is even or odd. If $\widetilde{\omega}$ is even, then

$$\sum_{i=0}^{\widetilde{\omega}} \binom{n}{i} = \binom{n}{0} + \sum_{i=1}^{\widetilde{\omega}/2} \left\{ \binom{n}{2i-1} + \binom{n}{2i} \right\} = \binom{n+1}{0} + \sum_{i=1}^{\widetilde{\omega}/2} \binom{n+1}{2i} = |\widetilde{\mathcal{T}}^{n+1}|.$$

If $\widetilde{\omega}$ is odd, then

$$\sum_{i=0}^{\widetilde{\omega}} \binom{n}{i} = \sum_{i=0}^{\lfloor \widetilde{\omega}/2 \rfloor} \left\{ \binom{n}{2i} + \binom{n}{2i+1} \right\} = \sum_{i=0}^{\lfloor \widetilde{\omega}/2 \rfloor} \binom{n+1}{2i+1} = |\widetilde{\mathcal{T}}^{n+1}|.$$

Consequently, we obtain $|\mathcal{T}^n| \leq |\widetilde{\mathcal{T}}^{n+1}|$.

8.4.5 Difficulties in Further Extension

From the discussion on the even case, one may think that the techniques used in the proof can be extended to the case where the degree of each term in the objective function is a multiple of a positive integer m. More precisely, as an extension from the cases of $m = 1$ and $m = 2$ to that of an arbitrary m, we may consider the following set:

$$\mathcal{S}^d_{0=(d \bmod m)} \coloneqq \{S \in 2^{[n]} : \ |S| = 0, m, \ldots, km (= d)\}.$$

We now show the extension is not so straightforward. For instance, when $m = 3$, the Cayley graph can be connected, and thus there are no function $\phi(\cdot)$ which changes from

$$\mathcal{T}_{0=(d \bmod m)} \coloneqq \mathcal{T}_0 \cup \mathcal{T}_m \cup \cdots \cup \mathcal{T}_{km}$$

to

$$\mathcal{T}_{i=(d \bmod m)} \coloneqq \mathcal{T}_i \cup \mathcal{T}_{m+i} \cup \cdots \cup \mathcal{T}_{(k-1)m+i}, \quad i = 1, 2.$$

Therefore, to use the techniques as in Sections 8.4.1 and 8.4.2, we need to solve the complicated problem of finding a function $\phi(\cdot)$ that maps one maximal clique to another in the connected Cayley graph. As a result, it is difficult to extend the proof for $m = 1$ and $m = 2$ to an arbitrary m ($m \geq 3$).

8.5 Experiments

We conducted numerical experiments to see whether the bounds proved in Sections 8.3 and 8.4 could be empirically obtained. The numerical results for general and even-degree binary POPs are presented in Sections 8.5.1 and 8.5.2, respectively. As test instances, we used randomly generated binary POPs and binary POPs such that the coefficients of f are all ones.

8.5.1 General Binary POPs

In the case of general binary POPs (8.1), the relaxation order ω indicates that the moment matrix is truncated by $\mathcal{T} = \mathcal{T}_0 \cup \mathcal{T}_1 \cup \cdots \cup \mathcal{T}_\omega$.

Random Instances

Table 8.1: The relaxation order ω required to obtain the exact optimal solutions among all 1000 random general binary POPs instances with various numbers of variables n and degrees d. The number in the parenthesis shows the theoretical upper bound $\bar{\omega} = \left\lceil \frac{n+d-1}{2} \right\rceil$. The bold digits indicate the result such that the upper bound is tight.

	d								
n	1	2	3	4	5	6	7	8	9
3	1 (2)	**2 (2)**	2 (3)						
4	1 (2)	2 (3)	2 (3)	2 (4)					
5	1 (3)	2 (3)	2 (4)	2 (4)	3 (5)				
6	1 (3)	2 (4)	2 (4)	2 (5)	3 (5)	3 (6)			
7	1 (4)	2 (4)	2 (5)	2 (5)	3 (6)	3 (6)	4 (7)		
8	1 (4)	2 (5)	2 (5)	2 (6)	3 (6)	3 (7)	4 (7)	4 (8)	
9	1 (5)	2 (5)	2 (6)	3 (6)	3 (7)	3 (7)	4 (8)	4 (8)	5 (9)

We randomly generated test instances with the coefficients c_α ($\alpha \in \{0,1\}^n$) of binary POPs (8.2) generated as $c_\alpha = 10 \times u_\alpha$, where u_α was sampled from the standard normal distribution. We generated 1000 test instances for each (n, d) where $n = 3, 4, \ldots, 9$ and $d = 1, 2, \ldots, 9$. The relaxation order was increased from $\omega = \lceil d/2 \rceil$.

