Efficient Branch-and-Cut Algorithms for Submodular Function Maximization

Submitted to
Graduate School of Information Science and Technology
Osaka University

January 2020

Naoya UEMATSU

List of Publications

Journal Papers

1. Uematsu, N., S. Umetani, and Y. Kawahara. An efficient branch-and-cut algorithm for submodular function maximization. *Journal of the Operations Research Society of Japan 63*. (accepted for publication).

Proceedings

1. ©[2019] IEEE. Reprinted, with permission, from [Uematsu, N., S. Umetani, and Y. Kawahara (2019, Nov.). An efficient branch-and-cut algorithm for approximately submodular function maximization. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 3160–3167.]

International Conferences

1. Uematsu, N., S. Umetani, and H. Morita (2017, Oct.). An integer programming model of route assignment problem for reducing traffic congestion in urban area. *INFORMS annual meeting*, Houston.

Technical Reports

1. Uematsu, N., S. Umetani, Y. Kawahara (2018, Nov.). An efficient branch-and-bound algorithm for submodular function maximization, *The institute*

of electronics, information and communication engineers. 183–190. (written in Japanese)

Domestic Conferences

1. Uematsu, N., S. Umetani, H. Morita (2017, Nov.). Route assignment problem for reducing traffic congestion, *Information and Physical Sciences (IPS2017)*, Osaka. (written in Japanese)

2. Uematsu, N., S. Umetani, H. Morita (2017, Oct.). Route recommendation for reducing traffic congestion, *The Operations Research Society of Japan, Kansai, Young researchers conference'17*, Osaka. (written in Japanese)

3. Uematsu, N., S. Umetani, H. Morita (2017, Dec.). Route assignment problem for reducing traffic congestion of train, *the 33rd Osaka Advanced Research Collaboration Forum for Information Science and Technology (OACIS)*, Osaka. (written in Japanese)

4. Uematsu, N., S. Umetani, Y. Kawahara (2018, Nov.). An efficient branch-and-bound algorithm for submodular function maximization, *The 21st Information-Based Induction Sciences Workshop (IBIS2018)*, Sapporo. (written in Japanese)

5. Uematsu, N., S. Umetani, Y. Kawahara (2019, Sep.). An efficient branch-and-cut algorithm for submodular function maximization, *The Operations Research Society of Japan, fall conference*, Hiroshima. (written in Japanese)

6. Uematsu, N., S. Umetani, Y. Kawahara (2019, Nov.). An efficient branch-and-cut algorithm for approximately submodular maximization, *The 22nd Information-Based Induction Sciences Workshop (IBIS2019)*, Nagoya. (written in Japanese)

Preface

This thesis considers combinatorial optimization problems arising from machine learning and data mining areas, where we select a subset of a finite set for maximizing or minimizing some utility function. Some of such utility functions are known to be submodular which is expressed as discrete analogue of convex functions. A set function $f\colon 2^N \to \mathbb{R}$ is called *submodular* if it satisfies $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$ for all $S \subseteq T \subseteq N$ and $i \notin T$, where $N = \{1, \ldots, n\}$ is a finite set. The property of submodularity in information science and economic areas are called the *law of diminishing marginal utility* or *Gossen's laws*, respectively. The submodular function, maximization and minimization, problems have been studied for the last several decades. For the minimization problem, many efficient polynomial time algorithms have been proposed. Unlike the minimization problem, the maximization problem has been known as NP-hard, but it began to be used for formulating facility location, sensor placement, document summarization, feature selection, and influence maximization problems. Since then, the submodular function maximization problem has played an important role in computer science, machine learning, and data mining areas.

The first topic is the submodular function maximization problem under a cardinality constraint. Although it is known that a variety of greedy algorithms find feasible solutions quickly, we often encounter applications that ask an optimal solution or solutions better than a solution obtained by greedy algorithms. Therefore, our objective is to propose an *exact algorithm* that obtains an optimal solution within a reasonable computation time. The conventional exact algorithm, con-

straint generation algorithm proposed by Nemhauser and Wolsey, is not efficient for some real applications, therefore we propose an improved constraint generation algorithm which generates a promising set of constraints at each iteration. To improve the efficiency, we incorporate it into a branch-and-cut algorithm.

The second topic is an approximately submodular function maximization problem under a cardinality constraint. Since some real applications such as active learning and feature selection problems do not hold submodularity exactly, an approximate version of submodular function has been considered recently. We formulate the problem into a binary integer programming (BIP) problem with an exponential number of constraints. We propose three exact algorithms including a branch-and-cut algorithm based on the BIP formulation. Due to the BIP formulation, our algorithms have a potential to deal with problems under not only a cardinality constraint but also any linear constraints.

The studies on the submodular function maximization problems are still developing, and many important problems remain to be considered in this area. The author hopes that this research provides useful ideas to solve the problems.

January, 2020

Naoya Uematsu

Acknowledgment

This thesis would not have been possible without the support of many others. First of all, I am deeply grateful to Dr. Shunji Umetani of Osaka University who provided me with lots of great opportunities to learn new materials. I greatly appreciate his invaluable support for my research, methodology, all domestic and international conferences I attended, and all documents I wrote.

I would like to offer my special thanks to Dr. Yoshinobu Kawahara of Kyushu University, who provided me with the topics and ideas for my research. He provided me with lots of tremendous opportunities where I learn new topics and area.

I also would like to thank Dr. Hiroshi Morita of Osaka University who provided me with lots of equipment and the opportunity to study in the Ph.D program.

I would like to thank all laboratory members, especially Naoya Takimoto and Yuya Masumura for meaningful advice.

I appreciate professors of Osaka University, Dr. Yasumasa Fujisaki, Dr. Hiroshi Morita, Dr. Hideyuki Suzuki, and Dr. Yutaro Yamaguchi for giving me useful comments.

I also would like to appreciate RIKEN Center Advanced Intelligence Project for the great support.

I greatly appreciate my wife, my parents, and my grandmother who supported me mentally and financially. I could not continue this Ph.D program without one of my family.

Contents

Chapter 1

Introduction

1.1 Submodular Function Maximization

When we approach problems in computer science, we often encounter situations where we need to select a subset of a finite set for maximizing some utility function f which is often called the *objective function*. First, we consider a combinatorial problem which is defined as follows:

$$\begin{aligned} &\text{maximize} \quad f(S) \\ &\text{subject to} \quad S \in F, \end{aligned} \tag{1.1}$$

where F is the set of solutions S which meet all the given constraints (e.g. cardinality constraint or budget constraint et al.). A set F represents the *feasible region* and each $S \in F$ is called a *feasible solution*. If $f(S^*) \geq f(S)$ holds for all other feasible solutions $S \in F$, then we call the solution S^* *optimal solution*. An optimization problem (1.1) is called a *combinatorial optimization problem* if a region F is combinatorial due to the given constraints [81]. We often encounter combinatorial optimization problems in many real applications such as facility location [22, 49, 51, 63, 67], vehicle routing [37, 42], scheduling [2, 9], etc. Next, a combinatorial optimization problem under a cardinality constraint is defined as follows:

$$\begin{aligned} &\text{maximize} \quad f(S) \\ &\text{subject to} \quad |S| \leq k,\ S \subseteq N, \end{aligned} \tag{1.2}$$

where a finite set $N = \{1, \ldots, n\}$ and a positive integer $k \leq n$. k represents a cardinality constraint for the size of S.

A general case of a cardinality constraint is a knapsack constraint. We have a set of items $N = \{1, \ldots, n\}$ and their sizes w_i for each $i \in N$, and the knapsack capacity B. The objective is to find a subset $S \subseteq N$ for maximizing the total value $f(S)$ of items where the total size $\sum_{i \in S}\ w_i$ of items is B or less. If $B = k$ and each $w_i = 1$ for any $i \in N$, then we can consider that the knapsack constraint is a general case of the cardinality constraint. A combinatorial optimization problem under a knapsack constraint is defined as follows:

$$\begin{aligned} &\text{maximize} \quad f(S) \\ &\text{subject to} \quad \sum_{i \in S} w_i \leq B,\ S \subseteq N. \end{aligned} \tag{1.3}$$

If we are given a subset of a whole set $N = \{1, 2, 3\}$ with a cardinality constraint $|S| \leq 2$ $(k = 2)$, then $F = \{\{\},\ \{1\},\ \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$ including an empty set. When we encounter problems with the large sizes n and k, we need to consider an exponential number of feasible solutions to obtain an optimal solution. This phenomena is called *combinatorial explosion*, and it is the main reason why we often have a difficulty to obtain the optimal solutions in many applications. When we have a function without any structure which describes the data, to enumerate all feasible solutions (brute-force computation) is the only option, which is often very difficult within a realistic computation time.

In this thesis, we focus on functions that have *submodularity*, which are called *submodular function* or *submodular set function*. The submodular function can be considered as discrete counterparts of convex functions through the continuous relaxation called the *Lovász extension* [61]. Problems such as active learning [50], document summarization [55, 56, 57], sensor placement [46, 49], influence spread [47], and feature selection [18, 44, 85] are formulated as the problem max-

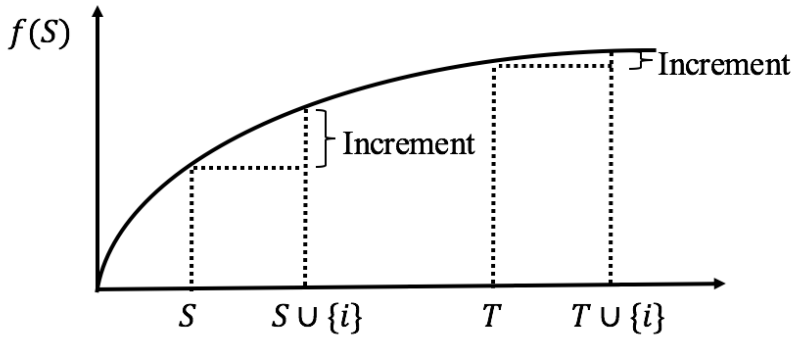

Fig. 1.1: Image of submodular function.

imizing a submodular function, which is called *submodular function maximization problem* which has been known as NP-hard. We introduce one of the most common definitions of submodular function.

Definition 1.1.1. *A set function* $f\colon 2^N \to \mathbb{R}$ *is submodular if it satisfies*

$$f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$$

for all $S \subseteq T \subseteq N$ *and* $i \notin T$*, where* $N = \{1, \ldots, n\}$ *is a finite set* [67, 68, 84]. *Another representation of submodular function: A set function* f *is submodular if it satisfies*

$$f(S) + f(T) \geq f(S \cap T) + f(S \cup T)$$

for all $S, T \subseteq N$.

When we add an element $i \notin T$ into two sets $S, T; S \subseteq T \subseteq N$, we can obtain a marginal gain for each S, T. The marginal gain is represented as $f(S \cup \{i\}) - f(S)$ (we often use $f(\{i\} \mid S)$ later for convenience). The marginal gain of a small set S when adding an element i is always greater or equal to the marginal gain of a large set T (see Figure 1.1). In economic area, this property is called *law of diminishing marginal utility* or *Gossen's laws*. In information science or economic area, we often encounter functions which have the law of diminishing marginal utility. That is because the law from the submodular function shows the essential

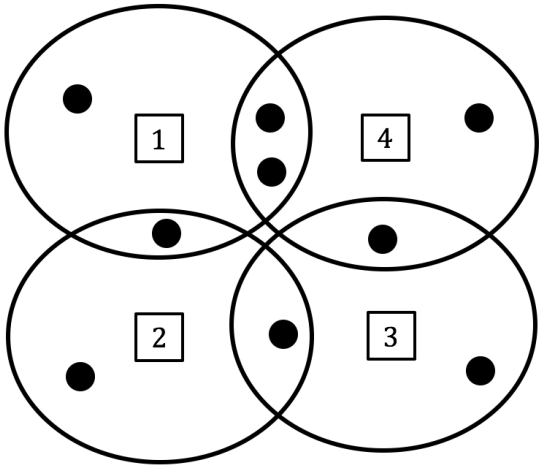

Fig. 1.2: Image of sensor placement.

property of the information. Intuitively, we gain more information when we have less information, compared to the situations where we have more information.

To review the submodular maximization problem under a cardinality constraint [11, 13, 30, 46, 67] and a knapsack constraint [70], we prepare three well-known examples of submodular function such as sensor placement [52, 53, 67], document summarization [55, 56, 57, 58], and graph cut problems [29, 35, 45]. We review a variety of greedy algorithms and the approximation ratios for the three problems.

(Sensor placement problem) For the first example of submodular function, we consider a sensor placement problem with a coverage function. The objective of the problem is to locate only $k < n$ sensors in the n candidate places for maximizing the number of buildings in the park covered by the sensors. We let the set of candidate places to locate sensors, $N = \{1, \ldots, n\}$. We select a set of places to locate sensors $S \subseteq N$. Given a set S, we let $f(S)$ be the number of buildings covered by the set S of sensors. We use Figure 1.2 as an example. In Figure 1.2, we define a set of sensors $N = \{1, 2, 3, 4\}$, black dots which represent buildings, and circles which represent the areas covered by sensors. We see some simple examples of the function f as follows:

When only sensor 1 is located, then four buildings are covered; $f(\{1\}) = 4$.

When only sensor 2 is located, then three buildings are covered; $f(\{2\}) = 3$.

When sensors 1, 2 are located, then six buildings are covered; $f(\{1,2\}) = 6$.

Since sensor 1 and sensor 2 cover the common building, we obtain $f(\{1\}) + f(\{2\}) \neq f(\{1,2\})$. By the relationship, we see that for the cover function f does not hold linearity.

Definition 1.1.2. *A function f is subadditive if it satisfies $f(S)+f(T) \geq f(S \cup T)$ for all $S, T \subseteq N$* [20, 25, 26].

This function is subadditive and also submodular, and we will check the submodularity later. We note that a function f is *superadditive* if it satisfies $f(S) + f(T) \leq f(S \cup T)$ for all $S, T \subseteq N$. With the whole set $N = \{1, 2, 3, 4\}$, we have 16 $(= 2^4)$ subsets including the empty set $\{\}$ and the whole set N. Therefore, the function $f : 2^N \to \mathbb{R}$ has values for the each subset. We check the submodularity of the function f; again $f(S \cup \{i\}) - f(S) \geq f(T \cup \{i\}) - f(T)$ for all $S \subseteq T \subseteq N$ and $i \notin T$. For example for checking submodularity of f, we consider two sets $\{1\} \subseteq \{1, 2\}$, and we add $\{3\}$ into these two sets.

The increment of $\{1\}$: $f(\{1\} \cup \{3\}) - f(\{1\}) = 7 - 4 = 3$.

The increment of $\{1, 2\}$: $f(\{1, 2\} \cup \{3\}) - f(\{1, 2\}) = 8 - 6 = 2$.

If we check for all pairs of sets, we can check the function f is submodular. Intuitively, we can tell that the function f is *non-decreasing*, which is defined as follows.

Definition 1.1.3. *A set function is non-decreasing (monotone) if $f(S) \leq f(T)$ for all $S \subseteq T$ and $f(\emptyset) = 0$.*

We note the following relationship between monotone submodular function and monotone subadditive function. We have the property that monotone submodular

functions are monotone subadditive functions. However, *monotone subadditive functions are not always monotone submodular functions.* For instance, we have a function f returning the following values: $f(\{1\}) = 3, f(\{2\}) = 4, f(\{3\}) = 2, f(\{1,2\}) = 6, f(\{1,3\}) = 4, f(\{1,2,3\}) = 8$. This function is subadditive. We check this function f is not submodular by considering two sets $\{1\} \subseteq \{1,3\}$, and we add $\{2\}$ into these two sets.

The increment of $\{1\}$: $f(\{1\} \cup \{2\}) - f(\{1\}) = 6 - 3 = 3$.

The increment of $\{1,3\}$: $f(\{1,3\} \cup \{2\}) - f(\{1,3\}) = 8 - 4 = 4$.

Therefore, an algorithm proposed for monotone submodular functions can not be always applied for monotone subadditive functions.

(Greedy algorithm for a cardinality constraint) For a non-decreasing submodular function maximization problem, a greedy algorithm [84] is efficient for obtaining good feasible solutions relatively quickly. The algorithm starts with a feasible solution S and sets $S^{'} = S$. At each iteration, we obtain $i^* = \mathrm{argmax}_{i \in N \setminus S} f(S^{'} \cup \{i\}) - f(S^{'})$. If $|S^{'} \cup \{i^*\}| \leq k$ holds, then we add the element i^* into $S^{'}$. Otherwise, we output a feasible solution $S^{'}$ and exit. The pseudo code of the standard greedy algorithm is shown below.

Algorithm GR(S)

Input: The initial feasible solution S.

Output: A feasible solution $S^{'}$.

Step 1: Set $S^{'} \leftarrow S$.

Step 2: Obtain $i^* = \mathrm{argmax}_{i \in N \setminus S^{'}} f(S^{'} \cup \{i\}) - f(S^{'})$.

Step 3: If $|S^{'} \cup \{i^*\}| \leq k$ holds, then go to Step 4. Otherwise, output a feasible solution $S^{'}$ and exit.

Step 4: Set $S^{'} \leftarrow S^{'} \cup \{i^*\}$, and return to Step 2.

For instances with larger n and k, the standard greedy algorithm is often not efficient to obtain feasible solutions. We introduce an accelerated greedy algorithm proposed by Minoux [64] here.

(Accelerated greedy algorithm for a cardinality constraint) Minoux [64] presented an accelerated greedy algorithm which gives the same solutions obtained by the standard greedy algorithm. When we choose k elements out of n elements, the standard greedy algorithm spends $O(nk)$. However, Minoux's algorithm often spends much less computational cost than that of the standard greedy algorithm. In some applications, evaluating the function f is sometimes expensive. The standard greedy algorithm must identify the element $\{i\}$ with maximum marginal gain $f(S^{(t-1)} \cup \{i\}) - f(S^{(t-1)})$, where $S^{(t-1)}$ is the solution at $t-1$th iteration. When a function f is non-decreasing and submodular, then it is not necessary to compute the marginal gain for all elements $i \in N \setminus S^{(t-1)}$. For the limited space of the paper, we use the following notation $f(S \mid T) = f(T \cup S) - f(T)$. When we have submodularity, we have the following inequality such as $f(\{i\} \mid S^{(t)}) \leq f(\{i\} \mid S^{(t-1)})$ for any $S^{(t-1)} \subseteq S^{(t)}$ for any $i \in N \setminus S^{(t)}$. That is, the marginal gain is non-increasing during the iterations of the standard greedy algorithm. It is not necessary to compute the marginal gain $f(\{i\} \mid S^{(t)})$ for each $i \in N \setminus S^{(t)}$. The accelerated greedy algorithm maintains a list of marginal gain values in decreasing order. In each iteration, the algorithm computes the marginal gains for each element of the ordered list $L \subseteq N$. Suppose we check the marginal gains for some elements $L_0 \subseteq L$ from the top of the ordered list L. If $\max_{i \in L_0} f(\{i\} \mid S^{(t)}) \geq \max_{i \in L \setminus L_0} f(\{i\} \mid S^{(t-1)})$ holds, submodularity guarantees the following statement $\max_{i \in L \setminus L_0} f(\{i\} \mid S^{(t-1)}) \geq \max_{i \in L \setminus L_0} f(\{i\} \mid S^{(t)})$. Therefore, the algorithm selects the element which gives the maximum marginal gain. This idea of evaluation gives speed-up performance for many instances.

Proposition 1. *For the non-decreasing submodular function maximization problem under a cardinality constraint, a standard greedy algorithm* [67] *returns a solution S which achieves*

$$f(S) \geq \left(1 - \frac{1}{e}\right) f(S^*),$$

where S^* is an optimal solution.

The proof of the greedy algorithm is as follows [38, 67, 73]. We let $S^{(t)}$ be the solution obtained by the greedy algorithm at tth iteration.

Proof. By non-decreasing and submodularity, we obtain the following inequality. We let an optimal solution $S^* = \{i_1, i_2, ..., i_k\}$.

$$\begin{aligned}
f(S^*) &\leq f(S^{(t)} \cup S^*) \\
&= f(S^{(t)}) + f(S^{(t)} \cup \{i_1\}) - f(S^{(t)}) + \cdots + \\
&\quad f(S^{(t)} \cup \{i_1, ..., i_k\}) - f(S^{(t)} \cup \{i_1, ..., i_{k-1}\}) \\
&= f(S^{(t)}) + f(\{i_1\} \mid S^{(t)}) + f(\{i_2\} \mid S^{(t)} \cup \{i_1\}) + \cdots + \\
&\quad f(\{i_k\} \mid S^{(t)} \cup \{i_1, \ldots, i_{k-1}\}) \\
&\leq f(S^{(t)}) + f(\{i_1\} \mid S^{(t)}) + f(\{i_2\} \mid S^{(t)}) + \cdots + f(\{i_k\} \mid S^{(t)}) \\
&= f(S^{(t)}) + \{f(\{i_1\} \mid S^{(t)}) + f(\{i_2\} \mid S^{(t)}) + \cdots + f(\{i_k\} \mid S^{(t)})\} \\
&\leq f(S^{(t)}) + k\{f(S^{(t+1)}) - f(S^{(t)})\}.
\end{aligned}$$

By reformulation, we obtain:

$$\frac{1}{k}\{f(S^*) - f(S^{(t)})\} \leq f(S^{(t+1)}) - f(S^{(t)}).$$

Let $\delta_t = f(S^*) - f(S^{(t)})$, and we obtain:

$$\begin{aligned}
\frac{1}{k}\delta_t &\leq \delta_t - \delta_{t+1} \\
\delta_{t+1} &\leq \left(1 - \frac{1}{k}\right)\delta_t.
\end{aligned}$$

By induction, we have:

$$\delta_k \leq \left(1 - \frac{1}{k}\right)^k \delta_0.$$

We apply this well-known inequality $1 - x \leq e^{-x}$ to the previous inequality, and we obtain:

$$\delta_k \leq \left(1 - \frac{1}{k}\right)^k \delta_0 \leq \left(e^{-\frac{1}{k}}\right)^k \delta_0 = \frac{1}{e}\delta_0.$$

We use the definition of δ_t to reformulate, and obtain:

$$f(S^*) - f(S^{(k)}) \leq \frac{1}{e}\left(f(S^*) - f(\emptyset)\right).$$

By $f(\emptyset) = 0$, we obtain

$$f(S^{(k)}) \geq \left(1 - \frac{1}{e}\right) f(S^*).$$

□

(Curvature) Vondarák [82] showed that a curvature value c [6] which represents how much the marginal values $f(\{j\} \mid S)$ can decrease as a function of S. The *total curvature* of a monotone submodular function f is defined as follows.

$$c = 1 - \min_{S, j \notin S} \frac{f(\{j\} \mid S)}{f(\{j\})}. \tag{1.4}$$

If $c = 0$ then the marginal values are independent of S, that means f is *additive*. A function f is additive if $f(S) + f(T) = f(S \cup T)$ for all $S, T \subseteq N$. In the case, the greedy algorithm gives the optimal solution [82]. By using the structure and the feature of the submodular function, the standard greedy algorithm can attain a better approximation ratio, as follows [82].

Proposition 2. *For the non-decreasing submodular function maximization problem under a cardinality constraint, the greedy algorithm returns a solution S which achieves*

$$f(S) \geq \left(\frac{1 - e^{-c}}{c}\right) f(S^*),$$

where S^* is an optimal solution.

The details are in [82]. For example, when $c = 0.5$, the approximation ratio attains $\approx$ 0.78. Other cases and analysis of the curvature are given in [15].

(Document summarization problem) Document summarization problem is defined as follows. Suppose that we have a document including n sentences $N = \{1, \ldots, n\}$. The purpose of the problem is to select $k \leq n$ sentences $S \subseteq N$ to represent the original document as much as possible if we consider the problem under a cardinality constraint. In general, we can describe the document more with more sentences. However, the effect of an additional sentence becomes less when we have more sentences. Since the document summarization problem has the property similar to the law of diminishing marginal utility, the problem is often formulated as the submodular function maximization problem. To select variety of sentences, Lin and Bilmes [57] proposed the following properties called *relevance* and *non-redundancy*. The redundancy penalty usually violates the monotonicity of the objective functions [32, 56]. Although the redundancy has been considered with a negative penalizing function, Lin and Bilmes considered the redundancy as a positive reward. Lin and Bilmes [57] model the summary objective function as

$$f(S) = L(S) + \lambda R(S),$$

where $L(S)$ measures the coverage, or "fidelity" of a summary set S to the document, $R(S)$ rewards diversity in S, and $\lambda \geq 0$ is a trade-off coefficient.

For a function L, there are several functions to be applied. We introduce some candidate functions for L from [57], and the first idea is as follows: $L(S) = \sum_{i \in N, j \in S} w_{i,j}$, where $w_{i,j}$ represents the similarity between sentences i and j. Another way to define $L(S)$ is as follows: $L(S) = \sum_{i \in N} \max_{j \in S} w_{i,j}$. We have many other possible functions for acting as L. More details are given in [57]. These two functions above are both non-decreasing and submodular.