Table 8.1 shows the minimum relaxation order ω such that all 1000 test instances with the specified n and d were exactly solved. The numbers in the parentheses indicate the theoretical bound $\bar{\omega} := \lceil \frac{n+d-1}{2} \rceil$. We observe that, in the case of randomly generated binary POPs, the empirical order ω is smaller than the theoretical bound $\bar{\omega}$ for almost all problems of (n, d), except for $n = 3$ and $d = 2$. In fact, most instances in Table 8.1 were solved exactly via SDP relaxation with $\omega = \lceil d/2 \rceil$ except for the case $d = 2$ and only two instances out of 1000 trials with $(n, d) = (9, 4)$. Considering the results, it would be interesting to study whether the SDP relaxation with $\omega = \lceil d/2 \rceil$ can be exact for random instances of binary POPs under some assumptions.

For binary linear optimization problems with $d = 1$, it is known that the SDP relaxation with $\omega = 1$ can find the optimal solution, although the theoretical upper bound is larger than 1. We see from the first column of the table that a gap exists between the known bound $\omega = 1$ and our theoretical bound $\bar{\omega}$. To narrow this gap will also be interesting future work.

All-one-coefficient Instances

Table 8.2: The relaxation order ω required to obtain the exact optimal solutions for all-one-coefficient general binary POPs instances with various numbers of variables n and degrees d. The number in the parenthesis shows the theoretical upper bound $\bar{\omega} = \left\lceil \frac{n+d-1}{2} \right\rceil$. The bold digits indicate the results such that the upper bound is tight.

	d								
n	1	2	3	4	5	6	7	8	9
3	1 (2)	1 (2)	**3 (3)**						
4	1 (2)	**3 (3)**	**3 (3)**	**4 (4)**					
5	1 (3)	1 (3)	2 (4)	3 (4)	**5 (5)**				
6	1 (3)	**4 (4)**	2 (4)	3 (5)	4 (5)	**6 (6)**			
7	1 (4)	1 (4)	2 (5)	4 (5)	3 (6)	5 (6)	**7 (7)**		
8	1 (4)	**5 (5)**	2 (5)	5 (6)	3 (6)	5 (7)	6 (7)	**8 (8)**	
9	1 (5)	1 (5)	2 (6)	4 (6)	3 (7)	4 (7)	5 (8)	6 (8)	**9 (9)**

We consider special instances of binary POP (8.2) where $c_\alpha = 1$ for all $\alpha \in \{0, 1\}^n$ and $\|\alpha\|_1 \leq d$. Table 8.2 displays the minimum relaxation order

ω required to exactly solve the test instances. The numbers in parentheses indicate the theoretical bound $\bar{\omega} := \left\lceil \frac{n+d-1}{2} \right\rceil$.

Note that the exact optimal values of the problem of $d = n$ were obtained at the theoretical bound $\bar{\omega}$, which implies that our theoretical bound is tight. This supports the results of [Kurpisz *et al.*, 2017], which showed that $\omega = n$ is the lower bound where any binary POPs with $d = n$ have no integrality gap. The result in the table also implies that the theoretical bound is tight for binary QOPs with even n values.

8.5.2 Even-degree Binary POPs

In the case of even-degree binary POPs, the relaxation order ω means that the moment matrix can be truncated by

$$\mathcal{T} = \begin{cases} \mathcal{T}_0 \cup \mathcal{T}_2 \cup \cdots \cup \mathcal{T}_\omega & \text{if } \omega \text{ is even,} \\ \mathcal{T}_1 \cup \mathcal{T}_3 \cup \cdots \cup \mathcal{T}_\omega & \text{if } \omega \text{ is odd.} \end{cases}$$

Random Instances

Table 8.3: The relaxation order ω required to obtain the exact optimal solutions for all 1000 randomly generated instances of even-degree binary POPs with various numbers of variables n and degrees d. The number in the parenthesis shows the theoretical order $\bar{\omega} = \left\lceil \frac{n+d-2}{2} \right\rceil$. The bold digits indicate the results such that the upper bound is tight.

	d			
n	2	4	6	8
3	**2 (2)**			
4	**2 (2)**	2 (3)		
5	2 (3)	2 (4)		
6	2 (3)	2 (4)	3 (5)	
7	2 (4)	2 (5)	3 (6)	
8	2 (4)	3 (5)	3 (6)	4 (7)
9	2 (5)	2 (6)	3 (7)	4 (8)

As in Section 8.5.1, we randomly generated 1000 test instances for each (n, d) with $n = 3, 4, \ldots, 9$ and $d = 2, 4, 6, 8$; the coefficients c_α ($\alpha \in \{0, 1\}^n$) of

even-degree binary POP (8.2) were generated as $c_\alpha = 10 \times u_\alpha$, where u_α was sampled from the standard normal distribution. We solved the test instances using the SDP relaxations with the truncated moment matrices.

Table 8.3 shows the minimum order ω with which the exact optimal values of all test instances with specified d and n were obtained. The theoretical bound $\bar{\omega} := \lceil \frac{n+d-2}{2} \rceil$ is shown in the parentheses. We observe that almost all randomly generated test problems were solved with the relaxation order smaller than the theoretical bound, except for the instances with $n = 3, 4$ and $d = 2$. Furthermore, as in Section 8.5.1, the SDP relaxation with $\omega = \lceil d/2 \rceil$ was exact for all instances in Table 8.3 except for the case $d = 2$ and only one instance with $(n, d) = (8, 4)$.

All-one-coefficient Instances

Table 8.4: The relaxation order ω required to obtain the exact optimal solutions for all-one-coefficient instances of even-degree binary POPs with various numbers of variables n and degrees d. The number in the parenthesis shows the theoretical order $\bar{\omega} = \left\lceil \frac{n+d-2}{2} \right\rceil$. The bold digits indicate the results such that the upper bound is tight.

	d			
n	2	4	6	8
3	**2 (2)**			
4	1 (2)	**3 (3)**		
5	**3 (3)**	**4 (4)**		
6	1 (3)	3 (4)	**5 (5)**	
7	**4 (4)**	3 (5)	**6 (6)**	
8	1 (4)	4 (5)	5 (6)	**7 (7)**
9	**5 (5)**	5 (6)	5 (7)	**8 (8)**

We consider even-degree binary POP (8.2) where $c_\alpha = 1$ for all $\alpha \in \{0, 1\}^n$ and $||\alpha||_1$ is even. The numerical results for even-degree binary POP (8.2) of degree $d = 2, 4, 6, 8$ with all-one coefficients are shown in Table 8.4. We observe that the exact optimal value of the problem is obtained at the theoretical bound $\bar{\omega} := \lceil \frac{n+d-2}{2} \rceil$ in all instances with $d = n$, which confirms the tightness of our upper bound and the lower bound of Kurpisz *et al.* [2017]

as in Section 8.5.1. Furthermore, for binary QOPs without linear terms, it is proved by Laurent [2003] that $\omega = \lceil n/2 \rceil$ is the lower bound for the exact relaxation, which is confirmed by the results for odd n in the table.

8.6 Conclusion

We have shown that the exact optimal values of binary POPs are obtained by solving the $\lceil (n+d-1)/2 \rceil$th SDP relaxation in Lasserre's hierarchy of SDP relaxations. We have also improved the relaxation order to $\lceil (n+d-2)/2 \rceil$ for the case of even-degree binary POPs. The techniques used to derive these results are based on [Fawzi *et al.*, 2015], and the bound $\lceil (n+d-2)/2 \rceil$ for even-degree binary POPs generalizes the result proved by [Fawzi *et al.*, 2015] for binary QOPs. Experiments confirmed the tightness of our results.

Here, the moment matrix of a given binary POP has been truncated using a Fourier support of a chordal cover that covers the Cayley graph arising from the binary POP. In [Waki *et al.*, 2006], the sparse SDP relaxation was proposed by exploiting the sparsity of a given POP. Specifically, the size of the moment matrix for the sparse SDP relaxation was reduced by using the maximal cliques of an extended chordal graph obtained from the sparsity pattern graph of the given POP. It will be interesting to investigate the relationship between the two approaches. Furthermore, as in Section 8.5.1, almost all random instances of binary POPs with $d > 2$ have been solved exactly via SDP relaxation with $\omega = \lceil d/2 \rceil$. Thus it may be worthwhile to study conditions for SDP relaxation with $\lceil d/2 \rceil$ to be exact.

Solving POPs by the hierarchy of SDP relaxations is a very challenging problem since it requires to solve increasingly large SDP relaxations for the desired accuracy. Moreover, the SDPs induced from POPs are often degenerate, causing a great deal of numerical difficulties for the SDP solvers. It is well-known that the SDP solvers based on the interior-point methods [Sturm, 1999; Tütüncü *et al.*, 2003; Fujisawa *et al.*, 2008] cannot handle the SDPs with more than several thousand variables; as a result, POPs with moderate n and d values cannot be solved by the solvers. Therefore, from the computational viewpoints, it is essential to have the size of the SDPs as small as possible for accurate solutions of POPs. The bounds presented in this chapter for binary POPs provide the important information on the largest size of SDPs to be solved for computing exact optimal values of binary POPs in advance.