Lin and Bilmes [57] proposed a reward function R for representing diversity as follows:

$$R(S) = \sum_{i=1}^{t} \sqrt{\sum_{j \in P_i \cap S} r_j},$$

where $P_i, i = 1, \ldots, t$ is a partition of the whole set N (i.e., $\bigcup_i P_i = N$ and $P_i \cap P_j = \emptyset$ for all i, j) into separate clusters, and $r_i \geq 0$ represents the reward of

a single i. The value r_i is obtained due to the importance of i to the summary. Clearly, the function R behaves as the law of diminishing marginal utility. R is a non-decreasing submodular function due to the composition rule of Proposition 3 in [57].

Proposition 3. *Given functions $F : 2^N \to \mathbb{R}$ and $f : \mathbb{R} \to \mathbb{R}$, the composition $F^{'} = f \circ F : 2^N \to \mathbb{R}$ (i.e.,$F^{'}(S) = f(F(S))$) is non-decreasing submodular, if f is non-decreasing concave and F is non-decreasing submodular* [57].

Proposition 4. *Given any submodular function $f_1, \ldots, f_k$ and non-negative numbers $c_1, \ldots, c_k$. The function $g(S) = \sum_{i=1}^{k} c_i f_i(S)$ is submodular* [4, 72, 75].

Therefore, by using the summation rule of Proposition 4 we have the property that the sum of submodular functions is submodular. Since two functions L and R are both non-decreasing submodular, the function f is still non-decreasing submodular [57].

In addition, if we let l_i be the length of a sentence i, we can consider the problem under a knapsack constraint that $\sum_{i \in S} l_i \leq k$, where k is a knapsack capacity. For the non-decreasing submodular function maximization problem under the knapsack constraint, a variety of greedy algorithms [48, 57] exist. Here, we introduce some simple greedy algorithms [48, 57, 74].

(Greedy algorithm for a knapsack constraint) This algorithm starts with a feasible solution S and sets $S^{'} = S$. At each iteration, we obtain $i^* = \mathrm{argmax}_{i \in N \setminus S^{'}}(f(S^{'} \cup \{i\}) - f(S^{'}))/l_i$. If $\sum_{i \in S^{'}} l_i + l_{i^*} \leq k$ holds, then we add the element i^* into $S^{'}$. Otherwise, we output a feasible solution $S^{'}$ and exit. The pseudo code of the greedy algorithm for a knapsack constraint is shown below.

Algorithm KGR(S)

Input: The initial feasible solution S.

Output: A feasible solution $S^{'}$.

Step 1: Set $S^{'} \leftarrow S$.

Step 2: Obtain $i^{*} = \mathrm{argmax}_{i \in N \setminus S^{'}}(f(S^{'} \cup \{i\}) - f(S^{'}))/l_i$.

Step 3: If $\sum_{i \in S^{'}} l_i + l_{i^{*}} \leq k$ holds, then go to Step 4. Otherwise, output a feasible solution $S^{'}$ and exit.

Step 4: Set $S^{'} \leftarrow S^{'} \cup \{i^{*}\}$. Return to Step 2.

Proposition 5. *For the non-decreasing submodular function maximization problem under a knapsack constraint, whenever the first part of Step 3 (Algorithm KGR) evaluates to false, Algorithm KGR returns a solution $S^{'}$ which achieves*

$$f(S^{'} \cup \{i^{*}\}) \geq \left(1 - \frac{1}{e}\right) \ f(S^{*}),$$

where S^{*} is an optimal solution [38].

Proof. We let $S^{(t)} = \{i_1, \ldots, i_t\}$ be a feasible solution obtained by the algorithm at tth iteration. By non-decreasing and submodularity, we obtain the following inequalities [38].

$$\begin{aligned} f(S^{*}) &\leq f(S^{(t-1)}) + \sum_{i \in S^{*} \setminus S^{(t-1)}} f(\{i\} \mid S^{(t-1)}) \\ &= f(S^{(t-1)}) + \sum_{i \in S^{*} \setminus S^{(t-1)}} l_i \ \frac{f(\{i\} \mid S^{(t-1)})}{l_i} \\ &\leq f(S^{(t-1)}) + \frac{f(S^{(t)}) - f(S^{(t-1)})}{l_{i_t}} \sum_{i \in S^{*} \setminus S^{(t-1)}} l_i \\ &\leq f(S^{(t-1)}) + \frac{B}{l_{i_t}}(S^{(t)} - f(S^{(t-1)})), \end{aligned}$$

where the first inequality is by submodularity, the second one comes from the greedy algorithm, and the last one is from $\sum_{i \in S^*} \ l_i \leq B$. For convenience, we sometimes write $l_S = \sum_{i \in S} l_i$.

We subtract $B/w(x_i)$ from both sides and obtain the following inequality:

$$f(S^{(t)}) - f(S^*) \geq \left(1 - \frac{l_{i_t}}{B}\right)(f(S^{(t-1)}) - f(S^*)).$$

Solve the recursive inequality yields:

$$f(S^{(t)}) \geq \left(1 - \prod_{k=1}^{t}\left(1 - \frac{l_{i_t}}{B}\right)\right) f(S^*).$$

Next, we use the well-known inequality $1 - x \leq e^{-x}$ and we obtain the following inequality:

$$f(S^{(t)}) \geq \left(1 - \exp\left(\frac{-l_{S^{(t)}}}{B}\right)\right) \ f(S^*).$$

If the Step 4 is false at some iteration, then we still consider adding the i^* into $S^{'}$.

$$f(S^{'} \cup \{i^*\}) \geq \left(1 - \exp\left(\frac{-l_{S'} - l_{i^*}}{B}\right)\right) \ f(S^*).$$

We have the following condition $l_{S'} + l_{i^*} > B$, and the condition leads to the proof [38]. □

(Simple greedy algorithm for a knapsack constraint) The algorithm above does not hold an approximation guarantee because of the element i^* which should not be added into an obtained feasible solution. Therefore, we introduce a simple greedy algorithm [38] for the non-decreasing submodular function maximization problem under a knapsack constraint and the approximation guarantee of the algorithm. This algorithm starts with a feasible solution S, and we obtain an element $v^* = \text{argmax}_{v \in N, l_v \leq B} f(\{v\})$. We also obtain a faesible solution $S^{'} = \text{KGR}(S)$ obtained by the previous greedy algorithm. We finally obtain a feasible solution $S^{''} = \text{argmax}_{S \in \{S', \{v^*\}\}} f(S)$. The pseudo code of the simple greedy algorithm for a knapsack constraint is shown below.

Algorithm SKGR(S)

Input: The initial feasible solution S.

Output: A feasible solution $S^{''}$.

Step 1: Obtain $v^* \leftarrow \mathrm{argmax}_{v \in N, l_v \leq B} f(\{v\})$.

Step 2: Obtain a feasible solution $S^{'}$ by Algorithm KGR(S).

Step 3: Output a feasible solution $S^{''} \leftarrow \mathrm{argmax}_{S \in \{S', \{v^*\}\}} f(S)$ and exit.

Proposition 6. *For the non-decreasing submodular function maximization problem under a knapsack constraint, Algorithm SKGR returns a solution S which achieves*

$$f(S) \geq \frac{1}{2}\left(1 - \frac{1}{e}\right) f(S^*),$$

where S^* is an optimal solution [38].

Proof. We let $v^* = \mathrm{argmax}_{v \in N, l_v \leq B} f(\{v\})$. We let i^* be the element from Step 3 of Algorithm KGR at the iteration when the Step 3 becomes false.

$$\begin{aligned} 2f(S) &\geq f(S) + f(\{v^*\}) \geq f(S) + f(\{i^*\}) \\ &\geq f(S \cup \{i^*\}) \\ &\geq \left(1 - \frac{1}{e}\right) f(S^*), \end{aligned}$$

where the first and second inequalities are obtained clearly. The second to the last inequality is from submodularity, and finally the last one is from the previous proof [38]. □

In addition, Sviridenko [74] proposed an algorithm which obtains $(1 - 1/e)$-approximation for the problem under a knapsack constraint. However, the algorithm requires $O(n^5)$. Lin and Bilmes [56] proposed an algorithm which obtains $(1 - 1/\sqrt{e})$-approximation for the problem.

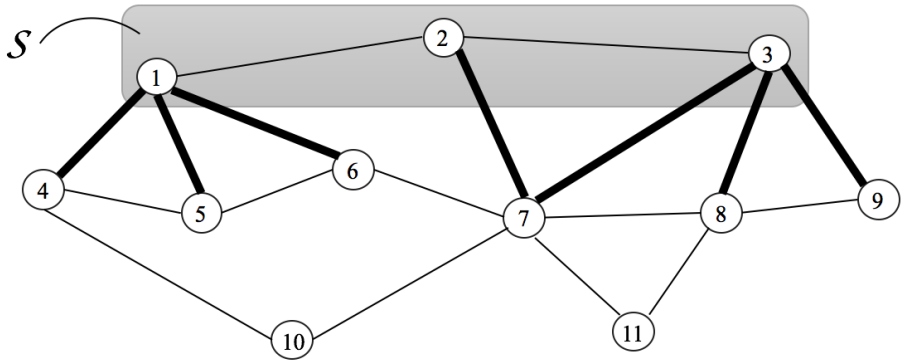

Fig. 1.3: Undirected graph.

(Graph cut problem: Undirected graph) We consider a graph cut problem which plays an important role in network problems or optimization problems related to graphs. We introduce a *cut function* of a graph, which has submodularity. First, we consider an *undirected graph* with edges that have no direction. Given a whole set of vertices $N = \{1, \ldots, n\}$ and a set of edges $\bar{E}$ for $i, j \in N$ where (i, j) is an unordered pair. We obtain the undirected graph $\bar{G} = \{N, \bar{E}\}$. For a subset of vertices $S \subseteq N$, we define $\bar{E}(S)$ represents a subset of a set of edges $\bar{E}$, where one vertex is included in the set S and the other vertex is included in the set $N \setminus S$. Figure 1.3 shows an example of an undirected graph with $N = \{1, 2, \ldots, 11\}$.

We consider that each edge $e \in \bar{E}$ in an undirected graph $\bar{G}$ has a capacity of the edge $c(e)$. We next define a cut function which returns a real value for each set of vertices $S \subseteq N$:

$$f_{\mathrm{u}}(S) = \sum_{e \in \bar{E}(S)} c(e). \tag{1.5}$$

$f_{\mathrm{u}}(S)$ sums up the all capacities $c(e)$ of each edge $e \in \bar{E}(S)$. We note that when S is $\emptyset$ or N, $f_{\mathrm{u}}(\emptyset) = f_{\mathrm{u}}(N) = 0$. We call a function $f_{\mathrm{u}} : 2^N \to \mathbb{R}$ a cut function of an undirected graph. In the problem, we have the following property $f_{\mathrm{u}}(S) = f_{\mathrm{u}}(N \setminus S)$ which is called *symmetric*.

Definition 1.1.4. *A function f is symmetric if $f(S) = f(N \setminus S)$ for all $S \subseteq N$* [27].

Proposition 7. *For any submodular function f and a set $S \subseteq N$, $\tilde{f}(S) = f(N \backslash S)$ is submodular.*

We consider the undirected graph of the Figure 1.3 and a cut function $f_{\mathrm{u}} : 2^N \to \mathbb{R}$, and we suppose that all capacity of edge are 1 for computational convenience. For a set $S = \{1, 2, 3\}$, $f(S) = 7$ which represents a set of the all bold edges in Figure 1.3.

In Figure 1.3, we obtain $|\bar{\mathcal{E}}(\{1\})| = 4$ which is $f_{\mathrm{u}}(\{1\}) = 3$ for a set $\{1\}$. For sets $\{1, 2\}, \{1, 3\}$, and $\{1, 2, 3\}$, we obtain $f_{\mathrm{u}}(\{1, 2\}) = 5, f_{\mathrm{u}}(\{1, 3\}) = 8$, and $f_{\mathrm{u}}(\{1, 2, 3\}) = 7$, respectively. We check the submodularity of the function with an example. If $S = \{1\}, T = \{1, 2\}, \{i\} = \{3\}$ in the definition 1.1.1 of submodular function, we have the following inequality: $f_{\mathrm{u}}(\{1, 3\}) - f_{\mathrm{u}}(\{1\}) \geq f_{\mathrm{u}}(\{1, 2, 3\}) - f_{\mathrm{u}}(\{1, 2\})$.

(Graph cut problem: Directed graph) Next, we consider a cut function for a directed graph whose each edge has a direction. An edge with a direction is represented as an arrow. Again, given a whole set of vertices $N = \{1, \ldots, n\}$ and a set of edges $(i, j) \in E$ for $i, j \in N$, where (i, j) is an ordered pair. An edge (i, j) represents an arrow from i to j. We obtain the directed graph $G = \{N, E\}$.

For a subset of vertices $S \subseteq N$, we define $E(S)$ represents a subset of the set of edges E, where the initial vertex i of an edge (i, j) is included in S and the terminal vertex j of the edge (i, j) is included in $N \setminus S$. We suppose that each edge e has the capacity of the edge $c(e) > 0$. In this situation, we define $f_{\mathrm{d}}(S)$ for a set of vertices $S \subseteq N$ as follows.

$$f_{\mathrm{d}}(S) = \sum_{e \in \mathcal{E}(S)} c(e). \tag{1.6}$$

Therefore, $f_{\mathrm{d}}(S)$ sums up the all capacities $c(e)$ of each edge $e \in \mathcal{E}(S)$. We note that when S is $\emptyset$ or N, $f_{\mathrm{d}}(\emptyset) = f_{\mathrm{d}}(N) = 0$. We call a function $f_{\mathrm{d}} : 2^N \to \mathbb{R}$ a cut function of a directed graph with the same manner for undirected graph. For a directed graph, the symmetric property, $f_{\mathrm{d}}(S) = f_{\mathrm{d}}(N \setminus S)$, does not hold. In

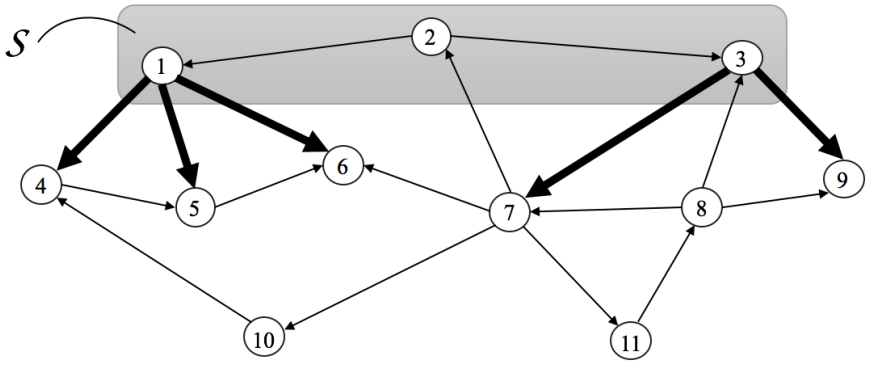

Fig. 1.4: Directed graph.

Figure 1.4, if a set $S = \{1, 2, 3\}$ holds, then $f_{\mathrm{d}}(S) = 5$ which represents a set of the all bold edges in the Figure.

Again, in the example of the Figure 1.4, we obtain $|E(\{1\})| = 3$ which is $f_{\mathrm{d}}(\{1\}) = 3$ for a set $\{1\}$. For sets $\{1, 2\}, \{1, 3\}$, and $\{1, 2, 3\}$, we obtain $f_{\mathrm{d}}(\{1, 2\}) = 4$, $f_{\mathrm{d}}(\{1, 3\}) = 5$, and $f_{\mathrm{d}}(\{1, 2, 3\}) = 5$, respectively. We check the submodularity of the function with an example. If $S = \{1\}, T = \{1, 2\}, \{i\} = \{3\}$ in the definition 1.1.1 of submodular function, $f_{\mathrm{d}}(\{1, 3\}) - f_{\mathrm{d}}(\{1\}) \geq f_{\mathrm{d}}(\{1, 2, 3\}) - f_{\mathrm{d}}(\{1, 2\})$.

Proposition 8. *The cut function f_{d} for a directed graph is submodular.*

Proof. This proof is from [45]. Given any two sets $S, T \subseteq N$. Here, we suppose that

$$N_1 = S \setminus T,\ N_2 = T \setminus S,\ N_3 = S \cap T,\ N_4 = N \setminus (S \cup T).$$

By the formulations above, N is divided into four parts as $N = N_1 \cup N_2 \cup N_3 \cup N_4$. Moreover, we obtain the followings; $S = N_1 \cup N_3$, $T = N_2 \cup N_3$, $S \cup T = N_1 \cup N_2 \cup N_3$. We also define $D_{a,b}$ which is the sum of all edges that start from N_a and get into N_b.

$$\begin{aligned}
f_{\mathrm{d}}(S) &= D_{1,2} + D_{1,4} + D_{3,2} + D_{3,4} \\
f_{\mathrm{d}}(T) &= D_{2,1} + D_{2,4} + D_{3,2} + D_{3,4} \\
f_{\mathrm{d}}(S \cup T) &= D_{1,4} + D_{2,4} + D_{3,4} \\
f_{\mathrm{d}}(S \cap T) &= D_{1,3} + D_{3,2} + D_{3,4}
\end{aligned}$$

Again, submodular function is represented as $f(S)+f(T) \geq f(S \cap T)+f(S \cup T)$ for all $S, T \subseteq N$, and we obtain the following inequality: $f_{\mathrm{d}}(S) + f_{\mathrm{d}}(T) - f_{\mathrm{d}}(S \cup T) - f_{\mathrm{d}}(S \cap T) = D_{1,2} + D_{2,1} \geq 0$. We conclude that f_{d} is submodular function. $\square$

In addition, due to the relationship between undirected and directed graphs, we can also show that the cut function f_{u} for an undirected graph is submodular (see details in [45]).

1.2 Approximately Submodular Function Maximization

Meanwhile, in many practical situations, utility functions may not necessarily have the property of *exact* submodularity. However, submodularity can be approximately satisfied in various problems such as feature selection [18, 19, 85], boosting influence spread [59, 60], data summarization [7, 65] and combinatorial auction [16, 80]. With many reasons such as the noise from data, hypothesis function values, or naturally close to submodular function, we consider an approximately version of submodular function, which is often called *approximately submodular function* or *weakly submodular function*. According to [18, 36, 39], the approximately version has been considered because of the following reasons.

Revealed preference theory *"Luce's famouse model assumes that an agent's revealed utility $\tilde{f} : 2^N \to \mathbb{R}$ can be approximated by a well-behaved utility function $f : 2^N \to \mathbb{R}$ such that $\tilde{f}(S) = f(S)+\xi_S$ for every $S \subseteq N$ where ξ_S is drawn i.i.d from a distribution* [12]*. $\tilde{f}$ is a utility function that approximates f, and multiple queries to $\tilde{f}$ return the same response"* [36].

Active learning *"There is a long line of work on noise-robust learning where one has access to a noisy membership oracle $\tilde{f}(x) = f(x)+\xi_x$ and for every x we have that ξ_x is drawn i.i.d. from a distribution* [3, 10, 28, 31, 41, 71]*. In this model as well, the oracle is consistent and multiple queries return the same*

response. For set functions, one can consider active learning in experimental design applications where the objective function is often submodular and the goal would be to optimize $f : 2^N \to \mathbb{R}$ *given* $\tilde{f}$*."* [36]

Statistics and learning theory *"The assumption in learning is that the data we observe is generated by* $\tilde{f}(x) = f(x) + \xi_x$ *where* f *is in some well-behaved hypothesis class and* ξ_x *is drawn i.i.d. from some distribution. The use of* $\tilde{f}$ *is not to model corruption by noise but rather the fact that data is not exactly manufactured by a function in the hypothesis class"* [36].

Feature selection A subset selection problem for regression is as follows: *"Given the (normalized) covariances between n variables* X_i *(which can in principle be observed) and a variable* Z *(which is to be predicted), select a subset of* $k(\ll n)$ *of the variables* X_i *and a linear prediction function of* Z *from the selected* X_i *that maximizes the* R^2 *fit"* [36]. With the strong condition "absence of conditional suppressors" proposed by [18], R^2 objective is submodular where R^2 is squared multiple correlation in this case. However, without the condition, R^2 objective is not always submodular but close to submodular.

For these reasons above, *an approximately submodular function* has been attracted an increasing attention recently [13, 18, 19, 23, 24, 33, 36, 39].

As an extension of the submodular function maximization problem, we consider an *approximately submodular function maximization problem* under a cardinality constraint. A variety of approximately submodular functions with different definitions have been considered recently. For example, in feature selection problem above, Das and Kempe [18, 19] consider the feature selection problem which is formulated as maximizing R^2 fit value. They present that R^2 fit behaves close to a submodular function with respect to selected features. Therefore, they formulate the R^2 fit as an approximately version of non-decreasing submodular function. To achieve the objective, they formulate the problem by using a submodular ratio γ which represents how much functions are close to submodular. Let $f(S)$ be a non-negative set function which returns the R^2 fit value of a selected subset $S \subseteq N$.

The *submodular ratio* (*submodularity ratio*) of f with respect to a set N and a parameter $k \geq 1$ is defined as

$$\gamma = \min_{S \subseteq N, |T| \leq k, T \cap S = \emptyset} \frac{\sum_{x \in T} f(S \cup \{x\}) - f(S)}{f(S \cup T) - f(S)}. \tag{1.7}$$

If the submodular ratio $\gamma \geq 1$ holds, then a function f is submodular [8, 18]. If γ is close to 0, a function f behaves as a general set function. Therefore, it is preferable to obtain a submodular ratio close to 1. For the feature selection problem which selects k features out of n features, Das and Kempe [18] presented two kinds of greedy algorithms, which are called Forward Selection algorithm and Orthogonal Matching Pursuit algorithm. Both algorithms attain $(1 - e^{-\gamma})$-approximation ratio for a given submodular ratio γ. The proof and details are in [18].

We note that to obtain the maximum submodular ratio is NP-hard in general according to [5]. Bai and Bilmes [5] also showed that submodular ratios can not be computed in polynomial time in the oracle model [19]. However, for some applications it is possible to obtain a lower bound of submodular ratio γ quickly. For example, in a special case of the feature selection problem, Das and Kempe [18] showed that a lower bound of submodular ratio can be obtained by computing the smallest eigenvalue of a covariance matrix which is positive semi-definite.

1.3 Research Objectives and Overview

(Research objectives) For "the submodular function" and "the approximately submodular function" problems under a cardinality constraint or others, many heuristic algorithms have been investigated and proposed. However, in feature selection, document summarization, or sensor placement problems, they often ask for the optimal solutions or solutions better than solutions obtained by heuristic algorithms. We present some main reasons that heuristic algorithms are not enough for the problems, below.

In feature selection problem [18, 19], the problem asks to select the essential features to represent a model with minimal loss of information. A variety of greedy

algorithms often fail to select essential features that can not be removed without affecting the original conditional target distribution when considering a strong relevant feature [85]. Due to the feature of a greedy algorithm, the algorithm often returns a set of features far from an optimal solution.

In sensor placement problem [51, 63], the problem asks to select a limited number of placed sensors for covering the maximum area. For the problem, we would like to obtain the optimal solution or better solutions than solutions obtained by a variety of greedy algorithms because we can spend the sufficient amount of computation time. The placement of the sensors becomes crucial because the sensors are often operated for a long time after they have been once installed. For companies or some societies, the loss of the misplacement for each year might impose a larger loss in the future.

In document summarization problem [32, 55, 57, 62], the problem asks to select a limited number of sentences to represent the original document as much as possible. For the problem, we would like to obtain the optimal set of sentences while considering the main three properties *relevance, redundancy,* and *length* of selected sentences at the same time [62]. Therefore, it is very difficult to solve the optimization problem because the simple heuristic algorithms often can not provide the sophisticated approaches while satisfying the three properties simultaneously.

In this thesis, according to the reasons above, we focus on developing exact algorithms for the problems under a cardinality constraint. We propose branch-and-cut algorithms for "the submodular function" and "the approximately submodular function" maximization problems in Chapters 2 and 3.

(Overview) First, Nemhauser and Wolsey [67] formulated the submodular function maximization problem with a cardinality constraint into a binary integer programming (BIP) problem as follows:

$$
\begin{aligned}
\text{maximize} \quad & z \\
\text{subject to} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S)\, y_i,\ S \in F, \\
& \sum_{i \in N} y_i \leq k, \\
& y_i \in \{0,1\},\ i \in N,
\end{aligned}
\tag{1.8}
$$

where F denotes the set of all feasible solutions satisfying the cardinality constraint $|S| \leq k$. We first point out the advantage of this BIP formulation which can apply for the submodular function maximization problem under any linear constraints with some modifications. Therefore, we do not need to develop a new algorithm under different kinds of constraints.