Chapter 9

Conclusion

In this dissertation, we have addressed three kinds of optimization problems: bandit combinatorial optimization, submodular function maximization, and binary polynomial optimization. Below we conclude this dissertation by summarizing each contribution and presenting future work.

Chapter 3: Bandit Combinatorial Optimization with ZDDs. We developed a practical ZDD-based algorithm and proved its any-time regret bounds. The per-round complexity of our algorithms is linear in the ZDD size, which is empirically much smaller than the size of action sets. Experiments confirmed the efficiency and practical utility of our algorithm.

If we are allowed to perform more costly operations, such as projection onto the convex hull of feasible solutions, in each round, then it may be possible to improve the regret bounds. Therefore, it will be interesting to study whether such operations can be performed efficiently with ZDDs.

Chapter 5: Submodular Function Maximization over Graphs with ZDDs. We developed a ZDD-based algorithm for monotone submodular function maximization with various constraints defined on graphs. Our algorithm achieves a $(1 - \kappa)$-approximation, and its computation complexity is linear in the ZDD size. Experiments confirmed that our algorithm is effective for a wide variety of real-world instances.

The obtained curvature-dependent approximation ratio can sometimes be pessimistic (close to 0) in practice, and so to develop efficient ZDD-based algorithms with better approximation guarantees is important future work.

Chapter 6: Best-first Search Algorithm for Submodular Knapsack Problem. We proposed an accelerated best-first search algorithm for submodular knapsack problems. The acceleration is achieved by using a new upper-bound computation method and termination condition. We experimentally confirmed that our algorithm can run much faster than existing best-first search algorithms.

As future work, it will be interesting to develop fast best-first search algorithms for other constraints, e.g., matroids and independence systems.

Chapter 7: k-submodular Function Maximization under Matroid Constraint. We proved that the greedy algorithm can achieve a 1/2-approximation guarantee for monotone k-submodular function maximization with a matroid constraint. This result provides the first constant-factor approximation guarantee for the problem.

As mentioned in the chapter, to improve the approximation guarantee by using a bounded k value will be a challenging but interesting future research direction. It will also be interesting to study k-submodular function maximization with other constraints, e.g., knapsack constraints.

Chapter 8: Exact SDP Reformulation of Binary Polynomial Optimization via Graph Representation. We proved that, given an n-dimensional binary polynomial optimization instance with a d-degree objective function, the $\lceil (n+d-1)/2 \rceil$th SDP is sufficient for exactly reformulating the problem. We also proved that the $\lceil (n+d-1)/2 \rceil$th SDP is sufficient if objective functions have only even degree monomials.

As mentioned in Section 8.6, given binary POP instances with some special structures, we may be able to obtain smaller upper bounds on the relaxation order. One promising approach is to study the structure of the Cayley graphs: Our proof technique is based on the the idea of pasting cliques, which can be seen as an analogy of a path-decomposition of the Cayley graph. Hence, it will be interesting to study whether other techniques known in the field of graph theory, e.g., tree-decomposition, can be utilized to obtain empirically smaller exact SDP reformulations. Note, however, that to improve the above relaxation orders is impossible in the worst case since our results are guaranteed to be tight in general.

Bibliography

N. Ailon, K. Hatano, and E. Takimoto. Bandit online optimization over the permutahedron. In *Proceedings of the 25th International Conference on Algorithmic Learning Theory*, pages 215–229, 2014.

S. B. Akers. Binary decision diagrams. *IEEE Trans. Comput.*, 100(6):509–516, 1978.

N. Alon, I. Gamzu, and M. Tennenholtz. Optimizing budget allocation among channels and influencers. In *Proceedings of the 21st International Conference on World Wide Web*, pages 381–388, 2012.

M. F. Anjos. Semidefinite optimization approaches for satisfiability and maximum-satisfiability problems. *J. Satisf. Boolean Model. Comput.*, 1:1–47, 2005.

J.-Y. Audibert, S. Bubeck, and G. Lugosi. Regret in online combinatorial optimization. *Math. Oper. Res.*, 39(1):31–45, 2014.

D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Appl. Math.*, 65(1):21–46, 1996.

B. Awerbuch and R. D. Kleinberg. Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 45–53, 2004.

F. Bach. Learning with submodular functions: A convex optimization perspective. *Foundations and Trends® in Machine Learning*, 6(2–3):145–373, 2013.