For solving the non-decreasing submodular function maximization problem exactly, it is often very time-consuming to enumerate all the constraints F. Since we often do not need all the constraints F for solving the problem exactly, Nemhauser and Wolsey [67] proposed a constraint generation algorithm that starts from a reduced BIP problem with a small subset of constraints taken from all the constraints F. Their algorithm repeats solving a reduced BIP problem while adding a new constraint at each iteration. Their algorithm is not often efficient for solving the problem exactly since their algorithm needs to solve too many reduced BIP problems. In Chapter 2 of the thesis, to overcome the issue, we propose an improved constraint generation algorithm that adds a promising set of constraints at each iteration. To generate the set of constraints, we use an idea of column generation [14]. We incorporate the improved constraint generation algorithm into a branch-and-cut algorithm for improving the efficiency. For improving the incumbent solution (the best solution obtained so far), we introduce a simple local search algorithm, and we incorporate it into the branch-and-cut algorithm.

Next, for an extension of the submodular function maximization problem, we have considered an approximately submodular function maximization problem. This type of function is defined with *a submodular ratio* γ, which defined for a set function f as the maximum value $0 < \gamma < 1$ such that $f(S \cup \{i\}) - f(S) \geq \gamma\ (f(T \cup \{i\}) - f(T))$, for all $S \subseteq T \subseteq N$ and $i \notin T$. That is, a submodular ratio

γ measures how close the function is to submodular [18, 43]. We formulated the approximately submodular function with different forms by using a submodular ratio γ and an upper bound of submodular ratio $\bar{\gamma}$ proposed by Johnson et al. [43]. For the problem, to the best of my knowledge, no BIP formulation of its problem has been proposed. Therefore, the idea of the constraint generation algorithm [67] is not able to be applied. In Chapter 3, to overcome the issue, we first define an approximately submodular function and formulate the problem into a BIP problem with an exponential number of constraints with a given submodular ratio γ. Due to the success of formulating the problem into the BIP problem, we develop three exact algorithms including a branch-and-cut algorithm based on the BIP formulation with the similar manner for the previous problem.

The remainder of this thesis is organized as follows. In Chapter 2, we first review two existing exact algorithms, A* search algorithms [13, 70]. Next, we review an existing exact algorithm called a constraint generation algorithm proposed by Nemhauser and Wolsey [67]. We improve the constraint generation algorithm and propose three exact algorithms including a branch-and-cut algorithm. We also show some computational results using three types of well-known benchmark instances with three existing algorithms and three proposed algorithms. Then, in Chapter 3, we define an approximately submodular function by using two different submodular ratios. Based on the definition, we derive a BIP formulation of the problem. Then, we propose three exact algorithms for the problem based on the BIP formulation. Finally, we illustrate the effectivity of the proposed algorithm with the combinatorial auction problem, and show some computational results with the three types of well-known benchmark instances. Finally, we summarize our research of this thesis in Chapter 4.

Chapter 2

An Efficient Branch-and-Cut Algorithm for Submodular Function Maximization

2.1 Introduction

In this chapter, we address the problem of maximizing a submodular function with a cardinality constraint, formulated as

$$\begin{aligned} &\text{maximize} && f(S) \\ &\text{subject to} && |S| \leq k,\ S \subseteq N, \end{aligned} \tag{2.1}$$

where $N = \{1, \ldots, n\}$, n is the size of the finite set and $k \leq n$ is a positive integer comprising the cardinality constraint. This chapter focuses on non-decreasing submodular functions. This problem is well known as NP-hard, unlike submodular function minimization [54].

Recently, Chen et al. [13] proposed an A* search algorithm to obtain an optimal solution of the non-decreasing submodular function maximization problem. Their algorithm computes an upper bound by a variant of variable fixing techniques with O(n) oracle queries. Sakaue and Ishihata [70] improved the A* search algorithm to obtain better upper bounds for the non-decreasing submodular function max-

imization with a knapsack constraint. Their algorithms quickly find good upper bounds; however, the attained upper bounds are not often tight enough to prune nodes of the search tree effectively. Therefore, their algorithms often process many nodes of the search tree until obtaining an optimal solution.

In this chapter, we propose an efficient branch-and-cut algorithm for the non-decreasing submodular function maximization problem based on the following binary integer programming (BIP) formulation [67]:

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S)\, y_i,\ S \in F, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\},\ i \in N, \end{aligned} \tag{2.2}$$

where $f(T \mid S) = f(S \cup T) - f(S)$ for all $S, T \subseteq N$ and F denotes the set of all feasible solutions satisfying the cardinality constraint $|S| \leq k$.

Nemhauser and Wolsey [67] previously proposed an exact algorithm called the constraint generation algorithm based on the BIP formulation (2.2). The size of the BIP formulation grows exponentially compared to n since it has more than $\binom{n}{k}$ constraints. To overcome this, they proposed the constraint generation algorithm that starts from a reduced BIP problem with a small subset of constraints taken from the constraints. Their algorithm repeats solving a reduced BIP problem while adding a new constraint at each iteration. Unfortunately, this is not efficient in practice because their algorithm requires to solve many reduced BIP problems. They also proposed a branch-and-cut algorithm with solving linear programming (LP) relaxation problems of the reduced BIP problems to obtain upper bounds. Their branch-and-cut algorithm however can not prune nodes of the search tree efficiently because the LP relaxation problems often give much worse upper bounds than the reduced BIP problems.

Kawahara et al. [46] proposed an exact algorithm for updating a lower bound based on Tuy's cutting-plane method [76]. The algorithm reformulates the sub-

modular function maximization problem (2.1) by using the Lovász extension [61]. It iteratively finds a feasible solution and a unique hyperplane to cut off a feasible subset which clearly does not include any better solutions. To obtain an upper bound, they introduced the constraint generation algorithm [67] adding the obtained feasible solution as a constraint.

Both of the exact algorithms [46, 67] often need to solve a large number of reduced BIP problems because of generating only one or two constraints at each iteration. To overcome this, we propose an improved constraint generation algorithm to add a promising set of constraints at each iteration. We incorporate it into a branch-and-cut algorithm to attain good upper bounds while solving a smaller number of reduced BIP problems. To improve the efficiency of the branch-and-cut algorithm, we also introduce a local search algorithm to attain good lower bounds quickly. We evaluate the existing algorithms and our algorithms for three types of well-known benchmark instances called facility location, weighted coverage and bipartite influence. According to the performance profile [21] and the shifted geometric mean [66] of the computation time, we confirm that our algorithms improved the efficiency of the conventional constraint generation algorithm [68], and also performed better than the existing algorithms [13, 70].

2.2 Existing Algorithms

We review the A* search algorithms proposed by Chen et al. [13] and Sakaue and Ishihata [70], and the constraint generation algorithm proposed by Nemhauser and Wolsey [67] for the non-decreasing submodular function maximization problem. First, for A* search algorithms, we then illustrate two heuristic functions h_{mod} [13] and h_{dom} [70].

2.2.1 A* search Algorithms

We first define the search tree of the A* search algorithm. Each node S of the search tree represents a feasible solution, where the root node is set to $S \leftarrow \emptyset$. The parent of a node T is defined as $S = T \setminus \{T_{\max}\}$, where $T_{\max}$ is an element $i \in T$ with the largest number. For example, node $S = \{3\}$ is the parent of node $T = \{3, 5\}$, since $T \setminus \{T_{\max}\} = \{3, 5\} \setminus \{5\} = \{3\} = S$.

The A* search algorithm employs a list L to manage nodes of the search tree. The value of a node S is defined as $\bar{f}(S) = f(S) + h(S)$, where $h(\cdot)$ is a heuristic function. We note that $\bar{f}(\cdot)$ gives an upper bound of the optimal value of the problem (2.1) at the node S.

The initial feasible solution is obtained by the greedy algorithm [64, 68]. The algorithm repeats to extract a node S with the largest value $\bar{f}(\cdot)$ from the list L and insert its children $T \in F$ into the list L at each iteration. Let $S \in F$ be a node extracted from the list L, and S^* be the incumbent solution (i.e., the best feasible solution obtained so far). The algorithm may apply the greedy algorithm to the node S for obtaining a feasible solution $S' \in F$. If $f(S') > f(S^*)$ holds, then the algorithm replaces the incumbent solution S^* with S'. Then, all children $T \in F$ of the node S satisfying $\bar{f}(T) > f(S^*)$ are inserted into the list L. The algorithm repeats these procedures until the list L becomes empty.

Algorithm A$^*(S)$

Input: The initial feasible solution S.

Output: The incumbent solution S^*.

Step 1: Set $L \leftarrow \{\emptyset\}$ and $S^* \leftarrow S$.

Step 2: If $L = \emptyset$ holds, then output the incumbent solution S^* and exit.

Step 3: Extract a node S with the largest value $\bar{f}(\cdot)$ from the list L. If $\bar{f}(S) \leq f(S^*)$ holds, then return to Step 2.

Step 4: Obtain a feasible solution $S' \in F$ from the node S. If $f(S') > f(S^*)$ holds, then set $S^* \leftarrow S'$.

Step 5: Set $L \leftarrow L \cup \{T\}$ for all children T of the node S satisfying $T \in F$ and $\bar{f}(T) > f(S^*)$. Return to Step 2.

We then illustrate two heuristic functions h_{mod} [13] and h_{dom} [70] applied to the A* search algorithm.

2.2.1.1 Upper bound with modular functions (MOD)

Chen et al. [13] proposed a heuristic function h_{mod}. Let S be the current node of the A* search algorithm. We consider the following reduced problem of the problem (2.1) for obtaining $h(\cdot)$.

$$\begin{aligned} &\text{maximize} && f_S(T) \\ &\text{subject to} && T \subseteq N \setminus S^+, |T| \leq k - |S|, \end{aligned} \tag{2.3}$$

where $S^+ = \{i \in N \mid i \leq S_{\max}\}$ and $f_S(\cdot) = f(\cdot \mid S)$. Let T^* be an optimal solution of the reduced problem (2.3). Since the reduced problem (2.3) is still NP-hard, we consider obtaining an upper bound of $f_S(T^*)$. By submodularity, we obtain $\sum_{i \in T} f_S(\{i\}) \geq f_S(T)$ for any $T \subseteq N$ and the following inequality [69].

$$\max_{T \subseteq N \setminus S^+, |T| \leq k-|S|} \sum_{i \in T} f_S(\{i\}) \geq \sum_{i \in T^*} f_S(\{i\}) \geq f_S(T^*). \tag{2.4}$$

Let $\bar{S}^+$ be the non-increasing ordered set with respect to $f_S(\{i\})$ for $i \in N \setminus S^+$. We assume that $|S \cup \bar{S}^+| > k$, because we can obtain the upper bound by computing $f(S \cup \bar{S}^+)$ otherwise. Let $[p] = \{1, \ldots, p\}$ and $\bar{S}^+_{[p]}$ denote the set of the first $p = k - |S|$ elements of the sorted set $\bar{S}^+$. We then define a heuristic function h_{mod} by

$$h_{mod}(S) = \sum_{i \in \bar{S}^+_{[p]}} f_S(\{i\}). \tag{2.5}$$

We note that we let $\bar{S}^+_{[p]} \cup S$ be a feasible solution $S' \in F$ for the node S. If $f_S(\{i\}) = 0$ holds for some $i \in \bar{S}^+_{[p]}$, then we conclude $f_S(\bar{S}^+_{[p]}) = f_S(T^*)$ by submodularity. For a given node S, we compute an upper bound $\bar{f}(S) = f(S) + h_{mod}(S)$.

When we have different constraints such as multiple knapsack constraints, this algorithm is still adaptable by using LP solvers. We solve the following linear programming (LP) problem with LP solvers, and obtain upper bounds.

$$\begin{array}{ll} \text{maximize} & \displaystyle\sum_{i \in T} f_S(\{i\}) \\ \text{subject to} & T \subseteq F,\ T \subseteq N \setminus S. \end{array} \tag{2.6}$$

2.2.1.2 Upper bound with dominant elements (DOM)

Sakaue and Ishihata [70] proposed another heuristic function h_{dom}. We define T as the ordered set of $p = k - |S|$ elements added to the current solution S by the greedy algorithm. Let T_i and $T_{[i]}$ denote the i-th element and the subset of the first i elements of the sorted set T, respectively. We define

$$\beta_{[p]} = \begin{cases} 0 & \text{if } h_{mod}(S \cup T_{[i]}) = 0 \text{ for some } i \in [p] \\ \displaystyle\prod_{i=1}^{p} \beta_i & \text{otherwise,} \end{cases} \tag{2.7}$$

where

$$\beta_i = 1 - \frac{f_S(T_i \cup T_{[i-1]}) - f_S(T_{[i-1]})}{h_{mod}(S \cup T_{[i-1]})}, \quad i \in [p]. \tag{2.8}$$

We then define a heuristic function h_{dom} by

$$h_{dom}(S) = \frac{f_S(T_{[p]})}{(1 - \beta_{[p]})}. \tag{2.9}$$

Theorem 2.1. *The heuristic function holds $h_{dom}(S) \geq f_S(T^*)$ for any given $S \subseteq N$ [70].*

Proof. For $i = 0, \ldots, k$, we obtain the following inequality

$$\begin{array}{rcl} f_S(T^*) & \leq & f_S(T^* \cup T_{[i]}) \\ & = & f_S(T_{[i]}) + f_S(T^* \mid T_{[i]}) \\ & \leq & f_S(T_{[i]}) + f_S(T^* \mid T_{[i]}) \\ & \leq & f_S(T_{[i]}) + h_{mod}(S \cup T_{[i]}), \end{array} \tag{2.10}$$

where the third inequality comes from the definition of h_{mod}.

We first consider the case where $h_{mod}(S \cup T_{[i]}) = 0$ holds for some $i \in [p]$. In this case we obtain

$$f_S(T^*) \leq f_S(T_{[i]}) + h_{mod}(S \cup T_{[i]}) = f_S(T_{[i]}) \tag{2.11}$$

from the inequality (2.10). By the non-decreasingness of f_S, we have $f_S(T_{[p]}) \geq f_S(T_{[i]})$, and $h_{dom} = f_S(T_{[p]}) \geq f_S(T^*)$ holds; more precisely, $h_{dom} = f_S(T_{[p]}) = f_S(T^*)$ holds since T^* is optimal.

We then consider the case where $h_{dom}(T_{[p]}) > 0$ holds for all $i \in [p]$. For $i \in [p]$, we have

$$\begin{aligned} f_S(T^*) &\leq f_S(T_{[i-1]}) + h_{mod}(T_{[i-1]}) \\ &= f_S(T_{[i-1]}) + \frac{h_{mod}(T_{[i-1]})}{f_S(T_i \mid T_{[i-1]})}(f_S(T_{[i]}) - f_S(T_{[i-1]})) \\ &= f_S(T_{[i-1]}) + \frac{h_{mod}(T_{[i-1]})}{1-\beta_i}(f_S(T_{[i]}) - f_S(T_{[i-1]})) \end{aligned} \tag{2.12}$$

from the inequality (2.10). We rearrange the terms to obtain the following inequality,

$$f_S(T_{[i]}) \leq \beta_i f_S(T_{[i-1]}) + (1-\beta_i) f_S(T^*). \tag{2.13}$$

Therefore, we obtain the following inequality for $i \in [p]$:

$$f_S(T^*) - f_S(T_{[i]}) \leq \beta_i (f_S(T^*) - f_S(T_{[i-1]})). \tag{2.14}$$

Since $f_S(\emptyset) = 0$, we obtain

$$\begin{aligned} f_S(T^*) &\geq \left(1 - \prod_{i=1}^{p} \beta_i\right) f_S(T^*) \\ &= (1 - \beta_{[p]})\, f_S(T^*). \end{aligned} \tag{2.15}$$

Therefore, the statement of the theorem $h_{dom}(S) = f_S(T_{[p]})/(1-\beta_{[p]}) \geq f_S(T^*)$ holds. □

For a given node S, we compute an upper bound $\bar{f}(S) = f(S) + h_{dom}(S)$. With the cardinality constraint, we have $\beta_i = 1 - 1/k$ in the worst case. That leads to the $(1-1/e)$-approximation guarantee. Since $\beta_i < 1 - 1/k$, and h_{dom} is expected

to be closer to $f_S(T^*)$ than $f_S(T_{[k]})/(1-1/e)$. Sakaue and Ishihata [70] discussed situations where h_{dom} gives an upper-bound that is close to $f_S(T^*)$ (see details in [70]).

2.2.2 Constraint Generation Algorithm

Nemhauser and Wolsey [67] proposed a constraint generation algorithm based on a BIP formulation. They reformulated the submodular maximization problem (2.1) into a BIP problem. For reformulating the problem, we first need the definition of submodular function as follows.

Proposition 9. *Each of the following statements is equivalent and defines a submodular function* [68].

(i) $f(A) - f(A \cap B) \geq f(A \cup B) - f(B), \ \forall\, A, B \subseteq N.$

(ii) $f(\{i\} \mid S) \geq f(\{i\} \mid T), \ \forall\, S \subseteq T \subseteq N,\, i \notin T.$

(iii) $f(\{i\} \mid S) \geq f(\{i\} \mid S \cup \{e\}), \ \forall\, S \subseteq N, i \notin S \cup \{e\}.$

(iv) $f(T) \leq f(S) + \sum_{j \in T \setminus S} f(\{j\} \mid S) - \sum_{i \in S \setminus T} f(\{i\} \mid T \cup S \setminus \{i\}), \ \forall\, S, T \subseteq N.$

(v) $f(T) \leq f(S) + \sum_{j \in T \setminus S} f(\{j\} \mid S), \quad \forall\, S \subseteq T \subseteq N.$

(vi) $f(T) \leq f(S) - \sum_{i \in S \setminus T} f(\{i\} \mid S \setminus \{i\}), \ \forall\, T \subseteq S \subseteq N.$

(vii) $f(T) \leq f(S) - \sum_{i \in S \setminus T} f(\{i\} \mid S \setminus \{i\}) + \sum_{i \in T \setminus S} f(\{i\} \mid S \cap T), \ \forall\, S, T \subseteq N.$

The proof of Proposition 9 is omitted (see details in [68]). We next consider a submodular function which is non-decreasing.

Proposition 10. *Each of the following statements is equivalent and defines a non-decreasing submodular set function* [68].

($\mathbf{i}^*$) $f(A) + f(B) \geq f(A \cup B) + f(A \cap B),\ \forall A, B \subseteq N.$

($\mathbf{ii}^*$) $f(\{i\} \mid S) \geq f(\{i\} \mid T) \geq 0,\ \forall S \subseteq T \subseteq N,\ i \notin T.$

($\mathbf{iv}^*$) $f(T) \leq f(S) + \sum_{j \in T \setminus S} f(\{j\} \mid S), \quad \forall S, T \subseteq N.$

The proof of Proposition 10 is as follows [68].

Proof. First, we obtain (i*) $\Leftrightarrow$ (ii*) clearly from (i) $\Leftrightarrow$ (ii). Next, (ii*) $\Rightarrow$ (ii) $\Rightarrow$ (iv). Since $f(\{i\} \mid T) \geq 0$ implies that the last term of (iv) is non-positive, and the fact gives (iv*). Finally (iv*) $\Rightarrow$ (ii*) when we choose $S = T \cup \{i\}$ in (iv*) yields $f(T) \leq f(T \cup \{i\})$ or $f(\{i\} \mid T) \geq 0$. □

We next consider a set X of $(\eta, \boldsymbol{y})$ satisfying the following condition.

$$\eta \leq f(S) + \sum_{j \in N \setminus S} f(\{j\} \mid S)\, y_j,\ \forall S \subseteq N,\ |S| \leq k, \tag{2.16}$$

where $\boldsymbol{y} = (y_1, \ldots, y_n) \in \{0, 1\}^n$.

Proposition 11. *Suppose $f(\cdot)$ is a non-decreasing submodular function, $(\eta, \boldsymbol{y}^{\boldsymbol{T}})$ $\in X$ if and only if $\eta \leq f(T)$, $T = \{j \in N \mid y_j = 1\} \subseteq N$.*

The following proof is for Proposition 11 [67].

Proof. ($\Leftarrow$). Suppose $\eta \leq f(T)$, then for all $S \subseteq N$, we obtain the following inequality.

$$\begin{aligned} & f(S) + \sum_{j \in N \setminus S} f(\{j\} \mid S) y_j^T \\ = \ & f(S) + \sum_{j \in T \setminus S} f(\{j\} \mid S) \\ \geq \ & f(T) \geq \eta, \end{aligned} \tag{2.17}$$

where the inequality comes from the statement (iv*) of Proposition 10.

($\Rightarrow$). If $(\eta,\, \boldsymbol{y^T}) \in X$, we obtain the following inequality,

$$\begin{aligned} \eta &\leq f(T) + \sum_{j \in N \setminus T} f(\{j\} \mid T)\, y_j^T \\ &= f(T). \end{aligned} \tag{2.18}$$

□

The submodular function maximization problem can be formulated into the following BIP problem (2.19).

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S) y_i, \quad S \in F, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\},\ i \in N, \end{aligned} \tag{2.19}$$

where F denotes the set of all feasible solutions satisfying the cardinality constraint $|S| \leq k$. According to [67], we only need feasible solutions such that $|S| = k$ for a set F. By solving the problem (2.19) with all the constraints, an optimal solution and the optimal value can be obtained. However, it is often very expensive to prepare all the constraints within a reasonable computation time. Moreover, to solve the problem (2.19) optimally, we do not need all the constraints in most cases.

Nemhauser and Wolsey [67] have proposed an exact algorithm called the constraint generation algorithm starting from a reduced BIP problem with a small subset of constraints. The algorithm repeats solving a reduced BIP problem while adding a new constraint at each iteration. Given a set of feasible solutions $Q \subseteq F$, we define BIP(Q) as the following reduced BIP problem of the problem (2.19).

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S)\, y_i,\ S \in Q, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\},\ i \in N. \end{aligned} \tag{2.20}$$

The initial solution $S^{(0)}$ is obtained by the greedy algorithm [64, 68]. The constraint generation algorithm starts with a set $Q = \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}\}$, where $S_{[u]}$ denotes the first u elements of a feasible solution $S^{(0)}$ with the order obtained by the greedy algorithm. We now consider the t-th iteration of the constraint generation algorithm. The algorithm first solves BIP(Q) with $Q = \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k-1]}, S^{(0)}, \ldots, S^{(t-1)}\}$ to obtain an optimal solution $\boldsymbol{y}^{(t)} = (y^{(t)}_1, \ldots, y^{(t)}_n)$ and the optimal value $z^{(t)}$ that gives an upper bound of that of the problem (2.19). Let $S^{(t)}$ denote the optimal solution of BIP(Q) corresponding to $\boldsymbol{y}^{(t)}$, and S^* denote the incumbent solution of the problem (2.19) (i.e., the best feasible solution obtained so far). If $f(S^{(t)}) > f(S^*)$ holds, then the algorithm replaces the incumbent solution S^* with $S^{(t)}$. If $z^{(t)} > f(S^{(t)})$ holds, the algorithm concludes $S^{(t)} \notin Q$ and adds $S^{(t)}$ to Q, because $S^{(t)}$ does not satisfy any constraints of BIP(Q). That is, the algorithm adds the following constraint to BIP(Q) for improving the upper bound $z^{(t)}$ of the optimal value of the problem (2.19).

$$z \leq f(S^{(t)}) + \sum_{i \in N \setminus S^{(t)}} f(\{i\} \mid S^{(t)})\, y_i. \tag{2.21}$$

The algorithm repeats these procedures until $z^{(t)}$ and $f(S^*)$ meet (i.e., the algorithm proves the optimality of the incumbent solution S^*). We note that the value of $z^{(t)}$ is non-increasing with the number of iterations and the algorithm must terminate after at most $\binom{n}{k}$ iterations.

Algorithm CG$(S^{(0)})$

Input: The initial feasible solution $S^{(0)}$.

Output: The incumbent solution S^*.

Step 1: Set $Q \leftarrow \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}\}$, $S^* \leftarrow S^{(0)}$ and $t \leftarrow 1$.

Step 2: Solve BIP(Q). Let $S^{(t)}$ and $z^{(t)}$ be an optimal solution and the optimal value of BIP(Q), respectively.

Step 3: If $f(S^{(t)}) > f(S^*)$ holds, then set $S^* \leftarrow S^{(t)}$.

Step 4: If $z^{(t)} = f(S^*)$ holds, then output the incumbent solution S^* and exit. Otherwise; (i.e., $z^{(t)} > f(S^*) \geq f(S^{(t)})$), set $Q \leftarrow Q \cup \{S^{(t)}\}$, $t \leftarrow t + 1$ and return to Step 2.

The constraint generation algorithm [67] often solves a large number of reduced BIP problems because of generating only one constraint at each iteration. We accordingly propose an improved constraint generation algorithm to generate a promising set of constraints for attaining good upper bounds while solving a smaller number of reduced BIP problems.

We note that to generate and add a set of random constraints does not deteriorate upper bounds. However, the efficiency of algorithms often deteriorates because of generating and adding an unnecessary set of constraints to solve a problem. Therefore, we carefully need to generate a set of constraints to improve the upper bound.

2.3 Proposed Algorithms

2.3.1 Improved Constraint Generation Algorithm

Let $\boldsymbol{y}^{(t)} = (y^{(t)}_1, \ldots, y^{(t)}_n)$ and $z^{(t)}$ be an optimal solution and the optimal value of BIP(Q) at the t-th iteration of the constraint generation algorithm, respectively.

We note that $z^{(t)}$ gives an upper bound of the optimal value of the problem (2.19). To improve the upper bound $z^{(t)}$, it is necessary to add a new feasible solution $S' \in F$ to Q satisfying the following inequality.

$$z^{(t)} > f(S') + \sum_{i \in N \setminus S'} f(\{i\} \mid S')\, y_i^{(t)}. \tag{2.22}$$

After solving BIP(Q), we obtain at least one feasible solution $S^\natural \in Q$ attaining the optimal value $z^{(t)}$ of BIP(Q), i.e.,

$$z^{(t)} = f(S^\natural) + \sum_{i \in N \setminus S^\natural} f(\{i\} \mid S^\natural)\, y_i^{(t)}. \tag{2.23}$$

Let $S^{(t)}$ be the optimal solution of BIP(Q) corresponding to $\boldsymbol{y}^{(t)}$, where we assume $S^{(t)} \notin Q$. We then consider adding an element $j \in S^{(t)} \setminus S^\natural$ to $S^\natural$, and obtain the following inequality by submodularity:

$$\begin{aligned} z^{(t)} &= f(S^\natural) + \sum_{i \in N \setminus S^\natural} f(\{i\} \mid S^\natural)\, y_i^{(t)} \\ &= f(S^\natural) + f(\{j\} \mid S^\natural)\, y_j^{(t)} + \sum_{i \in N \setminus (S^\natural \cup \{j\})} f(\{i\} \mid S^\natural)\, y_i^{(t)} \\ &= f(S^\natural \cup \{j\}) + \sum_{i \in N \setminus (S^\natural \cup \{j\})} f(\{i\} \mid S^\natural)\, y_i^{(t)} \\ &\geq f(S^\natural \cup \{j\}) + \sum_{i \in N \setminus (S^\natural \cup \{j\})} f(\{i\} \mid S^\natural \cup \{j\})\, y_i^{(t)}, \end{aligned} \tag{2.24}$$

where $y_j^{(t)} = 1$ due to $j \in S^{(t)}$. From the inequality (2.24), we observe that it is preferable to add the element $j \in S^{(t)} \setminus S^\natural$ to $S^\natural$ for improving the upper bound $z^{(t)}$. Here, we note that it is necessary to remove another element $i \in S^\natural$ if $|S^\natural| = k$ holds.

Based on this observation, we develop a heuristic algorithm to generate a set of new feasible solutions $S' \in F$ for improving the upper bound $z^{(t)}$. Given a set of feasible solutions $Q \subseteq F$, let q_i be the number of feasible solutions $S \in Q$ including an element $i \in N$. We define the occurrence rate p_i of each element i with respect to Q as

$$p_i = \frac{q_i}{\sum_{j \in N} q_j}. \tag{2.25}$$

For each element $i \in S^{\natural} \cup S^{(t)}$, we set a random value r_i satisfying $0 \leq r_i \leq p_i$. If there are multiple feasible solutions $S^{\natural} \in Q$ satisfying the equation (2.23), then we select one of them at random. We take the k largest elements $i \in S^{\natural} \cup S^{(t)}$ with respect to the value r_i to generate a feasible solution $S' \in F$.

Algorithm SUB-ICG$(Q, S^{(t)}, \lambda)$

Input: A set of feasible solutions $Q \subseteq F$. A feasible solution $S^{(t)} \notin Q$. The number of feasible solutions to be generated λ.

Output: A set of feasible solutions $Q' \subseteq F$.

Step 1: Set $Q' \leftarrow \emptyset$ and $h \leftarrow 1$.

Step 2: Select a feasible solution $S^{\natural} \in Q$ satisfying the equation (2.23) at random. Set a random value r_i $(0 \leq r_i \leq p_i)$ for $i \in S^{\natural} \cup S^{(t)}$.

Step 3: If $|S^{\natural}| = k$ holds, then take the k largest elements $i \in S^{\natural} \cup S^{(t)}$ with respect to r_i to generate a feasible solution $S' \in F$. Otherwise, take the largest element $i \in S^{(t)} \setminus S^{\natural}$ with respect to r_i to generate a feasible solution $S' = S^{\natural} \cup \{i\} \in F$.

Step 4: If $S' \notin Q'$ holds, then set $Q' \leftarrow Q' \cup \{S'\}$ and $h \leftarrow h + 1$.

Step 5: If $h = \lambda$ holds, then output Q' and exit. Otherwise, return to Step 2.

We summarize the improved constraint generation algorithm as follows, in which we define Q as the set of feasible solutions $S^{(0)}, S^{(1)}, \ldots, S^{(t-1)}$ obtained by solving reduced BIP problems and Q^+ as the set of feasible solutions generated by SUB-ICG$(Q, S^{(t)}, \lambda)$.

Algorithm ICG$(S^{(0)}, \lambda)$

Input: The initial feasible solution $S^{(0)}$. The number of feasible solutions to be generated at each iteration λ.

Output: The incumbent solution S^*.

Step 1: Set $Q \leftarrow \{S^{(0)}\}$, $Q^+ \leftarrow \{S^{(0)}_{[0]}, \ldots, S^{(0)}_{[k]}\}$, $S^* \leftarrow S^{(0)}$ and $t \leftarrow 1$.

Step 2: Solve BIP(Q^+). Let $S^{(t)}$ and $z^{(t)}$ be an optimal solution and the optimal value of BIP(Q^+), respectively.

Step 3: If $f(S^{(t)}) > f(S^*)$ holds, then set $S^* \leftarrow S^{(t)}$.

Step 4: If $z^{(t)} = f(S^*)$ holds, then output the incumbent solution S^* and exit.

Step 5: Set $Q \leftarrow Q \cup \{S^{(t)}\}$, $Q^+ \leftarrow Q^+ \cup \{S^{(t)}\} \cup$ SUB-ICG$(Q, S^{(t)}, \lambda)$ and $t \leftarrow t + 1$.

Step 6: For each feasible solution $S \in$ SUB-ICG$(Q, S^{(t)}, \lambda)$, if $f(S) > f(S^*)$ holds, then set $S^* \leftarrow S$. Return to Step 2.

We note that the improved constraint generation algorithm often attains good lower bounds as well as the upper bounds because SUB-ICG gives a number of good feasible solutions at each iteration. There are other ways to generate constraints such as a randomized greedy algorithm at the beginning. However, it is often difficult to generate necessary constraints during the algorithm. Fortunately, this algorithm dynamically finds the constraints which are necessary at each iteration by using a solution $S^{(t)}$ and a constraint $S^{\natural}$.

2.3.2 Branch-and-Cut Algorithm

We propose a branch-and-cut algorithm incorporating the improved constraint generation algorithm. We first define the search tree of the branch-and-cut algorithm. Each node (S^0, S^1) of the search tree consists of a pair of sets S^0 and

S^1, where elements $i \in S^0$ (resp., $i \in S^1$) correspond to variables fixed to $y_i = 0$ (resp., $y_i = 1$) of the problem (2.19). The root node is set to $(S^0, S^1) \leftarrow (\emptyset, \emptyset)$. Each node (S^0, S^1) has two children $(S^0 \cup \{i^*\}, S^1)$ and $(S^0, S^1 \cup \{i^*\})$, where $i^* = \mathrm{argmax}_{i \in N \setminus (S^0 \cup S^1)} f(S^1 \cup \{i\})$.

The branch-and-cut algorithm employs a stack list L to manage nodes of the search tree. The value of a node (S^0, S^1) is defined as the optimal value $z^{(S^0,S^1)}$ of the following reduced BIP problem BIP(Q^+, S^0, S^1):

$$
\begin{array}{lll}
\text{maximize} & z & \\
\text{subject to} & z \leq f(S) + \displaystyle\sum_{i \in N \setminus S} f(\{i\} \mid S)\, y_i, \ S \in Q^+, & \\
& \displaystyle\sum_{i \in N \setminus (S^0 \cup S^1)} y_i \leq k - |S^1|, & \\
& y_i \in \{0, 1\}, & i \in N \setminus (S^0 \cup S^1), \\
& y_i = 0, & i \in S^0, \\
& y_i = 1, & i \in S^1,
\end{array}
\tag{2.26}
$$

where Q^+ is the set of feasible solution generated by the improved constraint generation algorithm so far. We note that $z^{(S^0,S^1)}$ gives an upper bound of the optimal value of the problem (2.19) at the node (S^0, S^1); i.e., under the condition that $y_i = 0$ $(i \in S^0)$ and $y_i = 1$ $(i \in S^1)$.

We start with a pair of sets $Q = \{S\}$ and $Q^+ = \{S_{[0]}, \ldots, S_{[k]}\}$, where S is the initial feasible solutions obtained by the greedy algorithm [64, 68]. To obtain good upper and lower bounds quickly, we first apply the first k iterations of the improved constraint generation algorithm. We then repeat to extract a node (S^0, S^1) from the top of the stack list L and insert its children into the top of the stack list L at each iteration. Thus, we employ a depth-first-search for the tree search of the branch-and-cut algorithm.

Let (S^0, S^1) be a node extracted from the stack list L, and S^* be the incumbent solution of the problem (2.19) (i.e., the best feasible solution obtained so far). We first solve BIP(Q^+, S^0, S^1) to obtain an optimal solution $S^{(S^0,S^1)}$ and the optimal value $z^{(S^0,S^1)}$. We then generate a set of feasible solutions by

SUB-ICG$(Q, S^{(S^0,S^1)}, \lambda)$. For each feasible solution $S' \in \{S^{(S^0,S^1)}\} \cup$ SUB-ICG $(Q, S^{(S^0,S^1)}, \lambda)$, if $f(S') > f(S^*)$ holds, then we replace the incumbent solution S^* with S'. If $z^{(S^0,S^1)} > f(S^*)$ holds, then we insert the two children $(S^0 \cup \{i^*\}, S^1)$ and $(S^0, S^1 \cup \{i^*\})$ into the top of the stack list L in this order.

To decrease the number of reduced BIP problems to be solved in the branch-and-cut algorithm, we keep the optimal value $z^{(S^0,S^1)}$ of BIP(Q^+, S^0, S^1) as an upper bound $\bar{z}^{(S^0 \cup \{i^*\},S^1)}$ (resp., $\bar{z}^{(S^0,S^1 \cup \{i^*\})}$) of the child $(S^0 \cup \{i^*\}, S^1)$ (resp., $(S^0, S^1 \cup \{i^*\})$) when inserted to the stack list L. If $\bar{z}^{(S^0,S^1)} \leq f(S^*)$ holds when we extract a node (S^0, S^1) from the stack list L, then we can prune the node (S^0, S^1) without solving BIP(Q^+, S^0, S^1). We set the upper bound $\bar{z}^{(\emptyset,\emptyset)}$ of the root node $(\emptyset, \emptyset)$ to ∞. We repeat these procedures until the stack list L becomes empty.

Algorithm BC-ICG(S, λ)

Input: The initial feasible solution S. The number of feasible solutions to be generated at each node λ.

Output: The incumbent solution S^*.

Step 1: Set $L \leftarrow \{(\emptyset, \emptyset)\}$, $\bar{z}^{(\emptyset,\emptyset)} \leftarrow \infty$, $Q \leftarrow \{S\}$, $Q^+ \leftarrow \{S_{[0]}, \ldots, S_{[k]}\}$ and $S^* \leftarrow S$.

Step 2: Apply the first k iterations of ICG(S, λ) to update the sets Q and Q^+ and the incumbent solution S^*.

Step 3: If $L = \emptyset$ holds, then output the incumbent solution S^* and exit.

Step 4: Extract a node (S^0, S^1) from the top of the stack list L. If $\bar{z}^{(S^0,S^1)} \leq f(S^*)$ holds, then return to Step 3.

Step 5: Solve BIP(Q^+, S^0, S^1). Let $S^{(S^0,S^1)}$ and $z^{(S^0,S^1)}$ be an optimal solution and the optimal value of BIP(Q^+, S^0, S^1), respectively.

Step 6: Set $Q \leftarrow Q \cup \{S^{(S^0,S^1)}\}$, $Q^+ \leftarrow Q^+ \cup \{S^{(S^0,S^1)}\} \cup$SUB-ICG$(Q, S^{(S^0,S^1)}, \lambda)$.

Step 7: For each feasible solution $S' \in \{S^{(S^0,S^1)}\} \cup$ SUB-ICG$(Q, S^{(S^0,S^1)}, \lambda)$, if $f(S') > f(S^*)$ holds, then set $S^* \leftarrow S$.

Step 8: If $z^{(S^0,S^1)} \leq f(S^*)$, then return to Step 3.

Step 9: If $|S^0 \cup S^1| \leq n-1$ and $|S^1| \leq k-1$ hold, then set $L \leftarrow L \cup \{(S^0 \cup \{i^*\}, S^1), (S^0, S^1 \cup \{i^*\})\}$, $\bar{z}^{(S^0 \cup \{i^*\}, S^1)} \leftarrow z^{(S^0,S^1)}$ and $\bar{z}^{(S^0,S^1 \cup \{i^*\})} \leftarrow z^{(S^0,S^1)}$, where $i^* = \mathrm{argmax}_{i \in N \setminus (S^0 \cup S^1)} f(S^1 \cup \{i\})$. Return to Step 3.

We note that the branch-and-cut algorithm is similar to that for the traveling salesman problem based on a BIP formulation with an exponential number of subtour elimination constraints [17, 34].

We finally propose an improved branch-and-cut algorithm that introduces a local search algorithm to improve the lower bound from the incumbent solution.

To improve the efficiency of the branch-and-cut algorithm, it is also important to improve the lower bound from the incumbent solution S^* (i.e., the best feasible solution obtained so far). We accordingly introduce a simple local search at each node (S^0, S^1) of the branch-and-cut algorithm. We first apply the greedy algorithm from S^1 to obtain an initial feasible solution $S \in F$, where we only consider adding an element $i \in N \setminus (S^0 \cup S^1)$ at each iteration. We then repeatedly replaces S with a better feasible solution S' in its neighborhood $\mathrm{NB}(S)$ until no better feasible solution is found in $\mathrm{NB}(S)$. For a given feasible solution $S \in F$, we define an exchange neighborhood as $\mathrm{NB}(S) = \{S' \subseteq N \mid S \setminus \{i\} \cup \{j\}, i \in S \setminus S^1, j \in N \setminus (S \cup S^0)\}$.

Algorithm LS(S^0, S^1)

Input: A node of the branch-and-cut algorithm (S^0, S^1).

Output: A feasible solution S.

Step 1: Apply the greedy algorithm from S^1 to obtain an initial feasible solution S.

Step 2: Find the best feasible solution $S' \in \mathrm{NB}(S)$. If $f(S') > f(S)$ holds, then set $S \leftarrow S'$ and return to Step 2; otherwise, output S and exit.

To improve lower bounds, for a node (S^0, S^1), we apply the local search algorithm and obtain a feasible solution S. If $f(S) > f(S^*)$ holds, then we replace the incumbent solution S^* with S. If we find the solution S such as $f(S) > f(S^*)$, then we add the S into Q^+ because a good feasible solution tends to be a good constraint to improve upper bounds. The improved branch-and-cut algorithm is defined as follows.

Algorithm BC-ICG+(S, λ)

Input: The initial feasible solution S. The number of feasible solutions to be generated at each node λ.

Output: The incumbent solution S^*.

Step 1: Set $L \leftarrow \{(\emptyset, \emptyset)\}$, $\bar{z}^{(\emptyset,\emptyset)} \leftarrow \infty$, $Q \leftarrow \{S\}$, $Q^+ \leftarrow \{S_{[0]}, \ldots, S_{[k]}\}$ and $S^* \leftarrow S$.

Step 2: Apply the first k iterations of ICG(S, λ) to update the sets Q and Q^+ and the incumbent solution S^*.

Step 3: If $L = \emptyset$ holds, then output the incumbent solution S^* and exit.

Step 4: Extract a node (S^0, S^1) from the top of the list L. Set $S \leftarrow \mathrm{LS}(S^0, S^1)$. If $f(S) > f(S^*)$ holds, then set $S^* \leftarrow S$ and $Q^+ \leftarrow Q^+ \cup \{S^*\}$. If $\bar{z}^{(S^0,S^1)} \leq f(S^*)$ holds, then return to Step 3.

Step 5: Solve BIP(Q^+, S^0, S^1). Let $S^{(S^0,S^1)}$ and $z^{(S^0,S^1)}$ be an optimal solution and the optimal value of BIP(Q^+, S^0, S^1), respectively.

Step 6: Set $Q \leftarrow Q \cup \{S^{(S^0,S^1)}\}$, $Q^+ \leftarrow Q^+ \cup \{S^{(S^0,S^1)}\} \cup$SUB-ICG$(Q, S^{(S^0,S^1)}, \lambda)$.

Step 7: For each feasible solution $S' \in \{S^{(S^0,S^1)}\} \cup$ SUB-ICG$(Q, S^{(S^0,S^1)}, \lambda)$, if $f(S') > f(S^*)$ holds, then set $S^* \leftarrow S$.

Step 8: If $z^{(S^0,S^1)} \leq f(S^*)$, then return to Step 3.

Step 9: If $|S^0 \cup S^1| \leq n-1$ and $|S^1| \leq k-1$ hold, then set $L \leftarrow L \cup \{(S^0 \cup \{i^*\}, S^1), (S^0, S^1 \cup \{i^*\})\}$, $\bar{z}^{(S^0 \cup \{i^*\}, S^1)} \leftarrow z^{(S^0,S^1)}$ and $\bar{z}^{(S^0,S^1 \cup \{i^*\})} \leftarrow z^{(S^0,S^1)}$, where $i^* = \mathrm{argmax}_{i \in N \setminus (S^0 \cup S^1)} f(S^1 \cup \{i\})$. Return to Step 3.

2.4 Computational Results

We tested three existing algorithms: (i) the A* search algorithm with the heuristic function h_{mod} (A*-MOD), (ii) the A* search algorithm with the heuristic function h_{dom} (A*-DOM) and (iii) the constraint generation algorithm (CG) and three proposed algorithms: (iv) the improved constraint generation algorithm (ICG), (v) the branch-and-cut algorithm (BC-ICG) and (vi) the improved branch-and-cut algorithm (BC-ICG+). All algorithms were tested on a personal computer with a 4.0 GHz Intel Core i7 processor and 32 GB memory. We report computational results for three types of well-known benchmark instances called *facility location* (LOC), *weighted coverage* (COV) and *bipartite influence* (INF) according to Kawahara et al. [46] and Sakaue and Ishihata [70]. We note that LOC and COV instances can be formulated as simple MIP models and directly solved by MIP solvers such as CPLEX 12.8. However, the submodular function maximization problem includes many classes of instances that can not be formulated as simple MIP models, e.g., bipartite influence (INF), influence spread [47], document summarization with diversity function [57, 56] and active learning [83].

Facility location (LOC) We are given a set of n locations $N = \{1, \ldots, n\}$ and a set of m clients $M = \{1, \ldots, m\}$. We consider selecting a set of k locations to build facilities. We define $g_{ij} \geq 0$ as the benefit of a client $i \in M$ attaining from a facility of location $j \in N$. We select a set of locations $S \subseteq N$ to build the facilities. Each client $i \in M$ attains the benefit from the most beneficial facility. The total benefit for the clients is defined as

$$f(S) = \sum_{i \in M} \max_{j \in S} g_{ij}. \tag{2.27}$$

Weighted coverage (COV) We are given a set of m items $M = \{1, \ldots, m\}$ and a set of n sensors $N = \{1, \ldots, n\}$. Let $M_j \subseteq M$ be the subset of items covered by a sensor $j \in N$, and $w_i \geq 0$ be a weight of an item $i \in M$. We select a set of sensors $S \subseteq N$ to cover items. The total weighted coverage for the items is defined

as

$$f(S) = \sum_{i \in M} w_i \max_{j \in S} a_{ij}, \tag{2.28}$$

where $a_{ij} = 1$ if $i \in M_j$ holds and $a_{ij} = 0$ otherwise.

Bipartite influence (INF) We are given a set of m targets $M = \{1, \ldots, m\}$ and a set of items $N = \{1, \ldots, n\}$. Given a bipartite graph $G = (M, N; A)$, where $A \subseteq M \times N$ is a set of directed edges, we consider an influence maximization problem on G. Let $p_j \in [0, 1]$ be the activation probability of an item $j \in N$. The probability that a target $i \in M$ gets activated by a set of items $S \subseteq N$ is $1 - \prod_{j \in S}(1 - q_{ij})$, where $q_{ij} = p_j$ if $(i, j) \in A$ holds and $q_{ij} = 0$ otherwise. We select a set of items $S \subseteq N$ to activate targets. The expected number of targets activated by a set of items $S \subseteq N$ is defined as

$$f(S) = \sum_{i \in M} \left(1 - \prod_{j \in S} (1 - q_{ij}) \right). \tag{2.29}$$

We tested all algorithms for 30 classes of randomly generated instances [78] that are characterized by several parameters. We set $m = n + 1$ and $k = 5, 8$ for LOC, COV and INF instances according to Kawahara et al. [46]. We set $n = 20, 30, 40, 50, 60$ for LOC instances and $n = 20, 40, 60, 80, 100$ for COV and INF instances. For LOC instances, g_{ij} is a random value taken from interval $[0, 1]$. For COV instances, a sensor $j \in N$ randomly covers an item $i \in M$ with probability 0.15, and w_i is a random value taken from interval $[0, 1]$. For INF instances, p_j is a random value taken from interval $[0, 1]$, and the bipartite graph G is a random graph in which an edge $(i, j) \in A$ is generated randomly with probability 0.1. We set these parameters to different values from those in Sakaue and Ishihata [70] considering the difference between the cardinality and knapsack constraints. For each class of instances, five instances were generated and tested. For all instances, we set the time limit to two hours (7200 seconds).

Tables 2.1 and 2.2 show the average computation time (in seconds) and the average number of processed nodes of the algorithms for each class of instances,

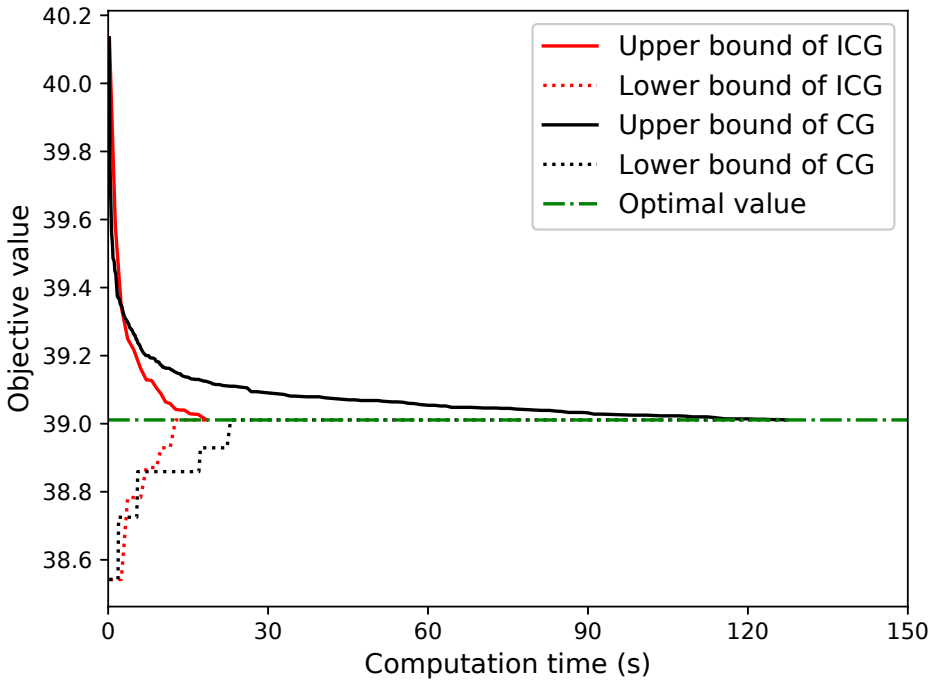

Fig. 2.1: Trends of the upper and lower bounds obtained by CG and ICG with respect to the elapsed computation time (LOC, $n = 40$, $k = 8$)

respectively (see detailed computational results in [77]). If an algorithm could not solve an instance optimally within the time limit, then we set the computation time to 7200 seconds. The best computation time among the compared algorithms is highlighted in bold. Table 2.3 shows the average relative gap $(z_{\mathrm{UB}} - z_{\mathrm{LB}})/z_{\mathrm{LB}} \times 100$ (%), where z_{UB} and z_{LB} are the upper and lower bounds obtained by the algorithms. The numbers in parentheses show the number of instances optimally solved within the time limit.

We first observed that ICG attained better results than CG for almost all classes. Figure 2.1 represents trends of the upper and lower bounds obtained by CG and ICG with respect to the elapsed computation time for an LOC instance with $n = 40$, $k = 8$.