R. B. Bapat. *Graphs and Matrices*. Springer, 2nd edition, 2014.

P. L. Bartlett, V. Dani, T. Hayes, S. Kakade, A. Rakhlin, and A. Tewari. High-probability regret bounds for bandit online linear optimization. In *Proceedings of the 21st Annual Conference on Learning Theory*, pages 335–342, 2008.

D. Bergman, A. A. Cire, W.-J. van Hoeve, and J. Hooker. *Decision Diagrams for Optimization*. Springer, 1st edition, 2016.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 1st edition, 2006.

G. Braun and S. Pokutta. An efficient high-probability algorithm for linear bandits. *arXiv preprint arXiv:1610.02072*, 2016.

R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, 100(8):677–691, 1986.

S. Bubeck, N. Cesa-Bianchi, and S. M. Kakade. Towards minimax policies for online linear optimization with bandit feedback. In *Proceedings of the 25th Conference on Learning Theory*, pages 1–14, 2012.

G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. *SIAM J. Comput.*, 40(6):1740–1766, 2011.

N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge university press, 2006.

N. Cesa-Bianchi and G. Lugosi. Combinatorial bandits. *J. Comput. Syst. Sci.*, 78(5):1404–1422, 2012.

C. Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, pages 245–253, 2005.

W. Chen, Y. Chen, and K. Weinberger. Filtered search for submodular maximization with controllable approximation bounds. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics*, pages 156–164, 2015.

R. Combes, M. Sadegh Talebi, A. Proutiere, and M. Lelarge. Combinatorial bandits revisited. In *Advances in Neural Information Processing Systems 28*, pages 2116–2124, 2015.

M. Conforti and G. Cornuéjols. Submodular set functions, matroids and the greedy algorithm: Tight worst-case bounds and some generalizations of the Rado–Edmonds theorem. *Discrete Appl. Math.*, 7(3):251–274, 1984.

O. Coudert. Solving graph optimization problems with ZBDDs. In *Proceedings of the European Design and Test Conference 1997*, pages 224–228, 1997.

V. Dani, S. M. Kakade, and T. P. Hayes. The price of bandit information for online optimization. In *Advances in Neural Information Processing Systems 20*, pages 345–352, 2008.

G. B. Dantzig. Discrete-variable extremum problems. *Oper. Res.*, 5(2):266–288, 1957.

A. Das and D. Kempe. Approximate submodularity and its applications: Subset selection, sparse approximation and dictionary selection. *J. Mach. Learn. Res.*, 19(3):1–34, 2018.

R. Diestel. *Graph Theory*. Springer, 2nd edition, 2000.

R. Ebendt and R. Drechsler. Weighted A* search — unifying view and application. *Artificial Intelligence*, 173(14):1310–1342, 2009.

E. R. Elenberg, R. Khanna, A. G. Dimakis, and S. Negahban. Restricted strong convexity implies weak submodularity. *Ann. Statist.*, 46(6B):3539–3568, 2018.

X. Fan, I. Grama, and Q. Liu. Hoeffding's inequality for supermartingales. *Stoch. Proc. Appl.*, 122(10):3545–3559, 2012.

H. Fawzi, J. Saunderson, and P. A. Parrilo. Sparse sum-of-squares certificates on finite abelian groups. In *Proceedings of the 54th IEEE Conference on Decision and Control*, pages 5909–5914, 2015.

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

Y. Filmus and J. Ward. Monotone submodular maximization over a matroid via non-oblivious local search. *SIAM J. Comput.*, 43(2):514–542, 2014.

M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral Combinatorics*, pages 73–87. Springer, 1978.

Y. Freund and R. E. Schapire. Adaptive game playing using multiplicative weights. *Games Econom. Behav.*, 29(1):79–103, 1999.

K. Fujisawa, M. Fukuda, K. Kobayashi, M. Kojima, K. Nakata, M. Nakata, and M. Yamashita. SDPA (SemiDefinite Programming Algorithm) user's manual – Version 7.0.5. Technical report, Research Report B-448, Dept. of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2008.

N. Garg, V. S. Santosh, and A. Singla. Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques. In *Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 103–111, 1993.

M. X. Goemans and D. P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. ACM*, 42(6):1115–1145, 1995.

I. Gridchyn and V. Kolmogorov. Potts model, parametric maxflow and k-submodular functions. In *Proceedings of the IEEE International Conference on Computer Vision 2013*, pages 2320–2327, 2013.