From Figure 2.1, we observed that ICG improves the lower bound as well as the upper bound. Although we have originally proposed ICG to improve the

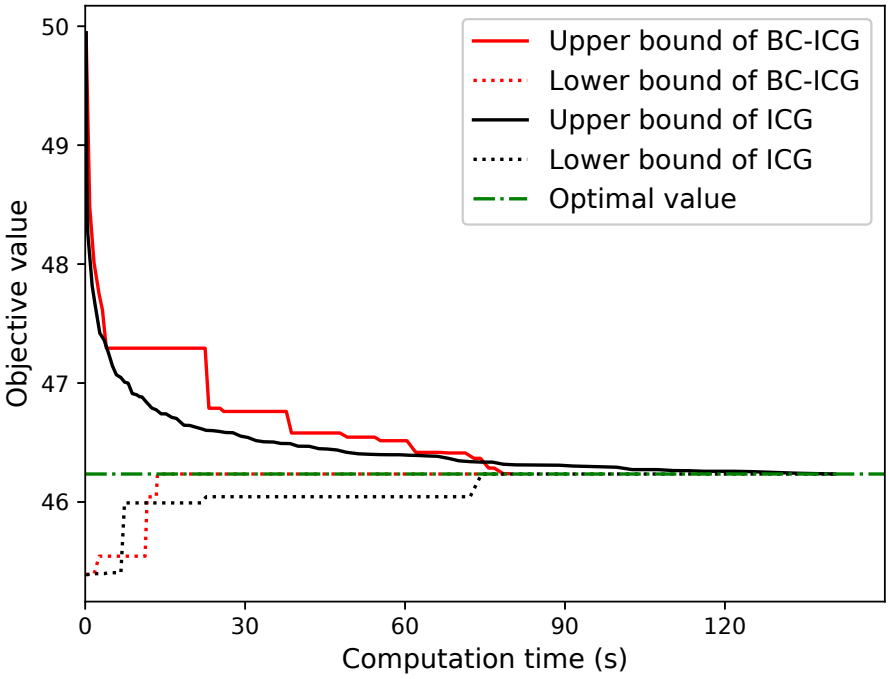

Fig. 2.2: Trends of the upper and lower bounds obtained by ICG and BC-ICG with respect to the elapsed computation time (LOC, $n = 50$, $k = 5$)

upper bounds, it also improves the lower bounds with a number of better feasible solutions than those obtained by CG.

Next, Figure 2.2 represents trends of the upper and lower bounds obtained by ICG and BC-ICG with respect to the elapsed computation time for an LOC instance with $n = 50$, $k = 5$. From Figure 2.2, we observe that BC-ICG could obtain good lower bounds at an early stage, and also it has the strength to improve the upper bound close to the optimal value.

Next, Figure 2.3 represents trends of the upper and lower bounds obtained by A*-MOD and BC-ICG with respect to the elapsed computation time for an COV instance with $n = 60$, $k = 5$. We can observe that BC-ICG could obtain good lower bounds at an early stage, and also it has the strength to improve the upper bound close to the optimal value. However, when A*-MOD attains good upper bounds at an early stage, A*-MOD often performs better than our algorithms.

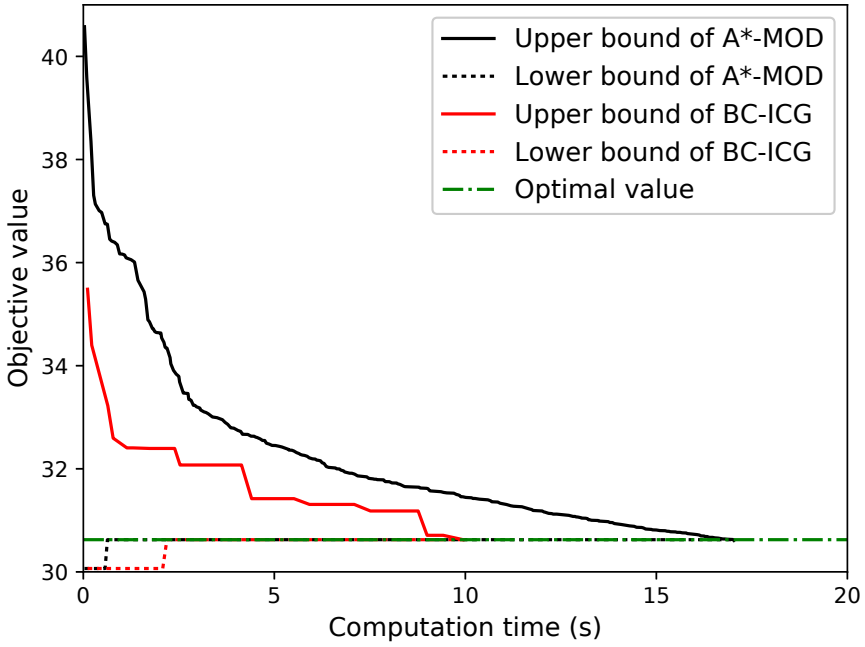

Fig. 2.3: Trends of the upper and lower bounds obtained by A*-MOD and BC-ICG with respect to the elapsed computation time (COV, $n = 60$, $k = 5$)

We observe that ICG, BC-ICG and BC-ICG+ solved optimally 138, 148 and 148 out of all 150 instances, respectively. On the other hand, A*-MOD, A*-DOM and CG solved optimally 124, 114 and 108 instances, respectively. Figure 2.4 shows performance profiles [21] of the algorithms for a parameter $1 \leq \beta \leq 10$. For a given set of algorithms $\mathcal{A}$ and instances $\mathcal{I}$, the performance profile is defined in terms of computation time $T(A, I)$ of an algorithm $A \in \mathcal{A}$ to solve an instance $I \in \mathcal{I}$ optimally. For a pair of algorithm $A \in \mathcal{A}$ and instance $I \in \mathcal{I}$, the performance ratio $R(A, I)$ (i.e., the ratio of computation time over the best) is defined as

$$R(A, I) = \frac{T(A, I)}{\min\limits_{A' \in \mathcal{A}} T(A', I)}, \tag{2.30}$$

where we set $R(A, I) = \infty$ if none of the algorithms solved the instance I optimally. We note that $R(A, I) \geq 1$ holds by definition. The performance profile of an algorithm $A \in \mathcal{A}$ illustrates the function $\rho_A(\gamma)$ that represents the number of instances $I \in \mathcal{I}$ satisfying $R(A, I) \leq \gamma$. From Figure 2.4, we observed that BC-

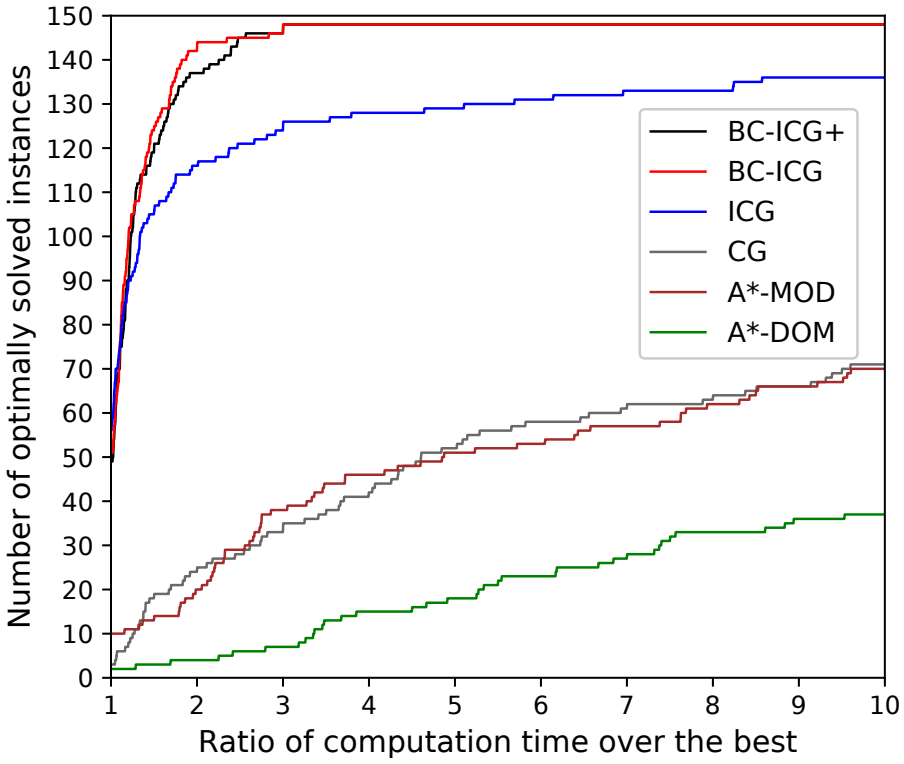

Fig. 2.4: Performance profiles for the algorithms

ICG and BC-ICG+ solve almost all instances with $\gamma = 3$. We also observed that ICG solve about 120 instances optimally out of all 150 instances with $\gamma = 3$ while the conventional constraint generation (CG) solved less than 40 instances.

Table 2.1 also shows the shifted geometric mean [1, 66] of the computation time of the algorithms in the bottom line. For an algorithm $A \in \mathcal{A}$, the shifted geometric mean $SGM(A)$ is defined as follows:

$$SGM(A) = \exp\left(\sum_{I \in \mathcal{I}} \frac{\ln\left(\max\left\{1, T(A, I) + \mathit{shift}\right\}\right)}{|\mathcal{I}|}\right) - \mathit{shift}, \tag{2.31}$$

where we set $\mathit{shift} = 10$ according to [66]. If an algorithm could not solve an instance optimally within the time limit, then we set the computation time to 7200 seconds. We can observe that the improvement between CG and ICG is much larger than the improvement between ICG and BC-ICG. According to comparison of the algorithms through the shifted geometric mean of their computation time, we observed that our BC-ICG and BC-ICG+ performed better than existing algorithms.

Tables 2.1 and 2.3 show that BC-ICG and BC-ICG+ performed better than the existing algorithms, CG, A*-MOD and A*-DOM especially for the instances with $k = 8$. We observe that the A* search algorithms obtain upper bounds by repeatedly adding an increment $f_S(\{i\})$ of an element $i \notin S$ from the current node S until $|S| = k$ holds. The A* search algorithms accordingly obtain much worse upper bounds when the size k of the cardinality constraint grows. On the other hand, our algorithms obtain upper bounds by solving reduced BIP problems with the original cardinality constraint $|S| = k$. Our algorithms accordingly obtain good upper bounds regardless of the size k of the cardinality constraint.

In Table 2.1, neither A*-MOD, A*-DOM, CG nor ICG could solve any of LOC instances with $n = 60$, $k = 8$. However, in Table 2.3, the average relative gap of ICG is much smaller than those of the others. Similarly, BC-ICG and BB-ICG+ attain good feasible solutions very close to optimal ones for a few remaining instances not optimally solved.

From Table 2.2, we observed that BC-ICG and BC-ICG+ attained optimal

solutions while processing much smaller numbers of nodes than A*-MOD and A*-DOM. These results show that ICG attained much better upper bounds than the heuristic functions h_{mod} and h_{dom} for most instances and then the proposed algorithms succeeded in drastically reducing the computation time.

2.5 Conclusion

We present an efficient branch-and-cut algorithm for the non-decreasing submodular function maximization problem based on a BIP formulation with a huge number of constraints. Nemhauser and Wolsey formulated the submodular function maximization problem into a BIP formulation with a huge number of constraints. They proposed a constraint generation algorithm that starts from a small subset of constraints taken from the constraints and repeats solving a reduced BIP problem while adding a new constraint at each iteration. However, their algorithm is often not efficient because it needs to solve many reduced BIP problems until obtaining an optimal solution. To overcome the issue, we propose an improved constraint generation algorithm that starts from a small subset of constraints taken from the constraints and repeats solving a reduced BIP problem while adding a promising set of constraints at each iteration. We incorporate it into a branch-and-cut algorithm to attain good upper bounds with much less computational effort. According to computational results for well-known benchmark instances, our algorithm achieved better performance than the existing A* search algorithms and the conventional constraint generation algorithm. We also confirmed that improved constraint generation algorithm obtained better upper bounds as well as lower bounds than the conventional constraint generation algorithm because our algorithm obtained good feasible solutions at each iteration.

Table 2.1: Computation time (in seconds) of the algorithms

Type	n	k	A*-MOD	A*-DOM	CG	ICG	BC-ICG	BC-ICG+
	20	5	2.73 (5)	4.56 (5)	1.84 (5)	**0.43** (5)	0.56 (5)	0.46 (5)
	30	5	12.86 (5)	30.10 (5)	17.51 (5)	**3.76** (5)	5.36 (5)	4.70 (5)
LOC	40	5	69.29 (5)	149.34 (5)	376.61 (5)	59.01 (5)	**39.87** (5)	40.52 (5)
	50	5	157.73 (5)	454.99 (5)	> 3077.19 (4)	653.63 (5)	**131.46** (5)	143.06 (5)
	60	5	761.56 (5)	1110.01 (5)	> 3573.43 (3)	> 3381.84 (4)	522.35 (5)	**508.97** (5)
	20	8	20.48 (5)	48.60 (5)	0.33 (5)	0.35 (5)	0.36 (5)	**0.30** (5)
	30	8	438.48 (5)	1374.28 (5)	12.57 (5)	**5.75** (5)	6.95 (5)	6.45 (5)
LOC	40	8	3505.04 (5)	> 6241.26 (3)	1628.49 (5)	204.23 (5)	**74.49** (5)	79.86 (5)
	50	8	> 7200.00 (0)	> 7200.00 (0)	> 5867.61 (2)	> 4468.14 (2)	906.46 (5)	**869.84** (5)
	60	8	> 7200.00 (0)	> 7200.00 (0)	> 7200.00 (0)	> 7200.00 (0)	> 4868.44 (3)	> **4758.71** (3)
	20	5	0.18 (5)	0.47 (5)	0.16 (5)	0.10 (5)	0.11 (5)	**0.08** (5)
	40	5	4.45 (5)	15.06 (5)	8.31 (5)	1.74 (5)	**1.59** (5)	1.80 (5)
COV	60	5	66.21 (5)	148.28 (5)	98.95 (5)	12.86 (5)	**11.55** (5)	11.67 (5)
	80	5	135.88 (5)	515.23 (5)	> 4802.33 (2)	182.03 (5)	**49.45** (5)	55.52 (5)
	100	5	512.39 (5)	1833.23 (5)	> 7200.00 (0)	838.32 (5)	**108.11** (5)	118.37 (5)
	20	8	4.62 (5)	3.22 (5)	0.10 (5)	**0.06** (5)	**0.06** (5)	0.07 (5)
	40	8	1551.59 (5)	> 3849.41 (4)	1.52 (5)	1.38 (5)	**1.02** (5)	1.22 (5)
COV	60	8	> 6732.94 (1)	> 7200.00 (0)	13.50 (5)	**5.57** (5)	6.23 (5)	6.37 (5)
	80	8	> 7200.00 (0)	> 7200.00 (0)	> 2267.01 (4)	207.66 (5)	**114.62** (5)	125.64 (5)
	100	8	> 7200.00 (0)	> 7200.00 (0)	> 7200.00 (0)	> 4722.16 (2)	**2879.97** (5)	3059.22 (5)
	20	5	0.09 (5)	0.29 (5)	0.47 (5)	**0.07** (5)	0.09 (5)	**0.07** (5)
	40	5	0.87 (5)	3.42 (5)	6.15 (5)	**0.24** (5)	0.25 (5)	0.28 (5)
INF	60	5	2.65 (5)	16.28 (5)	14.15 (5)	0.39 (5)	0.48 (5)	**0.38** (5)
	80	5	11.45 (5)	65.24 (5)	> 1654.00 (4)	0.78 (5)	0.89 (5)	**0.76** (5)
	100	5	36.68 (5)	229.64 (5)	> 3456.56 (4)	**1.66** (5)	2.07 (5)	2.53 (5)
	20	8	0.94 (5)	2.89 (5)	1.59 (5)	**0.33** (5)	0.55 (5)	0.38 (5)
	40	8	20.78 (5)	129.05 (5)	544.52 (5)	**1.99** (5)	2.40 (5)	2.73 (5)
INF	60	8	284.36 (5)	1928.39 (5)	> 6414.80 (1)	**14.71** (5)	19.15 (5)	25.91 (5)
	80	8	986.43 (5)	> 5710.28 (2)	> 7200.00 (0)	**29.82** (5)	37.85 (5)	51.39 (5)
	100	8	> 5403.02 (3)	> 7200.00 (0)	> 7200.00 (0)	**54.48** (5)	81.03 (5)	120.55 (5)
SGM			> 448.87	> 1041.65	> 471.74	> 89.23	> **64.98**	> 68.00

Table 2.2: Number of processed nodes by the algorithms

Type	n	k	A*-MOD	A*-DOM	BC-ICG	BC-ICG+
	20	5	2.66×10^3 (5)	2.02×10^3 (5)	3.80×10^0 (5)	3.00×10^0 (5)
	30	5	9.70×10^3 (5)	7.16×10^3 (5)	3.90×10^1 (5)	2.62×10^1 (5)
LOC	40	5	3.55×10^4 (5)	3.06×10^4 (5)	2.22×10^2 (5)	1.03×10^2 (5)
	50	5	5.76×10^4 (5)	5.00×10^4 (5)	1.94×10^2 (5)	1.93×10^2 (5)
	60	5	9.82×10^4 (5)	8.56×10^4 (5)	3.42×10^2 (5)	3.06×10^2 (5)
	20	8	2.45×10^4 (5)	7.51×10^3 (5)	1.00×10^0 (5)	1.00×10^0 (5)
	30	8	2.67×10^5 (5)	1.14×10^5 (5)	1.22×10^1 (5)	1.06×10^1 (5)
LOC	40	8	1.48×10^6 (5)	$> 4.68 \times 10^5$ (3)	6.94×10^1 (5)	7.18×10^1 (5)
	50	8	$> 1.77 \times 10^6$ (0)	$> 2.65 \times 10^5$ (0)	2.07×10^2 (5)	1.58×10^2 (5)
	60	8	$> 9.98 \times 10^5$ (0)	$> 1.54 \times 10^5$ (0)	$> 5.00 \times 10^2$ (3)	$> 3.33 \times 10^2$ (3)
	20	5	6.12×10^2 (5)	5.54×10^2 (5)	1.00×10^0 (5)	1.00×10^0 (5)
	40	5	4.88×10^3 (5)	4.74×10^3 (5)	6.20×10^0 (5)	5.40×10^0 (5)
COV	60	5	1.75×10^4 (5)	1.74×10^4 (5)	2.86×10^1 (5)	3.02×10^1 (5)
	80	5	2.14×10^4 (5)	2.14×10^4 (5)	1.02×10^2 (5)	1.07×10^2 (5)
	100	5	4.89×10^4 (5)	4.88×10^4 (5)	1.45×10^2 (5)	1.36×10^2 (5)
	20	8	1.32×10^4 (5)	9.50×10^2 (5)	1.00×10^0 (5)	1.00×10^0 (5)
	40	8	9.77×10^5 (5)	$> 4.55 \times 10^5$ (4)	1.00×10^0 (5)	1.00×10^0 (5)
COV	60	8	$> 1.57 \times 10^6$ (1)	$> 2.74 \times 10^5$ (0)	1.40×10^0 (5)	1.00×10^0 (5)
	80	8	$> 1.01 \times 10^6$ (0)	$> 1.03 \times 10^5$ (0)	6.20×10^0 (5)	4.20×10^0 (5)
	100	8	$> 6.91 \times 10^5$ (0)	$> 6.79 \times 10^4$ (0)	5.34×10^1 (5)	4.90×10^1 (5)
	20	5	2.14×10^2 (5)	2.13×10^2 (5)	1.00×10^0 (5)	1.00×10^0 (5)
	40	5	7.45×10^2 (5)	7.45×10^2 (5)	1.40×10^0 (5)	1.40×10^0 (5)
INF	60	5	1.14×10^3 (5)	1.14×10^3 (5)	1.40×10^0 (5)	1.00×10^0 (5)
	80	5	2.74×10^3 (5)	2.74×10^3 (5)	1.80×10^0 (5)	1.00×10^0 (5)
	100	5	5.96×10^3 (5)	5.96×10^3 (5)	5.00×10^0 (5)	4.20×10^0 (5)
	20	8	2.13×10^3 (5)	1.37×10^3 (5)	4.60×10^0 (5)	2.60×10^0 (5)
	40	8	1.29×10^4 (5)	1.29×10^4 (5)	1.42×10^1 (5)	1.38×10^1 (5)
INF	60	8	6.12×10^4 (5)	6.12×10^4 (5)	6.42×10^1 (5)	6.78×10^1 (5)
	80	8	1.22×10^5 (5)	$> 1.06 \times 10^5$ (2)	8.54×10^1 (5)	9.06×10^1 (5)
	100	8	$> 4.03 \times 10^5$ (3)	$> 7.14 \times 10^4$ (0)	1.22×10^2 (5)	1.27×10^2 (5)

Table 2.3: Relative gap $(z_{\text{UB}} - z_{\text{LB}})/z_{\text{LB}} \times 100$ (%) of the algorithms

Type	n	k	A*-MOD	A*-DOM	CG	ICG	BC-ICG	BC-ICG+
	20	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	30	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
LOC	40	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	50	5	0.00 (5)	0.00 (5)	0.01 (4)	0.00 (5)	0.00 (5)	0.00 (5)
	60	5	0.00 (5)	0.00 (5)	0.31 (3)	0.04 (4)	0.00 (5)	0.00 (5)
	20	8	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	30	8	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
LOC	40	8	0.00 (5)	0.56 (3)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	50	8	3.26 (0)	2.85 (0)	0.20 (2)	0.03 (2)	0.00 (5)	0.00 (5)
	60	8	7.17 (0)	4.75 (0)	0.71 (0)	0.35 (0)	0.16 (3)	0.14 (3)
	20	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	40	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
COV	60	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	80	5	0.00 (5)	0.00 (5)	1.31 (2)	0.00 (5)	0.00 (5)	0.00 (5)
	100	5	0.00 (5)	0.00 (5)	2.55 (0)	0.00 (5)	0.00 (5)	0.00 (5)
	20	8	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	40	8	0.00 (5)	0.40 (4)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
COV	60	8	8.66 (1)	9.89 (0)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	80	8	13.12 (0)	14.83 (0)	0.01 (4)	0.00 (5)	0.00 (5)	0.00 (5)
	100	8	23.24 (0)	19.04 (0)	1.54 (0)	0.10 (2)	0.00 (5)	0.00 (5)
	20	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	40	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
INF	60	5	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	80	5	0.00 (5)	0.00 (5)	0.03 (4)	0.00 (5)	0.00 (5)	0.00 (5)
	100	5	0.00 (5)	0.00 (5)	0.42 (4)	0.00 (5)	0.00 (5)	0.00 (5)
	20	8	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
	40	8	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)	0.00 (5)
INF	60	8	0.00 (5)	0.00 (5)	2.53 (1)	0.00 (5)	0.00 (5)	0.00 (5)
	80	8	0.00 (5)	0.28 (2)	5.52 (0)	0.00 (5)	0.00 (5)	0.00 (5)
	100	8	0.53 (3)	5.00 (0)	6.14 (0)	0.00 (5)	0.00 (5)	0.00 (5)

Chapter 3

An Efficient Branch-and-Cut Algorithm for Approximately Submodular Function Maximization

3.1 Introduction

In many practical situations, utility functions may not be necessarily submodular. However even in those cases, submodularity can be approximately satisfied in various problems such as feature selection [18, 85], boosting influence spread [59], data summarization [7] and combinatorial auction [16, 80]. For this reason, the optimization of *an approximately submodular function* has been attracted an increasing attention recently [19, 39]. By considering such a problem, we can contribute to solving wider range of problems. This type of function is defined with *a submodular ratio* γ, which defined for a set function f as the maximum value $0 < \gamma < 1$ such that $f(S \cup \{i\}) - f(S) \geq \gamma \left(f(T \cup \{i\}) - f(T)\right)$, for all $S \subseteq T \subseteq N$ and $i \notin T$. That is, a submodular ratio γ measures how close the function is to submodular [18, 43].

In this chapter, we address the problem of maximizing a non-decreasing approx-

imately submodular function f under a cardinality constraint (hereafter, referred to as approximately submodular function maximization problem):

$$\begin{aligned} &\text{maximize} \quad f(S) \\ &\text{subject to} \quad |S| \leq k, \;\; S \subseteq N, \end{aligned} \tag{3.1}$$

where $k \leq n$ is a positive integer comprising the cardinality constraint. A set function is non-decreasing if $f(S) \leq f(T)$ for all $S \subseteq T$ and $f(\emptyset) = 0$. Das and Kempe [18] presented a greedy algorithm for the problem that guarantees $(1-e^{-\gamma})$-approximation ratio for a given submodular ratio γ. Chen et al. [13] proposed an A* search algorithm to obtain an exactly optimal solution for the approximately submodular function maximization problem. Their algorithm computes an upper bound by a variant of variable fixing techniques with $O(n)$ oracle queries. Their algorithm quickly finds upper bounds; however the attained upper bounds are not often tight enough to prune nodes of the search tree effectively. Therefore, their algorithm often processes a huge number of nodes of the search tree until obtaining an optimal solution.

We present an efficient branch-and-cut algorithm for the approximately submodular function maximization problem based on its binary integer programming (BIP) formulation with an exponential number of constraints. First, we define approximately submodular function with different formulations by using submodular ratio γ and an upper bound of submodular ratio $\bar{\gamma}$ [43]. Based on the formulation, we derive a BIP formulation of the approximately submodular function maximization problem. Due to the success of the BIP formulation, we develop a constraint generation algorithm [67]. Unfortunately, the constraint generation algorithm is not efficient because of a large number of reduced BIP problems to be solved. To overcome this, we propose an improved constraint generation algorithm, where a promising set of constraints is added at each iteration. We further incorporate it into a branch-and-cut algorithm to attain good upper bounds while solving a smaller number of reduced BIP problems. Finally, we evaluate our algorithms under comparisons with the existing algorithms using three types of well-known benchmark instances and the combinatorial auction problem.

3.2 Existing Algorithms

We review the approximation guarantee of the standard greedy algorithm in Chapter 1, and also we introduce A* search algorithm proposed by Chen et al. [13] for the approximately submodular function maximization problem.

Since our definition of approximately submodular function is slightly different from others [18, 39], we show that the greedy algorithm achieves $(1 - e^{-\gamma})$-approximation ratio with our definition here. The manner of the following proof is similar to the previous proof for Proposition 1.

Proposition 12. *If f is nondecreasing, approximately submodular with a submodular ratio γ, $f(\emptyset) = 0$, then the greedy algorithm returns a solution S which achieves*

$$f(S) \geq \left(1 - \frac{1}{e^{\gamma}}\right) f(S^*),$$

where S^* is an optimal solution.

Proof. By non-decreasingness and approximately submodularity, we obtain the following inequality. We let $S^* = \{i_1, i_2, ..., i_k\}$ and $S^{(t)}$ be a solution obtained by the greedy algorithm at step t.

$$\begin{aligned}
f(S^*) &\leq f(S^{(t)} \cup S^*) \\
&= f(S^{(t)}) + f(S^{(t)} \cup \{i_1\}) - f(S^{(t)}) + \cdots + \\
&\quad f(S^{(t)} \cup \{i_1, \ldots, i_k\}) - f(S^{(t)} \cup \{i_1, \ldots, i_{k-1}\}) \\
&= f(S^{(t)}) + f(\{i_1\} \mid S^{(t)}) + f(\{i_2\} \mid S^{(t)} \cup \{i_1\}) + \cdots + \\
&\quad f(\{i_k\} \mid S^{(t)} \cup \{i_1, \ldots, i_{k-1}\}) \\
&\leq f(S^{(t)}) + (1/\gamma) f(\{i_1\} \mid S^{(t)}) + (1/\gamma) f(\{i_2\} \mid S^{(t)}) + \cdots + \\
&\quad (1/\gamma) f(\{i_k\} \mid S^{(t)}) \\
&= f(S^{(t)}) + (1/\gamma)\{f(\{i_1\} \mid S^{(t)}) + f(\{i_2\} \mid S^{(t)}) + ... + f(\{i_k\} \mid S^{(t)})\} \\
&\leq f(S^{(t)}) + k(1/\gamma)\{f(S^{(t+1)}) - f(S^{(t)})\}
\end{aligned}$$

By reformulation, we obtain the following inequality.

$$\frac{\gamma}{k}\{f(S^*) - f(S^{(t)})\} \leq f(S^{(t+1)}) - f(S^{(t)}).$$

Let $\delta_t = f(S^*) - f(S^{(t)})$. We obtain:

$$\begin{aligned} \frac{\gamma}{k}\delta_t &\leq \delta_t - \delta_{t+1} \\ \delta_{t+1} &\leq \left(1 - \frac{\gamma}{k}\right)\delta_t. \end{aligned}$$

By induction, we have:

$$\delta_k \leq \left(1 - \frac{\gamma}{k}\right)^k \delta_0.$$

We apply this common inequality $1 - x \leq e^{-x}$ to the previous inequality, and we obtain:

$$\delta_k \leq \left(1 - \frac{\gamma}{k}\right)^k \delta_0 \leq \left(e^{-\frac{\gamma}{k}}\right)^k \delta_0 = e^{-\gamma}\delta_0$$

We use the definition of δ_i to reformulate, and obtain:

$$f(S^*) - f(S_k) \leq e^{-\gamma}\left(f(S^*) - f(\emptyset)\right)$$

By $f(\emptyset) = 0$, we obtain

$$f(S_k) \geq \left(1 - \frac{1}{e^{\gamma}}\right) f(S^*).$$

□

Next, we review A* search algorithm proposed by Chen et al. [13] (see details in Section 2.2). Chen et al. [13] presented a heuristic function $h(\cdot)$ for the A* search algorithm. Let S be the current node of the A* search algorithm. We consider the following reduced problem of the approximately submodular function maximization problem for obtaining $h(\cdot)$.

$$\begin{aligned} &\text{maximize} && f_S(T) \\ &\text{subject to} && T \subseteq N \setminus S^+, |T| \leq k - |S|, \end{aligned} \tag{3.2}$$

where $S^+ = \{i \in N \mid i \leq S_{\max}\}$ and $f_S(\cdot) = f(\cdot \mid S)$. Let T^* be an optimal solution of the reduced problem (3.2). By approximately submodularity and non-decreasingness properties, we obtain $\sum_{i \in T}(1/\gamma) f_S(\{i\}) \geq f_S(T)$ for any $T \subseteq N$ and the following inequality.

$$\max_{T \subseteq N \setminus S^+, |T| \leq k - |S|} \frac{1}{\gamma} \sum_{i \in T} f_S(\{i\}) \geq \frac{1}{\gamma} \sum_{i \in T^*} f_S(\{i\}) \geq f_S(T^*). \tag{3.3}$$

Since the reduced problem (3.2) is still NP-hard, we consider obtaining an upper bound of $f_S(T^*)$. Let $\bar{S}^+$ be the non-increasing ordered set with respect to $f_S(\{i\})$ for $i \in N \setminus S^+$. We assume that $|S \cup \bar{S}^+| > k$, because we can obtain the upper bound by computing $f(S \cup \bar{S}^+)$ otherwise. Let $[p] = \{1, \ldots, p\}$ and $\bar{S}^+_{[p]}$ denote the set of the first $p = k - |S|$ elements of the sorted set $\bar{S}^+$. We then define a heuristic function $h(\cdot)$ by

$$h(S) = \frac{1}{\gamma} \sum_{i \in \bar{S}^+_{[p]}} f_S(\{i\}). \tag{3.4}$$

We note that we let $\bar{S}^+_{[p]} \cup S$ be a feasible solution $S' \in F$ for the node S (Step 4). If $f_S(\{i\}) = 0$ holds for some $i \in \bar{S}^+_{[p]}$, then we conclude $f_S(\bar{S}^+_{[p]}) = f_S(T^*)$. An upper bound for a given node S is computed as $\bar{f}(S) = f(S) + h(S)$.

3.3 Binary Integer Programming Formulation

In this section, we formulate the approximately submodular function maximization problem into a binary integer programming (BIP) problem with a huge number of constraints. For the preparation, we need to obtain two kinds of submodular ratios. First, by the definition of approximately submodular function, a submodular ratio γ is obtained as follows:

$$\gamma = \min_{S \subseteq T \subseteq N} \frac{f(\{i\} \mid S)}{f(\{i\} \mid T)}, \tag{3.5}$$

where we regard $0/0 = 1$. To analyze a function f, Johnson et al. [43] defined an upper bound $\bar{\gamma}$ of the submodular ratio γ.

$$\bar{\gamma} = \min_{S \subseteq N} \frac{f(\{i\} \mid S)}{f(\{i\} \mid S \cup \{j\})}, \tag{3.6}$$

where $i \notin S \cup \{j\}$. It is easy to see that $\gamma \leq \bar{\gamma}$ because of the considered solution spaces.

We note that to obtain exact submodular ratios γ and $\bar{\gamma}$ is NP-hard in general according to Bai and Bilmes [5]. They also showed that submodular ratios can not be computed in polynomial time in the oracle model [19]. However, for some

applications it is possible to obtain a lower bound of submodular ratio γ quickly. For example, in feature selection problem, Das and Kempe [18] showed that a lower bound of submodular ratio can be obtained by the smallest eigenvalue of a covariance matrix which is positive semi-definite.

For the preparation of the BIP formulation, we first define an approximately submodular function [79] as follows.

Proposition 13. *Each of the following statements is equivalent and defines an approximately submodular function if there exists constants* $1 > \bar{\gamma} \geq \gamma > 0$*:*

(i) $f(A) - f(A \cap B) \geq \gamma(f(A \cup B) - f(B)), \ \forall A, B \subseteq N.$

(ii) $f(\{i\} \mid S) \geq \gamma \, f(\{i\} \mid T), \ \forall S \subseteq T \subseteq N, \, i \notin T.$

(iii) $f(\{i\} \mid S) \geq \bar{\gamma} \, f(\{i\} \mid S \cup \{e\}), \ \forall S \subseteq N, i \notin S \cup \{e\}.$

(iv) $f(T) \leq f(S) + f(\{j_1\} \mid S) + \frac{1}{\bar{\gamma}} \, f(\{j_2\} \mid S) + \sum_{j \in T \setminus (S \cup \{j_1, j_2\})} \frac{1}{\gamma} \, f(\{j\} \mid S) - \sum_{i \in S \setminus T} \gamma f(\{i\} \mid T \cup S \setminus \{i\}), \ \forall S, T \subseteq N, \, j_1, j_2 \in T \setminus S.$

(v) $f(T) \leq f(S) + f(\{j_1\} \mid S) + \frac{1}{\bar{\gamma}} \, f(\{j_2\} \mid S) + \sum_{j \in T \setminus (S \cup \{j_1, j_2\})} \frac{1}{\gamma} \, f(\{j\} \mid S), \quad \forall S \subseteq T \subseteq N, j_1, j_2 \in T \setminus S.$

The following proof is for Proposition 13. We prove with the following steps, (i) $\Leftrightarrow$ (ii) $\Leftrightarrow$ (iii), (iii) $\Rightarrow$ (iv) $\Rightarrow$ (v) $\Rightarrow$ (iii).

Proof. (i) $\Rightarrow$ (ii). Let $S \subseteq T \subseteq N, i \notin T, A = S \cup \{i\}, B = T$ in (i). We obtain

$$f(\{i\} \mid S) \geq \gamma f(\{i\} \mid T). \tag{3.7}$$

(ii) $\Rightarrow$ (i). Let $\{j_1, \ldots, j_l\} = A \setminus B$. We put that into (ii), and we obtain the following inequality for $i = 1, \ldots, l$.

$$f(\{j_i\} \mid A \cap B \cup \{j_1, \ldots, j_{i-1}\}) \geq \gamma f(\{j_i\} \mid B \cup \{j_1, \ldots, j_{i-1}\}). \tag{3.8}$$

Take sum of the following l inequalities,

$$\begin{array}{rcl} f(\{j_1\} \mid A \cap B) & \geq & \gamma f(\{j_1\} \mid B) \\ f(\{j_2\} \mid A \cap B \cup \{j_1\}) & \geq & \gamma f(\{j_2\} \mid B \cup \{j_1\}) \\ & \vdots & \\ f(\{j_l\} \mid A \cap B \cup \{j_1, \ldots, j_{l-1}\}) & \geq & \gamma\{f(\{j_l\} \mid B \cup \{j_1, \ldots, j_{l-1}\})\}. \end{array} \tag{3.9}$$

We then obtain

$$f(A) - f(A \cap B) \geq \gamma(f(A \cup B) - f(B)). \tag{3.10}$$

(ii) $\Rightarrow$ (iii). It is clear when we let $T = S \cup \{e\}$. Since (ii) considers larger sets than (iii),

$$\gamma = \min_{S \subseteq T \subseteq N} \frac{f(\{i\} \mid S)}{f(\{i\} \mid T)} \leq \min_{S \subseteq N} \frac{f(\{i\} \mid S)}{f(\{i\} \mid S \cup \{e\})} = \bar{\gamma}.$$

(iii) $\Rightarrow$ (ii). If $\forall S \subseteq T \subseteq N, i \notin T$, $T \backslash S = \{k_1, \ldots, k_q\}$, we obtain the following inequalities by (iii). We let $S_j = S \cup \{k_1, \ldots, k_j\}$.

$$\begin{array}{rcl} f(\{i\} \mid S) & \geq & \bar{\gamma} f(\{i\} \mid S \cup \{k_1\}) \\ f(\{i\} \mid S \cup \{k_1\}) & \geq & \bar{\gamma} f(\{i\} \mid S \cup \{k_1, k_2\}) \\ & \vdots & \\ f(\{i\} \mid S \cup \{k_1, \ldots, k_{q-1}\}) & \geq & \bar{\gamma} f(\{i\} \mid T). \end{array} \tag{3.11}$$

By adding these inequalities, we obtain

$$\begin{array}{rcl} f(\{i\} \mid S) & \geq & \bar{\gamma} f(\{i\} \mid T) + (\bar{\gamma} - 1) f(\{i\} \mid S \cup \{k_1\}) + \ \ldots \\ & & + (\bar{\gamma} - 1) f(\{i\} \mid S \cup \{k_1, \ldots, k_{q-1}\}). \end{array} \tag{3.12}$$

By multiplying these inequalities from the bottom, we can obtain the following relation for $f(\{i\} \mid S \cup \{k_1, \ldots, k_t\})$, for $t = 1, \ldots, q-1$,

$$f(\{i\} \mid S \cup \{k_1, \ldots, k_t\}) \geq \bar{\gamma}^{q-t} f(\{i\} \mid T). \tag{3.13}$$

By using (3.13), we rewrite the inequality (3.12) as follows.

$$\begin{array}{rcl} f(\{i\} \mid S) & \geq & \bar{\gamma} f(\{i\} \mid T) + (\bar{\gamma} - 1)\{\bar{\gamma}^{q-1} f(\{i\} \mid T) + \bar{\gamma}^{q-2} f(\{i\} \mid T) \\ & & + \ \ldots \ + \bar{\gamma} f(\{i\} \mid T)\} \\ & = & \bar{\gamma} f(\{i\} \mid T) + \bar{\gamma} f(\{i\} \mid T)(\bar{\gamma} - 1) \left(\dfrac{1 - \bar{\gamma}^{q-1}}{1 - \bar{\gamma}} \right) \\ & = & \bar{\gamma}^q f(\{i\} \mid T). \end{array} \tag{3.14}$$

We note that $\bar{\gamma}^q$ represents the lower bound of γ.

(iii) $\Rightarrow$ (iv). For arbitrary S and T with $T \setminus S = \{j_1, \ldots, j_l\}$ and $S \setminus T = \{k_1, \ldots, k_q\}$, we obtain

$$\begin{aligned}
& f(S \cup T) - f(S) \\
= & \sum_{t=1}^{l} [f(S \cup \{j_1, \ldots, j_t\}) - f(S \cup \{j_1, \ldots, j_{t-1}\})] \\
= & \sum_{t=1}^{l} f(\{j_t\} \mid S \cup \{j_1, \ldots, j_{t-1}\}) \\
= & f(\{j_1\} \mid S) + f(\{j_2\} \mid S \cup \{j_1\}) + f(\{j_3\} \mid S \cup \{j_1, j_2\}) + \ldots \\
& + f(\{j_l\} \mid S \cup \{j_1, \ldots, j_{l-1}\}) \\
\leq & f(\{j_1\} \mid S) + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S) + \frac{1}{\gamma} f(\{j_3\} \mid S) + \cdots + \frac{1}{\gamma} f(\{j_l\} \mid S) \\
= & f(\{j_1\} \mid S) + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S) + \sum_{j \in T \setminus (S \cup \{j_1, j_2\})} \frac{1}{\gamma} f(\{j\} \mid S).
\end{aligned} \tag{3.15}$$

Similarly, we obtain the following inequality.

$$\begin{aligned}
& f(S \cup T) - f(T) \\
= & \sum_{t=1}^{q} [f(T \cup \{k_1, \ldots, k_t\}) - f(T \cup \{k_1, \ldots, k_{t-1}\})] \\
= & \sum_{t=1}^{q} f(\{k_t\} \mid T \cup \{k_1, \ldots, k_t\} \setminus \{k_t\}) \\
\geq & \sum_{t=1}^{q} \gamma f(\{k_t\} \mid T \cup S \setminus \{k_t\}) \\
= & \sum_{i \in S \setminus T} \gamma f(\{i\} \mid T \cup S \setminus \{i\}).
\end{aligned} \tag{3.16}$$

We obtain (iv) by adding these two inequalities.

(iv) $\Rightarrow$ (v). If $\forall S \subseteq T \subseteq N$, then $S \setminus T = \emptyset$. We obtain (v).

(v) $\Rightarrow$ (iii). Let $\forall S \subseteq N, T = S \cup \{j_1, j_2\}, j_1 \in N \setminus (S \cup \{j_2\})$ in (v), then we obtain

$$f(S \cup \{j_1, j_2\}) \leq f(S) + f(\{j_1\} \mid S) + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S). \tag{3.17}$$

$$\begin{aligned}
f(\{j_2\} \mid S \cup \{j_1\}) &= f(S \cup \{j_1, j_2\}) - f(S \cup \{j_1\}) \\
&= f(S \cup \{j_1, j_2\}) - f(\{j_1\} \mid S) - f(S) \\
&\leq \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S).
\end{aligned} \tag{3.18}$$

□

Proposition 14. *Each of the following statements is equivalent and defines a non-decreasing approximately submodular function if there exists constants* $1 > \bar{\gamma} \geq \gamma > 0$*:*

(**i***) $f(A) - f(A \cap B) \geq \gamma \left(f(A \cup B) - f(B)\right), \ \forall A, B \subseteq N.$

(**ii***) $f(\{i\} \mid S) \geq \gamma \, f(\{i\} \mid T) \geq 0, \ \forall S \subseteq T \subseteq N, \, i \notin T.$

(**iv***) $f(T) \leq f(S) + f(\{j_1\} \mid S) + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S) + \sum_{j \in T \setminus (S \cup \{j_1, j_2\})} \frac{1}{\gamma} f(\{j\} \mid S), \ \forall S, T \subseteq N, j_1, j_2 \in T \setminus S.$

Proof. (i*) $\Leftrightarrow$ (ii*). We skip the proof of (i*) $\Rightarrow$ (ii*) since it is the similar manner as (i) $\Rightarrow$ (ii).

(ii*) $\Rightarrow$ (i*). $\gamma f(\{i\} \mid T) \geq 0$. Since $\gamma > 0$, we obtain $f(\{i\} \mid T) \geq 0$ and $f(A) \leq f(B)$. The rest is the same manner as (ii) $\Rightarrow$ (i).

(ii*) $\Rightarrow$ (iv*). It is clear that (ii*) $\Rightarrow$ (ii) $\Rightarrow$ (iv). When $f(\{i\} \mid T) \geq 0$, the last term of (iv) is nonpositive, and that brings us (iv*).

(iv*) $\Rightarrow$ (ii*). Suppose that $S = T \cup \{i\}$ in (iv*), we obtain $f(T) \leq f(T \cup \{i\})$ or $f(\{i\} \mid T) \geq 0$ with $\gamma > 0$. □

We next consider a set X of $(\eta, \boldsymbol{y})$ satisfying the following condition.

$$\eta \leq f(S) + f(\{j_1\} \mid S) y_{j_1} + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S) y_{j_2} + \sum_{j \in N \setminus (S \cup \{j_1, j_2\})} \frac{1}{\gamma} f(\{j\} \mid S) y_j,$$
$$\forall S \subseteq N, |S| \leq k, j_1, j_2 \in N \setminus S, \tag{3.19}$$

where $\boldsymbol{y} = (y_1, \ldots, y_n) \in \{0, 1\}^n$.

Proposition 15. *Suppose $f(\cdot)$ is a non-decreasing approximately submodular function, $(\eta, \boldsymbol{y^T}) \in X$ if and only if $\eta \leq f(T)$, $T = \{j \in N \mid y_j = 1\} \subseteq N$.*

Proof. ($\Leftarrow$). Suppose $\eta \leq f(T)$, then for all $S \subseteq N$, we obtain the following inequality.

$$\begin{aligned} & f(S) + f(\{j_1\} \mid S) y_{j_1}^T + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S) y_{j_2}^T + \sum_{j \in N - (S \cup \{j_1, j_2\})} \frac{1}{\gamma} f(\{j\} \mid S) y_j^T \\ & = f(S) + f(\{j_1\} \mid S) y_{j_1}^T + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid S) y_{j_2}^T + \sum_{j \in T - (S \cup \{j_1, j_2\})} \frac{1}{\gamma} f(\{j\} \mid S) y_j^T \\ & \geq f(T) \geq \eta, \end{aligned} \tag{3.20}$$

where the first inequality comes from Proposition 14 (iv*).

($\Rightarrow$). If $(\eta,\, y^T) \in X$, we obtain the following inequality,

$$\begin{aligned} \eta \;&\leq\; f(T) + f(\{j_1\} \mid T) y_{j_1}^T + \frac{1}{\bar{\gamma}} f(\{j_2\} \mid T)\; y_{j_2}^T + \sum_{j \in N \setminus (T \cup \{j_1, j_2\})} \frac{1}{\gamma} f(\{j\} \mid T)\; y_j^T \\ &=\; f(T). \end{aligned} \tag{3.21}$$

□

We now replace $\bar{\gamma}$ with γ due to $\bar{\gamma} \geq \gamma$. We formulate the approximately submodular function maximization problem into the following BIP problem (3.22).

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & z \leq f(S) + f(\{j\} \mid S)\; y_j + \sum_{i \in N \setminus (S \cup \{j\})} \frac{1}{\gamma}\; f(\{i\} \mid S)\; y_i, \\ & j \in N \setminus S,\; S \in F, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0, 1\},\; i \in N, \end{aligned} \tag{3.22}$$

where $j = \text{argmax}_{i \in N \setminus S} f(\{i\} \mid S)$. F denotes the set of all feasible solutions satisfying the cardinality constraint $|S| \leq k$.

We note that by solving the problem (3.22), we obtain the optimal value and an optimal solution. Although an original problem does not hold submodularity,

by formulating the original problem with an approximately submodular function, we can obtain an optimal solution of the original problem.

3.4 Proposed Algorithms

We first present an existing exact algorithm called a constraint generation algorithm for the approximately submodular function maximization problem based on the algorithm [67]. The constraint generation algorithm often needs to solve a large number of reduced BIP problems because of generating only one constraint at each iteration. We accordingly propose an improved constraint generation algorithm to generate a promising set of constraints for attaining good upper bounds while solving a smaller number of reduced BIP problems. Moreover, we develop a branch-and-cut algorithm by using the above algorithm.

3.4.1 Constraint Generation Algorithm

We introduce an exact algorithm called a constraint generation algorithm proposed by Nemhauser and Wolsey [67] (see details in Section 2.2.2). The different part of the algorithm is presented as follows. Given a set of feasible solutions $Q \subseteq F$, we define BIP(Q) as the following reduced BIP problem of the problem (3.22).

$$\begin{aligned} \text{maximize} \quad & z \\ \text{subject to} \quad & z \leq f(S) + f(\{j\} \mid S)\, y_j + \sum_{i \in N \setminus (S \cup \{j\})} \frac{1}{\gamma}\, f(\{i\} \mid S)\, y_i,\ S \in Q, \\ & \sum_{i \in N} y_i \leq k, \\ & y_i \in \{0,1\},\ i \in N, \end{aligned} \tag{3.23}$$

where $j = \text{argmax}_{i \in N \setminus S} f(\{i\} \mid S)$.

Although a lower bound $\underline{\gamma}$ of the exact γ is sufficient for solving the problem (3.22) exactly, a better lower bound of submodular ratio $\underline{\gamma}$ close to the exact submodular ratio γ attains higher efficiency for the constraint generation algorithm.

We note that when we consider a general non-decreasing set function f with a submodular ratio γ which is able to be obtained, this algorithm has a potential to obtain an optimal solution. Although this algorithm is able to obtain an optimal solution with a submodular ratio γ close to 0, the efficiency of the algorithm becomes very low. In this thesis, we mainly consider problems with a given submodular ratio γ.