A. György, T. Linder, G. Lugosi, and G. Ottucsák. The on-line shortest path problem under partial monitoring. *J. Mach. Learn. Res.*, 8(Oct):2369–2403, 2007.

F. M. Harper and J. A. Konstan. The MovieLens datasets: History and context. *ACM Trans. Interact. Intell. Syst.*, 5(4):19:1–19:19, 2015. https://grouplens.org/datasets/movielens/. Last accessed 23 August 2017.

S. He, Z. Li, and S. Zhang. Approximation algorithms for discrete polynomial optimization. *J. Oper. Res. Soc. China*, 1(1):3–36, 2013.

K. Heeger and J. Vygen. Two-connected spanning subgraphs with at most $\frac{10}{7}$ OPT edges. *SIAM J. Discrete. Math.*, 31(3):1820–1835, 2017.

C. Hegde, P. Indyk, and L. Schmidt. A nearly-linear time framework for graph-structured sparsity. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 928–937, 2015.

A. Hertz. Polynomially solvable cases for the maximum stable set problem. *Discrete Appl. Math.*, 60(1):195–210, 1995.

H. Hirai and Y. Iwamasa. On k-submodular relaxation. *SIAM J. Discrete. Math.*, 30(3):1726–1736, 2016.

A. Huber and V. Kolmogorov. Towards minimizing k-submodular functions. In *Proceedings of the 2nd International Symposium on Combinatorial Optimization*, pages 451–462, 2012.

M. Imase and B. M. Waxman. Dynamic Steiner tree problem. *SIAM J. Discrete. Math.*, 4(3):369–384, 1991.

Y. Inoue and S. Minato. Acceleration of ZDD construction for subgraph enumeration via path-width optimization. Technical report, TCS-TR-A-16-80, Hokkaido University, 2016.

T. Inoue, H. Iwashita, J. Kawahara, and S. Minato. Graphillion: software library for very large sets of labeled graphs. *Int. J. Software Tool. Tech. Tran.*, 18(1):57–66, 2016.

M. Ishihata and T. Sato. Bayesian inference for statistical abduction using Markov chain Monte Carlo. In *Proceedings of the 3rd Asian Conference on Machine Learning*, pages 81–96, 2011.

M. Ishihata, Y. Kameya, T. Sato, and S. Minato. Propositionalizing the EM algorithm by BDDs. In *Proceedings of the 18th International Conference on Inductive Logic Programming*, pages 44–49, 2008.

S. Iwata, S. Tanigawa, and Y. Yoshida. Bisubmodular function maximization and extensions. Technical report, Technical Report METR 2013-16, The University of Tokyo, 2013.

S. Iwata, S. Tanigawa, and Y. Yoshida. Improved approximation algorithms for k-submodular function maximization. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 404–413, 2016.

S. Kale, L. Reyzin, and R. E. Schapire. Non-stochastic bandit slate problems. In *Advances in Neural Information Processing Systems 23*, pages 1054–1062, 2010.

Y. Kawahara, K. Nagano, K. Tsuda, and J. A. Bilmes. Submodularity cuts and applications. In *Advances in Neural Information Processing Systems 22*, pages 916–924, 2009.

J. Kawahara, T. Inoue, H. Iwashita, and S. Minato. Frontier-based search for enumerating all constrained subgraphs with compressed representation. *IEICE Trans. Fund. Electron. Comm. Comput. Sci.*, E100.A(9):1773–1784, 2017.

J. Kawahara, T. Saitoh, H. Suzuki, and R. Yoshinaka. Solving the longest oneway-ticket problem and enumerating letter graphs by augmenting the two representative approaches with ZDDs. In *Proceedings of the Computational Intelligence in Information Systems Conference 2017*, pages 294–305, 2017.

W. C. Ke, B. H. Liu, and M. J. Tsai. Efficient algorithm for constructing minimum size wireless sensor networks to fully cover critical square grids. *IEEE Trans. Wireless Commun.*, 10(4):1154–1164, 2011.

D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137–146, 2003.

S. Khuller, A. Moss, and J. S. Naor. The budgeted maximum coverage problem. *Inform. Process. Lett.*, 70(1):39–45, 1999.

R. Kleinberg, G. Piliouras, and E. Tardos. Multiplicative updates outperform generic no-regret learning in congestion games: Extended abstract. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 533–542, 2009.

S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. The Internet topology zoo. *IEEE J. Sel. Areas Commum.*, 29(9):1765–1775, 2011.

D. E. Knuth. *The Art of Computer Programming: Combinatorial Algorithms, Part 1*, volume 4A. Addison-Wesley Professional, 1st edition, 2011.

C.-W. Ko, J. Lee, and M. Queyranne. An exact algorithm for maximum entropy sampling. *Oper. Res.*, 43(4):684–691, 1995.

T. Kocák, G. Neu, M. Valko, and R. Munos. Efficient learning by implicit exploration in bandit problems with side observations. In *Advances in Neural Information Processing Systems 27*, pages 613–621, 2014.

W. M. Koolen, M. K. Warmuth, and J. Kivinen. Hedging structured concepts. In *Proceedings of the 23rd Annual Conference on Learning Theory*, pages 93–105, 2010.

B. Korte and J. Vygen. *Combinatorial Optimization*. Springer, 5th edition, 2012.

A. Krause and D. Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*, pages 71—104. Cambridge University Press, 2014.

A. Krause, C. Guestrin, A. Gupta, and J. Kleinberg. Near-optimal sensor placements: Maximizing information while minimizing communication cost. In *Proceedings of the 5th International Conference on Information Processing in Sensor Networks*, pages 2–10, 2006.

A. Krause, H. B. McMahan, C. Guestrin, and A. Gupta. Robust submodular observation selection. *J. Mach. Learn. Res.*, 9(Dec):2761–2801, 2008.

A. Krause, A. Singh, and C. Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *J. Mach. Learn. Res.*, 9(Feb):235–284, 2008.

R. Kumar, B. Moseley, S. Vassilvitskii, and A. Vattani. Fast greedy algorithms in mapreduce and streaming. *ACM Trans. Parallel Comput.*, 2(3):14:1–14:22, 2015.

A. Kurpisz, S. Leppänen, and M. Mastrolilli. Tight sum-of-squares lower bounds for binary polynomial optimization problems. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming*, pages 78:1–78:14, 2016.

A. Kurpisz, S. Leppänen, and M. Mastrolilli. On the hardest problem formulations for the 0/1 Lasserre hierarchy. *Math. Oper. Res.*, 42(1):135–143, 2017.

J. B. Lasserre. An explicit exact SDP relaxation for nonlinear 0-1 programs. In *Proceedings of the 8th Integer Programming and Combinatorial Optimization*, pages 293–303, 2001.

J. B. Lasserre. Global optimization with polynomials and the problems of moments. *SIAM J. Optim.*, 11(3):796–817, 2001.

M. Laurent. Lower bound for the number of iterations in semidefinite hierarchies for the cut polytope. *Math. Oper. Res.*, 28(4):871–883, 2003.

J. Lee, V. S. Mirrokni, V. Nagarajan, and M. Sviridenko. Maximizing nonmonotone submodular functions under matroid or knapsack constraints. *SIAM J. Discrete. Math.*, 23(4):2053–2078, 2010.

C.-Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Syst. Tech. J.*, 38(4):985–999, 1959.

H. Lin and J. Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920, 2010.

C. Ling, J. Nie, L. Qi, and Y. Ye. Biquadratic optimization over unit spheres and semidefinite programming relaxations. *SIAM J. Optim.*, 20(3):1286–1310, 2009.

X. Liu, L. Xiao, A. Kreling, and Y. Liu. Optimizing overlay topology by reducing cut vertices. In *Proceedings of the 2006 International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 17:1–17:6, 2006.

Z. Luo and S. Zhang. A semidefinite relaxation scheme for multivariate quartic polynomial optimization with quadratic constraints. *SIAM J. Optim.*, 20(4):1716–1736, 2010.

S. Madden. Intel Lab Data, 2004. http://db.csail.mit.edu/labdata/labdata.html. Last accessed 30 November 2016.

T. Maehara, Y. Kawase, H. Sumita, K. Tono, and K. Kawarabayashi. Optimal pricing for submodular valuations with bounded curvature. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence*, 2017.

H. B. McMahan and A. Blum. Online geometric optimization in the bandit setting against an adaptive adversary. In *Proceedings of the 17th Annual Conference on Learning Theory*, pages 109–123, 2004.

S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In *Proceedings of the 30th International Design Automation Conference*, pages 272–277, 1993.

M. Mohri. Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.

D. R. Morrison, E. C. Sewell, and S. H. Jacobson. Solving the pricing problem in a branch-and-price algorithm for graph coloring using zero-suppressed binary decision diagrams. *INFORMS J. Comput.*, 28(1):67–82, 2016.

G. L. Nemhauser and L. A. Wolsey. Best algorithms for approximating the maximum of a submodular set function. *Math. Oper. Res.*, 3(3):177–188, 1978.