3.4.2 Improved Constraint Generation Algorithm

We propose an improved constraint geration algorithm based on the previous imporoved constraint generation algorithm (see details in Section 2.3.1). Here, we only present the different part of the algorithm. Let $\boldsymbol{y}^{(t)} = (y_1^{(t)}, \ldots, y_n^{(t)})$ and $z^{(t)}$ be an optimal solution and the optimal value of BIP(Q) at the t-th iteration of the constraint generation algorithm, respectively. We note that $z^{(t)}$ gives an upper bound of the optimal value of the problem (3.22). To improve the upper bound $z^{(t)}$, it is necessary to add a new feasible solution $S' \in F$ to Q satisfying the following inequality.

$$z^{(t)} > f(S') + f(\{j\} \mid S')y_j^{(t)} + \sum_{i \in N \setminus (S' \cup \{j\})} \frac{1}{\gamma} f(\{i\} \mid S')\, y_i^{(t)}. \tag{3.24}$$

After solving BIP(Q), we obtain at least one feasible solution $S^\natural \in Q$ attaining the optimal value $z^{(t)}$ of BIP(Q), i.e.,

$$z^{(t)} = f(S^\natural) + f(\{j\}) \mid S^\natural)\, y_j^{(t)} + \sum_{i \in N \setminus (S^\natural \cup \{j\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)}. \tag{3.25}$$

Let $S^{(t)}$ be the optimal solution of BIP(Q) corresponding to $\boldsymbol{y}^{(t)}$, where we assume $S^{(t)} \notin Q$.

We then consider adding an element $j \in S^{(t)} \setminus S^\natural$ to $S^\natural$. In the case with satisfying $j = \mathrm{argmax}_{i \in N \setminus S} f(\{i\} \mid S)$, we obtain the following inequality by approximately submodularity:

$$
\begin{aligned}
z^{(t)} &= f(S^\natural) + f(\{j\} \mid S^\natural)\, y_j^{(t)} + \sum_{i \in N \setminus (S^\natural \cup \{j\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&= f(S^\natural) + f(\{j\} \mid S^\natural) + \sum_{i \in N \setminus (S^\natural \cup \{j\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&= f(S^\natural \cup \{j\}) + \sum_{i \in N \setminus (S^\natural \cup \{j\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&\geq f(S^\natural \cup \{j\}) + \sum_{i \in N \setminus (S^\natural \cup \{j\})} f(\{i\} \mid S^\natural \cup \{j\})\, y_i^{(t)},
\end{aligned}
\tag{3.26}
$$

where $y_j^{(t)} = 1$ due to $j \in S^{(t)}$.

In the other case when we add $u \in S^{(t)} \setminus S^\natural$ into $S^\natural$, where $u \neq \mathrm{argmax}_{i \in N \setminus S} f(\{i\} \mid S)$, we obtain a similar inequality.

$$
\begin{aligned}
z^{(t)} &= f(S^\natural) + f(\{j\} \mid S^\natural)\, y_j^{(t)} + \sum_{i \in N \setminus (S^\natural \cup \{j\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&= f(S^\natural) + f(\{j\} \mid S^\natural)\, y_j^{(t)} + \frac{1}{\gamma}\, f(\{u\} \mid S^\natural)\, y_u^{(t)} + \\
&\quad \sum_{i \in N \setminus (S^\natural \cup \{j,u\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&= f(S^\natural) + f(\{j\} \mid S^\natural)\, y_j^{(t)} + \frac{1}{\gamma}\, f(\{u\} \mid S^\natural) + \sum_{i \in N \setminus (S^\natural \cup \{j,u\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&\geq f(S^\natural \cup \{u\}) + f(\{j\} \mid S^\natural)\, y_j^{(t)} + \sum_{i \in N \setminus (S^\natural \cup \{j,u\})} \frac{1}{\gamma}\, f(\{i\} \mid S^\natural)\, y_i^{(t)} \\
&\geq f(S^\natural \cup \{u\}) + \gamma f(\{j\} \mid S^\natural \cup \{u\})\, y_j^{(t)} + \\
&\quad \sum_{i \in N \setminus (S^\natural \cup \{j,u\})} f(\{i\} \mid S^\natural \cup \{u\})\, y_i^{(t)},
\end{aligned}
\tag{3.27}
$$

where $y_u^{(t)} = 1$ due to $u \in S^{(t)}$.

By the inequality (3.26), (3.27) with $\gamma \simeq 1$, we observe that it is preferable to add the element $j \in S^{(t)} \setminus S^\natural$ to $S^\natural$ for improving the upper bound $z^{(t)}$. It is necessary to remove another element $i \in S^\natural$ if $|S^\natural| = k$ holds.

3.4.3 Branch-and-Cut Algorithm

We propose a branch-and-cut algorithm incorporating the improved constraint generation algorithm. The process of the branch-and-cut algorithm is almost same

as the previous one (see details in Section 2.3.2). The different part of the algorithm is as follows. The value of a node (S^0, S^1) is defined as the optimal value $z^{(S^0,S^1)}$ of the following reduced BIP problem BIP(Q^+, S^0, S^1):

$$
\begin{aligned}
\text{maximize} \quad & z \\
\text{subject to} \quad & z \leq f(S) + f(\{j\} \mid S) y_j + \sum_{i \in N \setminus (S \cup \{j\})} \frac{1}{\gamma} f(\{i\} \mid S) y_i, \\
& S \in Q^+, \\
& \sum_{i \in N \setminus (S^0 \cup S^1)} y_i \leq k - |S^1|, \\
& y_i \in \{0,1\}, \quad i \in N \setminus (S^0 \cup S^1), \\
& y_i = 0, \qquad i \in S^0, \\
& y_i = 1, \qquad i \in S^1,
\end{aligned}
\tag{3.28}
$$

where $j = \text{argmax}_{i \in N \setminus S} f(\{i\} \mid S)$, and Q^+ is the set of feasible solution generated by the improved constraint generation algorithm so far.

3.5 Computational Results

3.5.1 Application in Combinatorial Auction

We consider the combinatorial auction (CA) problem [16, 80] that asks a bidder to select a package of items to maximize its utility, which is formulated as the approximately submodular function maximization problem. A greedy algorithm obtains a feasible solution quickly, however it often fails to obtain important items for the problem. To do so, we need an optimal solution as well as other good solutions whose objective value are close to the optimal value. For this purpose, we modify BC-ICG to hold all incumbent solutions obtained so far as well as the final incumbent solution. For this problem, we can obtain a lower bound of submodular ratio quickly, therefore we show the formula for it.

Combinatorial auction (CA) We are given a set of n items $N = \{1, \ldots, n\}$. We select a set of items $S \subseteq N$ to make a package of items. We define w_i as the

individual utility of an item $i \in N$ and r_{ij} as the mutual utility for a pair of items $i, j \in N$. The utility of a package of items is defined as

$$f(S) = \sum_{i \in S} w_i + \sum_{i,j \in S} r_{ij}. \tag{3.29}$$

We have tested BC-ICG for an instance arising from a supermarket transaction data containing 170 items and 9835 transactions.[1] We set the size of the package $k = 2, 3$. We also set the individual utility randomly $w_i \in [1, 2]$, while setting the mutual utility $r_{ij} \in [-0.09, 0.01]$ according to the number of times that both items i, j are selected in the same transaction. We obtain a lower bound $\underline{\gamma}$ of the submodular ratio γ by the following formula:

$$\underline{\gamma} = \min_{i \in N} \left\{ \frac{w_i + \sum_{j \in L_{[q_1]}} r_{ij}}{w_i + \sum_{j \in U_{[q_2]}} r_{ij}} \right\}. \tag{3.30}$$

where the sets U and L are the non-increasing positive and non-decreasing negative ordered set with respect to r_{ij} for an item $i \in N$, respectively, and $q_1 = \min\{k - 1, |L|\}$, $q_2 = \min\{k - 1, |U|\}$. We note that $L_{[q_1]}$ (resp., $U_{[q_2]}$) represents the first q_1 (resp., q_2) elements of a sorted set L (resp., U). For the instance, we obtained $\underline{\gamma} = 0.906, 0.736$ with $k = 2, 3$, respectively.

Table 3.1 shows the frequency of items in the series of solutions obtained by BC-ICG. The optimal solutions obtained by BC-ICG are [“yogurt”, “frozen vegetables”], [“yogurt”, “sugar”, “organic products”] for $k = 2, 3$, respectively. On the other hand, the feasible solutions obtained the greedy algorithm are [“tea”, “yogurt”], [“tea”, “yogurt”, “frozen vegetables”] for $k = 2, 3$, respectively. We note that “tea” is not selected in the optimal solutions while it has the largest value of the 170 items. That is, the greedy algorithm sometimes fails to attain important items constitute the optimal solution. We can also see the difference between the solution obtained by a greedy algorithm and the optimal solution for $k = 3$. According to the Table 3.1, by obtaining the frequency of items in the

[1] http://www.sci.csueastbay.edu/ esuess/classes/Statistics_6620/Presentations/ml13/groceries.csv

Table 3.1: Frequency of items in a series of solutions obtained by BC-ICG.

	"tea"	"yogurt"	"sugar"	"organic products"	"frozen vegetables"	"softner"
$k = 2$	1	3	2	1	2	1
$k = 3$	1	4	2	2	3	0

series of solutions, we can see which elements are more important or are selected in good solutions.

We chose the mutual utility $r_{ij} \in [-0.09, 0.01]$ because we need to keep the submodular ratio value higher than 0.7. If we set $r_{ij} \in [-0.09, 0.05]$, then the submodular ratio became very small and we could not solve the problem within a reasonable computation time. On the other hand, if we set $r_{ij} \in [-0.09, 0.005]$, then we can obtain larger submodular ratio than those of this example. Moreover, when k becomes larger, a lower bound of submodular ratio $\underline{\gamma}$ often becomes smaller. Therefore, we often need to arrange the mutual setting utility value r_{ij} and k for maintaining the proper submodular ratio value such as $\underline{\gamma} > 0.7$.

3.5.2 Benchmark Instances

We tested two existing algorithms: (i) the A* search algorithm with the heuristic function h_{mod} (A*-MOD), (ii) the constraint generation algorithm (CG) and two proposed algorithms: (iii) the improved constraint generation algorithm (ICG), (iv) the branch-and-cut algorithm (BC-ICG). All algorithms were tested on a personal computer with a 4.0 GHz Intel Core i7 processor and 32 GB memory. For CG, ICG, and BC-ICG, we use a mixed integer programming (MIP) solver called CPLEX 12.8 [40] for solving reduced BIP problems, and the number of feasible solutions to be generated at each iteration λ is set to $10k$ based on computational results of preliminary experiments.

We report computational results for three types of well-known benchmark instances called *facility location* (LOC), *weighted coverage* (COV), and *bipartite influence* (INF) according to Kawahara et al. [46] and Sakaue and Ishihata [70]. We note that those instances were originally generated for the submodular function

maximization problem. For generating instances for the approximately submodular function maximization problem, we replace the original utility function $f(S)$ with $f(S) + r_S$ for a number of feasible solutions, where r_S is a reward value for a feasible solution $S \subseteq F$. We randomly selected $1000k$ feasible solutions $S \subseteq F$ and modified their utility functions while satisfying non-decreasing and a given lower bound $\underline{\gamma} = 0.8$ of the submodular ratio γ.

We tested all algorithms for 18 classes of randomly generated instances that are characterized by several parameters. We set $m = n + 1$ and $k = 5, 8$ for LOC, COV and INF instances according to Kawahara et al. [46]. We set $n = 20, 30, 40$ for LOC instances and $n = 20, 40, 60$ for COV and INF instances. For LOC instances, g_{ij} is a random value taken from interval $[0, 1]$. For COV instances, a sensor $j \in N$ randomly covers an item $i \in M$ with probability 0.15, and w_i is a random value taken from interval $[0, 1]$. For INF instances, p_j is a random value taken from interval $[0, 1]$, and the bipartite graph G is a random graph in which an edge $(i, j) \in A$ is generated randomly with probability 0.1. We set these parameters to different values from those in [70] considering the difference between the cardinality and knapsack constraints. For each class of instances, five instances were generated and tested. For all instances, we set the time limit to 7200 seconds.

Tables 3.2 and 3.3 show the average computation time (in seconds) and the average number of processed nodes of the algorithms for each class of instances, respectively. If an algorithm could not solve an instance optimally within the time limit, then we set the computation time to 7200 seconds. The best computation time among the compared algorithms is highlighted in bold. The numbers in parentheses show the number of instances optimally solved within the time limit. According to Table 3.2, ICG performed better than CG in most of instances. We can see that when $k = 8$, ICG tends to perform better than A*-MOD although when $k = 5$, A*-MOD performed better than our algorithms. Our algorithms can obtain good upper bounds regardless of k. Therefore, when k becomes very large, our algorithms sometimes perform very well. However, the existing A*-MOD adds increments from a node S, the upper bounds tended to become large

when k is large. However, when k is smaller, A*-MOD performed well because the computation time for each node S is much quicker than our algorithms which solve a reduced BIP problem for each node.

Figure 3.1 shows trends of the upper and lower bounds obtained of CG and ICG with respect to the computation time for an LOC instance ($n = 30$, $k = 8$). We can see that ICG attained better upper and lower bounds than those of CG by generating good feasible solutions and adding them as constraints at each iteration. The number of iterations that CG and ICG solved a reduced BIP problem were 275 and 35, respectively. We succeeded in improving the efficiency of CG by reducing the number of iterations.

Next, Figure 3.2 shows trends of upper and lower bounds obtained of ICG and BC-ICG with respect to the computation time for an LOC instance ($n = 40$, $k = 5$). We can see that BC-ICG obtained better lower bounds than those of ICG by searching nodes of a search tree. BC-ICG has the strength to improve the upper bound close to the optimal value. We obtained that ICG solved 105 reduced BIP problems, and BC-ICG solved 299 reduced BIP problems (searched 294 nodes). Although BC-ICG solved more reduced BIP problems than ICG, a tree search improved the efficiency to obtain better upper bounds.

Next, Figure 3.3 shows trends of upper and lower bounds obtained of A*-MOD and BC-ICG with respect to the computation time for an LOC instance ($n = 60$, $k = 5$). When $k = 8$, A*-MOD often obtained large upper bounds and spent much computation time, therefore we see the trend of an instance of $k = 5$. We can observe that our BC-ICG obtains much better upper bounds at an early stage than those of A*-MOD. We can see that BC-ICG improved upper bounds more efficient than A*-MOD according to the slope of the upper bounds trend. However, when A*-MOD obtains good upper bounds at an early stage, the algorithm tends to perform better than our algorithms.

According to Table 3.3, BC-ICG processed much smaller number of nodes than A*-MOD due to ICG attaining good upper bounds. We note that A*-MOD performed well for INF instances, because the utility function $f(S)$ became close to

linear and A*-MOD gave tight upper bound $\bar{f}(S) = f(S) + h(S)$ for the instances.

Figure 3.4 shows performance profiles [21] of the algorithms for a parameter $1 \leq \beta \leq 10$ (see details in Chapter 2). We observed that BC-ICG solved 78 instances optimally out of all 90 instances while CG solved only 41 instances with $\beta = 3$. These computational results show that BC-ICG improved the efficiency of the conventional CG.

Table 3.2 also shows the shifted geometric mean [66] of the computation time of the algorithms in the bottom line (see the details in Chapter 2). We can see that the improvement between ICG and CG is much larger than the improvement between ICG and BC-ICG. According to comparison of the algorithms through the shifted geometric mean of their computation time, we observed that our ICG improved CG, and BC-ICG performed better than other algorithms.

We note that if a submodular ratio γ is small, then it is very difficult to solve the original problem (3.22) optimally. The difficulty comes with a situation when we need to prepare all the constraints. The situation occurs as follows. Suppose that we have any constraint $S \in F; S \neq S^*$, a solution vector $\boldsymbol{y} = (y_1, \ldots, y_n)$, $y_i \in \{0, 1\}$. We consider a right hand side of the inequality of a problem (3.22);

$$f(S) + f(\{j\} \mid S)y_j + \sum_{i \in N \setminus (S \cup \{j\})} \frac{1}{\gamma} f(\{i\} \mid S)\, y_i, \tag{3.31}$$

where $j = \mathrm{argmax}_{i \in N \setminus S} f(\{i\} \mid S)$. If the value of the right hand side becomes larger than the optimal value of the original problem for any S, we end up with preparing all the constraints. Therefore, it is very important to maintain a submodular ratio γ to be large.

We also consider a situation that it is difficult to obtain an optimal solution. In the situation, we can use the algorithms as approximation algorithms which output a feasible solution when the gap between upper and lower bounds becomes small enough.

Table 3.2: Computation time (in seconds) of the proposed and the existing algorithms.

Type	n	k	$\underline{\gamma}$	A*-MOD	CG	ICG	BC-ICG
	20	5	0.8	0.97 (5)	1.39 (5)	**0.52** (5)	0.70 (5)
LOC	30	5	0.8	**8.69** (5)	27.98 (5)	11.57 (5)	10.98 (5)
	40	5	0.8	**51.00** (5)	2369.62 (5)	973.04 (5)	183.09 (5)
	20	8	0.8	13.22 (5)	30.01 (5)	0.35 (5)	**0.32** (5)
LOC	30	8	0.8	409.50 (5)	99.74 (5)	21.77 (5)	**17.29** (5)
	40	8	0.8	$>$ 5703.85 (4)	$>$ 5367.68 (3)	$>$ 3433.23 (3)	**667.62** (5)
	20	5	0.8	0.40 (5)	0.36 (5)	**0.25** (5)	0.29 (5)
COV	40	5	0.8	16.36 (5)	52.49 (5)	**9.98** (5)	14.15 (5)
	60	5	0.8	163.36 (5)	1739.64 (5)	262.39 (5)	**145.73** (5)
	20	8	0.8	8.71 (5)	**0.07** (5)	0.10 (5)	0.09 (5)
COV	40	8	0.8	$>$ 3468.47 (4)	$>$ 1441.96 (4)	**2.31** (5)	3.18 (5)
	60	8	0.8	$>$ 7200.00 (0)	108.86 (5)	35.81 (5)	**32.49** (5)
	20	5	0.8	**0.36** (5)	2.46 (5)	0.61 (5)	1.07 (5)
INF	40	5	0.8	**8.61** (5)	$>$ 1992.35 (4)	15.98 (5)	22.14 (5)
	60	5	0.8	61.20 (5)	$>$ 4467.06 (2)	**16.68** (5)	25.17 (5)
	20	8	0.8	3.32 (5)	3.46 (5)	15.98 (5)	**1.75** (5)
INF	40	8	0.8	**438.83** (5)	$>$ 3775.80 (3)	$>$ 1670.35 (4)	912.62 (5)
	60	8	0.8	$>$ **4980.66** (4)	$>$ 7200.00 (0)	$>$ 7200.00 (0)	$>$ 7200.00 (0)
SGM				$>$ 95.88	$>$ 118.12	$>$ 39.69	$>$ **31.42**

Table 3.3: Number of processed nodes by the proposed and the existing algorithms.

Type	n	k	$\underline{\gamma}$	A*-MOD	BC-ICG
	20	5	0.8	5.62×10^3 (5)	8.60×10^0 (5)
LOC	30	5	0.8	2.74×10^4 (5)	8.30×10^1 (5)
	40	5	0.8	1.03×10^5 (5)	3.77×10^2 (5)
	20	8	0.8	4.68×10^4 (5)	1.00×10^0 (5)
LOC	30	8	0.8	6.55×10^5 (5)	4.78×10^1 (5)
	40	8	0.8	$> 3.71 \times 10^6$ (4)	3.09×10^2 (5)
	20	5	0.8	2.03×10^3 (5)	1.00×10^0 (5)
COV	40	5	0.8	2.63×10^4 (5)	6.14×10^1 (5)
	60	5	0.8	1.07×10^5 (5)	1.96×10^2 (5)
	20	8	0.8	3.54×10^4 (5)	1.00×10^0 (5)
COV	40	8	0.8	$> 2.71 \times 10^6$ (4)	1.00×10^0 (5)
	60	8	0.8	$> 3.00 \times 10^6$ (0)	9.80×10^0 (5)
	20	5	0.8	1.16×10^3 (5)	2.62×10^1 (5)
INF	40	5	0.8	9.04×10^3 (5)	1.86×10^2 (5)
	60	5	0.8	3.02×10^4 (5)	1.71×10^2 (5)
	20	8	0.8	9.51×10^3 (5)	1.70×10^1 (5)
INF	40	8	0.8	3.90×10^5 (5)	5.20×10^2 (5)
	60	8	0.8	$> 2.01 \times 10^6$ (4)	$> 1.94 \times 10^3$ (0)

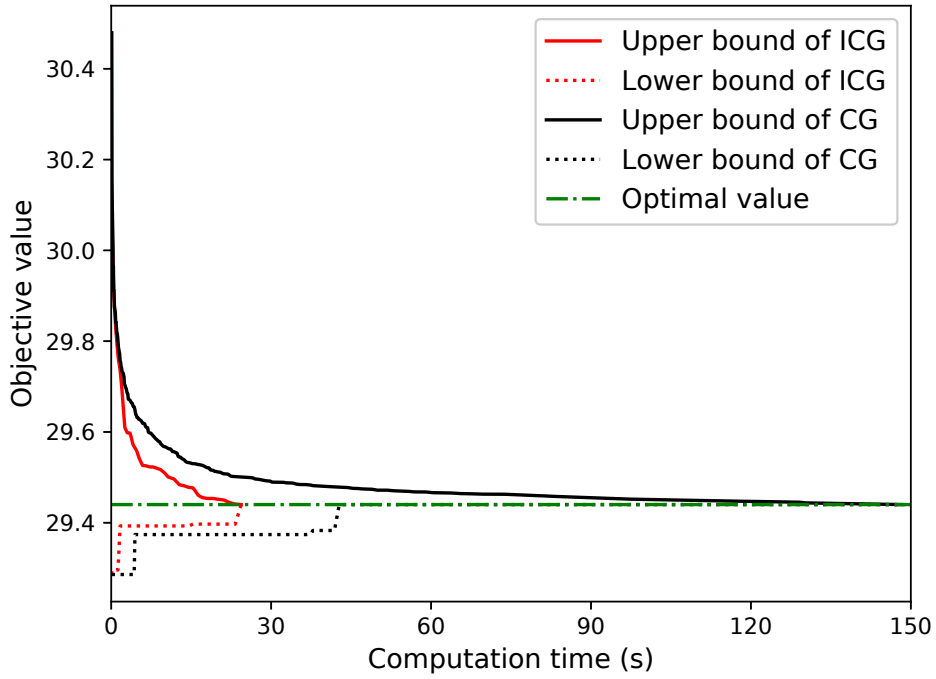

Fig. 3.1: Trends of the upper and lower bounds obtained by CG and ICG with respect to the elapsed computation time (LOC, $n = 30$, $k = 8$).

3.6 Conclusion

In this chapter, we define an approximately submodular function with a submodular ratio γ and an upper bound of submodular ratio $\bar{\gamma}$ [43]. By using the definition, we succeed in formulating the non-decreasing approximately submodular function maximization problem into a BIP formulation with an exponential number of constraints. For the problem, we develop a constraint generation algorithm which was propsed for submodular function maximization problem. The constraint generation algorithm starts from a small subset of constraints and repeats solving a reduced BIP problem while adding a new constraint at each iteration. Therefore, the constraint generation algorithm is often not efficient because the algorithm needs to solve too many reduced BIP problems until obtaining an optimal solution. To overcome the difficulties, we propose an improved constraint generation

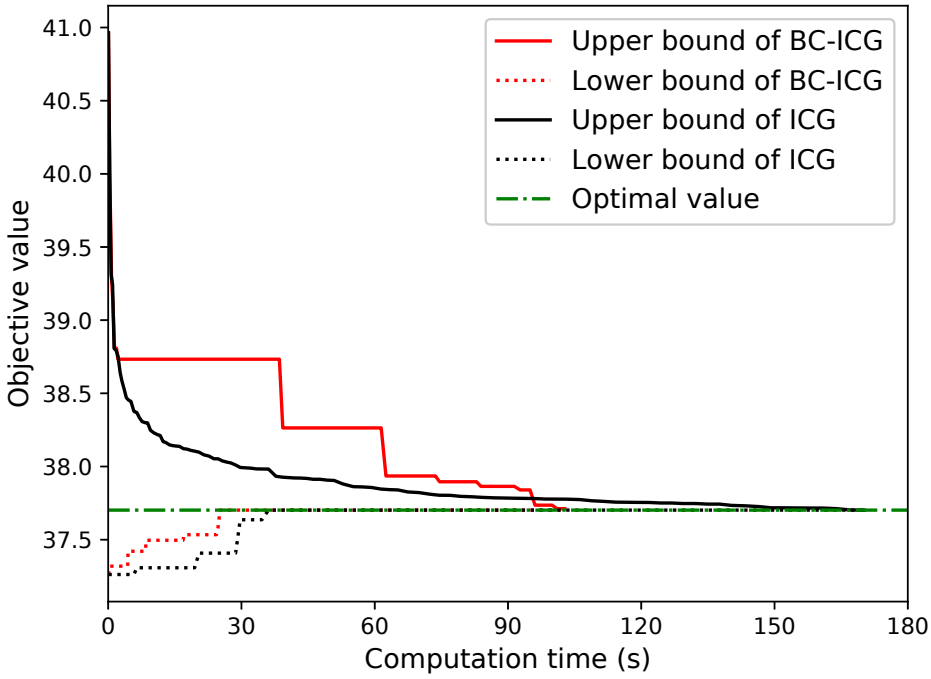

Fig. 3.2: Trends of the upper and lower bounds obtained by ICG and BC-ICG with respect to the elapsed computation time (LOC, $n = 40$, $k = 5$).

algorithm that starts from a small subset of constraints and repeats solving a reduced BIP problem while adding a promising set of constraints at each iteration. We then incorporate it into a branch-and-cut algorithm to attain good upper bounds. According to computational results for three types of well-known benchmark instances, the improved constraint generation performed better than conventional constraint generation algorithm. The improved constraint generation obtained better upper bounds as well as good lower bounds since it generates many good feasible solutions at each iteration. According to the performance profile and the shifted geometric mean criteria, we confirmed that branch-and-cut algorithm performed better than other algorithms, especially in larger-size instances.