G .L. Nemhauser and L. A. Wolsey. Maximizing submodular set functions: Formulations and analysis of algorithms. In *Annals of Discrete Mathematics (11)*, pages 279–301. North-Holland, 1981.

G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions—I. *Math. Program.*, 14(1):265–294, 1978.

N. Ohsaka and Y. Yoshida. Monotone k-submodular function maximization with size constraints. In *Advances in Neural Information Processing Systems 28*, pages 694–702, 2015.

T. Opsahl. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks*, 35(6):159–167, 2013. https://toreopsahl.com/datasets/#online_forum_network. Last accessed 23 August 2017.

G. Palaiopanos, I. Panageas, and G. Piliouras. Multiplicative weights update with constant step-size in congestion games: convergence, limit cycles and chaos. In *Advances in Neural Information Processing Systems 30*, pages 5872–5882, 2017.

I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970.

H. Rahmanian, S. Vishwanathan, and M. K. Warmuth. Online dynamic programming. In *Advances in Neural Information Processing Systems 30*, pages 2827–2837, 2017.

F. Rendl and H. Wolkowicz. A semidefinite framework for trust region subproblems with applications to large scale minimization. *Math. Program.*, 77(1):273–299, 1997.

H. Robbins. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.*, 58(5):527–535, 1952.

A. Roest and A. R. Mashariki. Open data for all, 2015. The used dataset is "City Subway Stations" available at NYC Open Data (https://data.cityofnewyork.us/Transportation/Subway-Stations/arq3-7z49/data). Last accessed 23 August 2017.

R. W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *Internat. J. Game Theory*, 2(1):65–67, 1973.

S. Sakaue. Greedy and IHT algorithms for non-convex optimization with monotone costs of non-zeros. In *Processings of the 22nd International Conference on Artificial Intelligence and Statistics*, pages 206–215, 2019.

R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley Professional, 4th edition, 2011.

D. Sharma, A. Kapoor, and A. Deshpande. On greedy maximization of entropy. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1330–1338, 2015.

A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser. Efficient informative sensing using multiple robots. *J. Artif. Intell. Res.*, 34(1):707–755, 2009.

J. F. Sturm. SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods and Softw.*, 11(1–4):625–653, 1999.

H. Suzuki and S. Minato. Adding the vertex indices for enumerating and indexing the graphs via ZDD (in Japanese). *Special Interest Group on Fundamental Problems in Artificial Intelligence*, 101:41–46, 2016.

H. Suzuki. Frontier based search with vertex indices. https://github.com/hsnazuna/FrontierBasedSearchWithVertexIndices. Last accessed 1 September 2017., 2016.

M. Sviridenko, J. Vondrák, and J. Ward. Optimal approximation for submodular and supermodular optimization with bounded curvature. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1134–1148, 2015.

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

E. Takimoto and M. K. Warmuth. Path kernels and multiplicative updates. *J. Mach. Learn. Res.*, 4(Oct):773–818, 2003.

M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1076–1087, 2009.

R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Math. Program.*, 95(2):189–217, 2003.

T. Uchiya, A. Nakamura, and M. Kudo. Algorithms for adversarial bandit problems with multiple plays. In *Proceedings of the 21st International Conference on Algorithmic Learning Theory*, pages 375–389, 2010.

H. Waki, S. Kim, M. Kojima, and M. Muramatsu. Sums of squares and semidefinite programming relaxations for polynomial optimization problems with structured sparsity. *SIAM J. Optim.*, 17:218–242, 2006.

J. Ward and S. Živný. Maximizing k-submodular functions and beyond. *ACM Trans. Algorithms*, 12(4):47:1–47:26, 2016.

S. Xiong and J. Li. An efficient algorithm for cut vertex detection in wireless sensor networks. In *Proceedings of the 30th International Conference on Distributed Computing Systems*, pages 368–377, 2010.

Y. Yang and Q. Yang. On solving biquadratic optimization via semidefinite relaxation. *Comput. Optim. Appl.*, 53:845–867, 2012.

Y. Yoshida. Maximizing a monotone submodular function with a bounded curvature under a knapsack constraint. *SIAM J. Discrete. Math.*, 33(3):1452–1471, 2019.

Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza, and Y. Xiang. Optimal route search with the coverage of users' preferences. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 2118–2124, 2015.

H. Zhang and Y. Vorobeychik. Submodular optimization with routing constraints. In *Proceedings of the 30th AAAI Conference on Artificial Intelligence*, pages 819–825, 2016.

Y. Zheng, F. Liu, and H.-P. Hsieh. U-Air: when urban air quality inference meets big data. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1436–1444, 2013.