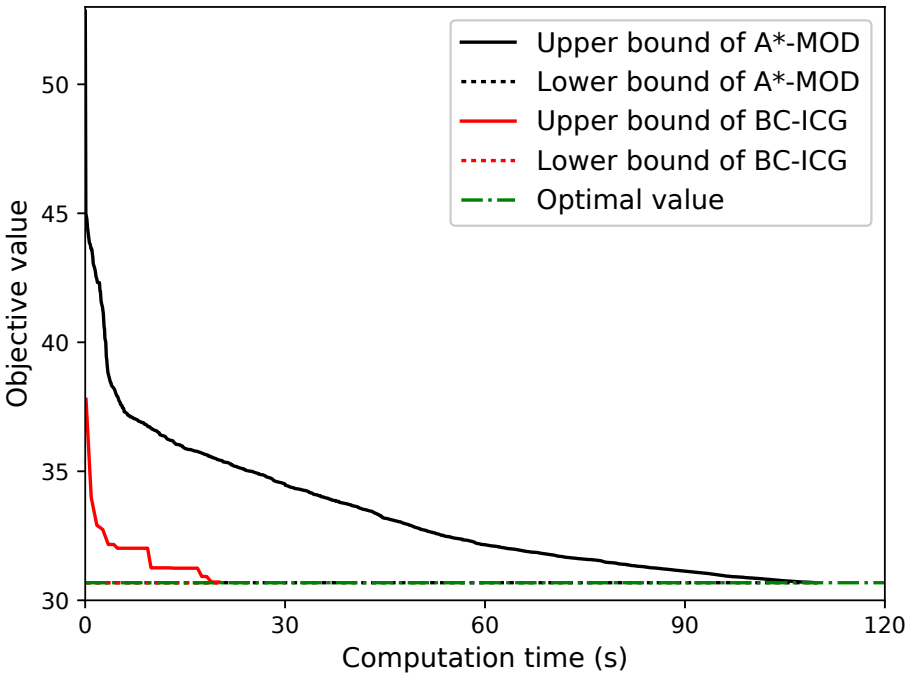

Fig. 3.3: Trends of the upper and lower bounds obtained by A*-MOD and BC-ICG with respect to the elapsed computation time (LOC, $n = 60$, $k = 5$).

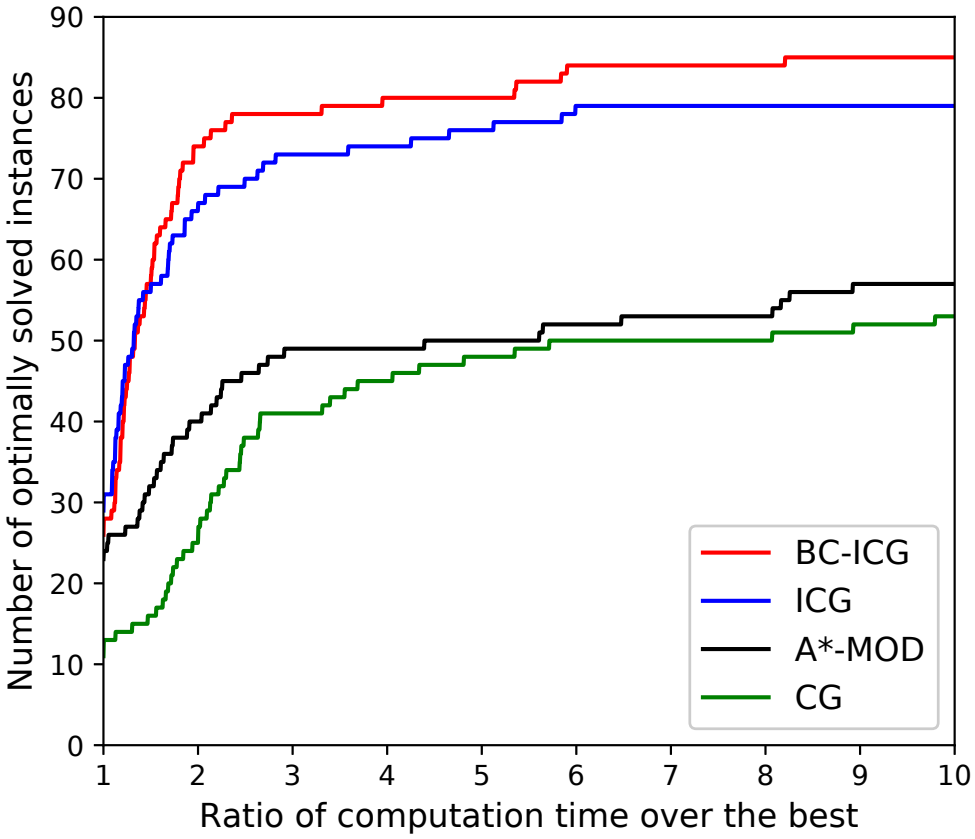

Fig. 3.4: Performance profile for the algorithms.

Chapter 4

Conclusion

Throughout this thesis, we have considered a combinatorial problem that selecting a subset S of a finite whole set N for maximizing some utility functions. This thesis focuses the combinatorial problem when a function f is non-decreasing and submodular which describes an important property in computer science and economic areas. We consider the maximization problem under a cardinality constraint, which is called the submodular function maximization problem. Moreover, real applications such as feature selection proposed by Das and Kempe [18] sometimes do not hold exact submodularity. As an extension of the submodular function maximization problem, we consider an approximately version of submodular function which is called approximately submodular function. For the both problems, "the submodular function" and "the approximately submodular function" maximization problems, many heuristic algorithms including a variety of greedy algorithms have been proposed. However, the heuristic algorithms do not satisfy the objective of some applications, that seeks an optimal solution. Therefore, this thesis focused on proposing efficient exact algorithms.

First, for the submodular function maximization problem (Chapter 2), Nemhauser and Wolsey [67] formulated the problem into a binary integer programming (BIP) problem with an exponential number of constraints. Based on the BIP formulation, Nemhauser and Wolsey proposed a constraint generation algorithm that

starts from a reduced BIP problem with a small subset of constraints taken from the constraints. Their algorithm repeats solving a reduced BIP problem while adding a new constraint at each iteration. Their algorithm often needs to solve many reduced BIP problems until obtaining an optimal solution. To overcome this difficulty, we proposed an improved constraint generation algorithm which generates a promising set of constraints at each iteration for improving upper bounds. We incorporated the improved constraint generation algorithm into a branch-and-cut algorithm to attain good upper bounds while solving a smaller number of reduced BIP problems. To improve lower bounds, we also introduced a local search algorithm for the branch-and-cut algorithm. According to the computational experiments, we confirmed that our algorithms improved the efficiency of the conventional constraint generation algorithm [67], and also performed better than the existing algorithms [13, 70].

Secondly, we considered the approximately submodular function maximization problem (Chapter 3). We first define an approximately submodular function, and we formulate it with different formulations by using a submodular ratio and an upper bound of submodular ratio [43]. By the success of the formulation, we formulated the approximately submodular function maximization problem into a BIP problem with an exponential number of constraints. Due to the BIP formulation, the conventional constraint generation algorithm [67] and the proposed algorithms, the improved constraint geration algorithm and the branch-and-cut algorithm could be applied for the problem with small modifications. According to the computational experiments with random instances, we could see that improved generation algorithm improved upper bound as well as lower bounds due to generating many feasible solutions. We also observed that the proposed algorithm performed better than the existing algorithms according to performance profiles [21] and shifted geometry means [66].

In future work, problems under many other constraints need to be considered because applications often involve with many complicated constraints. Although it is not easy for generating a new constraint for the problem under linear con-

straints, the author believes that we can generate a promising set of constraints by using a MIP solver. For the approximately version of the problem, to obtain the maximum submodular ratio has been known as NP-hard. Since a better lower bound of a submodular ratio can not be obtained easily for an original problem, we need to consider attaining a better lower bound of a submodular ratio γ for a node of a search tree. For example, an IP formulation to obtain the exact submodular needs to be considered so that a better lower bound can be obtained. By obtaining a better subodular ratio for a node, the branch-and-cut algorithm should perform better. The author will research the topics and considers proposing efficient algorithms which can be applied in wider area.

In recent years, "the submodular function" and "the approximately submodular function" maximization problems has attracted attention. In real applications, there are many situations that seek the optimal solutions due to the importance of the optimal solutions. Although we have consider an approximately submodular function, there should have other approximately versions of submodular function. If we can formulate a maximization problem with such a function into a BIP problem, the proposed algorithms in Chapter 2 can be applied for solving the problem exactly. For real applications, we need to consider a problem under many constraints not only a cardinality constraint. The author believes that the proposed algorithms based on the BIP formulation can deal with the problems under other constraints with some modifications. The author hopes that this thesis will provide some assistance to the community of the submodular function maximization problem.

References

[1] Achterberg, T. (2009). Scip: solving constraint integer programs. *Mathematical Programming Computation 1*, 1–41.

[2] Aickelin, U. and A. K. Dowsland (2004). An indirect genetic algorithm for a nurse-scheduling problem. *Computers Operations Research 31*.

[3] Angluin, D. (1988). Queries and concept learning. *Machine Learning 2*, 319–342.

[4] Bach, F. (2013). Learning with submodular functions:a convex optimization perspective. *Foundations and Trends in Machine Learning 6*, 145–373.

[5] Bai, W. and A. J. Bilmes (2018). Greed is still good: Maximizing monotone submodular+supermodular functions. https://arxiv.org/pdf/1801.07413.pdf.

[6] Bai, W., R. Iyer, K. Wei, and J. Bilmes (2016). Algorithms for optimizing the ratio of submodular functions. In *Proceedings of the 33rd International Conference on Machine Learning*, Volume 48, pp. 2751–2759.

[7] Balkanski, E., B. Mirzasoleiman, A. Krause, and Y. Singer (2016). Learning sparse combinatorial representations via two-stage submodular maximization. In *Proceedings of the 33rd International Conference on Machine Learning*, pp. 2207–2216.

[8] Bogunovic, I., J. Zhao, and V. Cevher (2018). Robust maximization of non-submodular objectives. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*.

[9] Brucker, P. (1995). *Scheduling Algorithms*. Arthena Scientific.

[10] Bshouty, H. N. and V. Feldman (2002). On using extended statistical queries to avoid membership queries. *Journal of Machine Learning Research 2*, 359–395.

[11] Buchbinder, N., M. Feldman, and J. Naor (2014). Submodular maximization with cardinality constraints. In *Proceedings of the Twenty-Fifth Annual Symposium on Discrete Algorithms (SODA'14)*, pp. 1433–1452.

[12] C., C. P. and F. Echenique (2016). *Revealed Preference Theory*. Cambride University Press.

[13] Chen, W., Y. Chen, and K. Weinberger (2015). Filtered search for submodular maximization with controllable approximation bounds. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS'15)*, Volume 38, pp. 156–164.

[14] Chvatal, V. (1983). *Linear Programming*. New York and Oxford: W. H. Freeman and Company.

[15] Conforti, M. and G. Cornuéjols (1984). Submodular set functions, matroids and the greedy algorithm: tight worst-case bounds and some generalizations of the rado-edmonds theorem. *Discrete Applied Math 7*, 251–274.

[16] Conitzer, V., T. Sandholm, and P. Santi (2005). Combinatorial auctions with k-wise dependent valuations. In *Proceedings of the American Association for Artificial Intelligence 2005 (AAAI2015)*, pp. 248–254.

[17] Crowder, H. and W. M. Padberg (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Management Science 26*, 495–509.

[18] Das, A. and D. Kempe (2011). Submodular meets spectral: greedy algorithms for subset selection, sparse approximation and dictionary selection. In *Proceedings of the 28th International Conference on Machine Learning (ICML'11)*, pp. 1057–1064.

[19] Das, A. and D. Kempe (2018). Approximate submodularity and its applications: subsetselection, sparse approximation and dictionary selection. *Journal of Machine Learning Research 19*, 1–34.

[20] Dobzinski, S., N. Nisan, and M. Schapira (2010). Approximation algorithms for combinatorial auctions with complement-free bidders. *Mathematics of Operations Research 35*(1), 1–13.

[21] Dolan, E. D. and J. J. More (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming 91*, 201–213.

[22] Drezner, Z. and W. H. Hamacher (2004). *Facility Location: Applications and Theory*. Springer-Verlag Berlin.

[23] Elenberg, R. E., G. A. Dimakis, M. Feldman, and A. Karbasi (2017). Streaming weak submodularity: Interpreting neural networks on the fly. In *Adcances in Neural Information Processing Systems 30*, pp. 4044–4054.

[24] Elenberg, R. E., R. Khanna, G. A. Dimakis, and S. Negahban (2018). Restricted strong convexity implies weak submodularity. *Ann. Statist. 46*(6B), 3539–3568.

[25] Feige, U. (2006). On maximizing welfare when utility functions are subadditive. In *Proceedings of the thirty-eighth annual ACM symposium on Theory of Computing*, pp. 41–50.

[26] Feige, U. (2009). On maximizing welfare when utility functions are subadditive. *SIAM Journal on Computing 39*, 122–142.

[27] Feige, U., V. Mirrokni, and J. Vondrák (2007). Maximizing non-monotone submodular functions. In *Proceeding of 48th IEEE FOCS*, pp. 461–471.

[28] Feldman, V. (2009). On the power of membership queries in agnostic learning. *Journal of Machine Learning Research 10*, 163–182.

[29] Goemans, X. M. and P. D. Williamson (1995). Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the Association for Computing Machinery 42*(6), 1115–1145.

[30] Goldengorin, B., G. Sierksma, A. G. Tijssen, and M. Tso (1999). The data-correcting algorithm for the minimization of supermodular functions. *Management Science 45*(11), 1539–1551.

[31] Goldman, A. S., J. M. Kearns, and E. R. Schapire (1990). Exact identification of circuits using fixed points of amplification functions (abstract). In *Proceeding of the Third Annual Workshop on Computation Learning Theory*, pp. 388.

[32] Goldstein, J., V. Mittal, J. Carbonell, and M. Kantrowitz (2000). Multi-document summarization by sentence extraction. In *Proceedings of the ANLP/NAACL Workshop on Automatic Summarization.*

[33] Golovin, D. and A. Krause (2011). Adaptive submodularity: theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research 42*, 427–486.

[34] Grötschel, M. and O. Holland (1991). Solution of large-scale symmetric travelling salesman problems. *Mathematical Programming 51*, 141–202.

[35] Halperin, E. and U. Zwick (2001). Combinatorial approximation algorithms for the maximum directed cut problem. In *Proceedings of 12th SODA*, pp. 1–7.

[36] Hassidim, A. and Y. Singer (2018). Optimization for approximate submodularity. In *Proceedings of the 32nd Conference on Neural Information Proceedings Systems (NIPS2018).*

[37] Herer, T. Y. (1999). Submodularity and the traveling salesman problem. *European Journal of Operational Research 11*, 489–508.

[38] Horel, T. (2015). Notes on greedy algorithms for submodular maximization. https://thibaut.horel.org/submodularity/notes/02-12.pdf.

[39] Horel, T. and Y. Singer (2016). Maximization of approximately submodular functions. In *Proceedings of the 30th Conf. on Neural Information Proceeding Systems (NIPS2016)*.

[40] IBM (2019). Ilog cplex optimization studio. https://www.ibm.com/products/ilog-cplex-optimization-studio.

[41] Jackson, C. J. (1994). An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. In *35th Annual Symposium on Foundations of Computer Science*, pp. 42–53.

[42] Jawaid, T. S. and L. S. Smith (2013). The maximum traveling salesman problem with submodular rewards. In *American Control Conference (ACC)*.

[43] Johnson, K., A. R. Stine, and P. D. Foster (2016). Submodularity in statistics: comparing the success of model selection methods. arXiv:1510.06301v2.

[44] Jovic, A., K. Brkic, and N. Bogunovic (2015). A review of feature selection methods with applications. In *Proceedings of 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO'15)*, pp. 1200–1205.

[45] Kawahara, Y. and K. Nagano (2015). *Machine Learning with Submodular Functions*. Kodansha (published in Japanese).

[46] Kawahara, Y., K. Nagano, K. Tsuda, and J. A. Bilmes (2009). Submodularity cuts and applications. In *Advances in Neural Information Processing Systems 26*, pp. 916–924.

[47] Kempe, D., J. Kleinberg, and E. Tardos (2003). Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)*, pp. 137–146.

[48] Khuller, S., A. Moss, and J. Naor (1999). The budgeted maximum coverage problem. *Information Processing Letters 70*(1), 39–45.

[49] Kratica, J., D. Tosic, V. Filipovic, I. Ljubic, and P. Tolla (2001). Solving the simple plant location problem by genetic algorithm. *RAIRO-Operations Research 35*(1), 127–142.

[50] Krause, A. and D. Golovin (2014). Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press.

[51] Krause, A. and C. Guestrin (2007). Near-optimal observation selection using submodular functions. In *American Association for Artificial Intelligence (AAAI)*.

[52] Krause, A., J. Leskovec, C. Guestrin, J. VanBriesen, and C. Faloustson (2008). Efficient sensor placement optimization for securing large water distribution networks. *Journal of Water Resources Planning and Management 134*, 516–526.

[53] Krause, A., A. Singh, and C. Guestrin (2008). Near-optimal sensor placements in gaussian processes: theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research 9*, 235–284.

[54] Lee, J., V. S. Mirrokni, V. Nagarajan, and M. Sviridenko (2009). Non-monotone submodular maximization under matroid and knapsack constraints. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC'09)*, pp. 323–332.

[55] Lin, H. and A. Bilmes (2012). Learning mixtures of submodular shells with application to document summarization with application to document summarization. In *Proceedings of the 28th Conf. on Uncertainty in Artificial Intelligence (UAI'12)*, pp. 479–490.

[56] Lin, H. and J. Bilmes (2010). Multi-document summarization via budgeted maximization of submodular functions. In *Proceedings of the 48th Ann. Meeting of the Association for Computational Linguistics: Human Language Technologies (HLT'10)*, pp. 912–920.

[57] Lin, H. and J. Bilmes (2011). A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (HLT'11)*, pp. 510–520.

[58] Lin, Y. (2016). Rouge: A package for automatic evaluation of summaries. In *Text Summarization Brances out: Proceedings of the ACL-04*, pp. 2207–2216.

[59] Lin, Y., W. Chen, and S. C. J. Lui (2017). Boosting information spread: an algorithmic approach. *IEEE Transactions on Computational Social Systems 5*, 344–357.

[60] Liontis, K. and E. Pitoura (2016). Boosting nodes for improving the spread of influence. https://arxiv.org/abs/1609.03478.

[61] Lovász, L. (1983). Submodular functions and convexity. In A. Bachem, M. Grotschel, and B. Korte (Eds.), *Mathematical Programming — The State of the Art*, pp. 235–257. Berlin, Heidelberg: Springer.

[62] McDonald, R. (2007). A study of global inference algorithms in multi-document summarization. In *Advances in Information Retrieval: 29th European Conference on IR Research, ECIR 2007*, pp. 557–564.

[63] Mehr, N. and R. Horowitz (2018). A submodular approach for optimal sensor placement in traffic networks. In *Annual American Control Conference (ACC)*.

[64] Minoux, M. (1978). Accelerated greedy algorithms for maximizing submodular set functions. *Lecture Notes in Control and Information Sciences 7*, 234–243.

[65] Mitrovic, M., E. Kazemi, M. Zadimoghaddam, and A. Karbasi (2018). Data

summarization at scale: a two-stage submodular approach. In *Proceedings of the 35th International Conference on Machine Learning*, pp. 3593–3602.

[66] Mittelmann, H. (2019). Decision tree for optimization software: Benchmarks for optimization software. http://plato.asu.edu/bench.html.

[67] Nemhauser, G. L. and L. Wolsey (1981). Maximizing submodular set functions: Formulations and analysis of algorithms. *Studies on Graphs and Discrete Programming 11*, 279–301.

[68] Nemhauser, G. L., L. A. Wolsey, and M. L. Fisher (1978). An analysis of approximations for maximizing submodular set functions I. *Mathematical Programming 14*(1), 265–294.

[69] Russell, S. J., , and P. Norvig (2003). *Artificial Intelligence; A Modern Approach*. Pretice Hall; 2nd ediction.

[70] Sakaue, S. and M. Ishihata (2018). Accelerated best-first search with upper-bound computation for submodular function maximization. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI'18)*, pp. 1413–1421.

[71] Shamir, E. and C. Schwartzman (1995). Learning by extended statistical queries and its relation to pac learning. In *Computation Learning Theory: Eurocolt'95*, pp. 357–366.

[72] Shanu, I., S. Goel, C. Arora, and P. Singla (2016). Exploiting sum of submodular structure for inference in very high order mrf-map problems.

[73] Soma, T. (2016). *Submodular and sparse optimization methods for machine learning and communication*. Ph. D. thesis, the University of Tokyo.

[74] Sviridenko, M. (2004). A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters 32*, 41–43.

[75] Svitkina, Z. and L. Fleischer (2010). Algorithms and lower bounds for submodular cuts and approximating submodular functions. *RIMS kokyutoku Bessatsu*, 213–232.

[76] Tuy, H. (1964). Concave programming under linear constraints. *Soviet Mathematics Doklady 5*, 1437–1440.

[77] Uematsu, N., S. Umetani, and Y. Kawahara. An efficient branch-and-cut algorithm for submodular function maximization. *Journal of the Operations Research Society of Japan 63*.

[78] Uematsu, N., S. Umetani, and Y. Kawahara (2019a, Nov.). An efficient branch-and-cut algorithm for approximately submodular function maximization. In *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 3160–3167. ©[2019] IEEE. Reprinted, with permission.

[79] Uematsu, N., S. Umetani, and Y. Kawahara (2019b). An efficient branch-and-cut algorithm for approximately submodular function maximization. https://arxiv.org/abs/1904.12682.

[80] Umeda, H. and T. Asano (2017). Nash equilibria in combinatorial auctions with item bidding by two bidders. *Journal of Information Processing 25*, 745–754.

[81] Umetani, S. (2003). *Studies on local search approaches to one dimensional cutting stock problems*. Ph. D. thesis, Kyoto University.

[82] Vondarák, J. (2010). Submodularity and curvature: the optimal algorithm. *RIMS Kokyuroku Bessatsu*, 253–266.

[83] Wei, K., R. Iyer, and J. Bilmes (2015). Submodularity in data subset selection and active learning. In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*, pp. 1954–1963.

[84] Wolsey, A. L. (1982). An analysis of the greedy algorithm for the submodular set covering problem. *Combinatorica 2*(4), 385–393.

[85] Yu, L. and H. Liu (2004). Efficient feature selection via analysis of relevance and redundancy. *Journal of machine learning research 5*, 1205–1224.

Appendices

Appendix A

Computational Results

A.1 Computational Results for Simple MIP Models

For the submodular function maximization problem in Chapter 2, Facility location (FOC) and weighted coverage (COV) problem can be formulated as simple MIP models and directly solved by MIP solvers such as CPLEX 12.8.

The facility location (LOC) problem can be formulated as the following MIP model [67].

$$\begin{aligned}
\text{maximize} \quad & \sum_{i \in M} \sum_{j \in N} g_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{j \in N} x_{ij} \leq 1, \quad && i \in M, \\
& \sum_{j \in N} y_j \leq k, && \\
& x_{ij} \leq y_j, && i \in M, j \in N, \\
& y_j \in \{0,1\}, && j \in N, \\
& x_{ij} \in \{0,1\}, && i \in M, j \in N,
\end{aligned} \tag{A.1}$$

where $y_j = 1$ means that a facility is placed at location $j \in N$, i.e., the set of locations $S \subseteq N$ corresponds to a vector $\boldsymbol{y} = (y_1, \ldots, y_n)$ satisfying $y_j = 1$ for

Table A.1: Computation time (in seconds) and average gap $(z_{\text{LP}} - z_{\text{IP}})/z_{\text{LP}} \times 100$ (%) of a MIP solver (CPLEX 12.8)

Type	n	k	Time (s)	Gap (%)
	20	5	0.22	0.06
	30	5	0.42	0.34
LOC	40	5	1.19	0.92
	50	5	1.96	1.15
	60	5	1.44	1.04
	20	8	0.15	0.00
	30	8	0.22	0.04
LOC	40	8	0.72	0.24
	50	8	0.88	0.26
	60	8	1.88	0.50
	20	5	0.02	0.02
	40	5	0.04	3.67
COV	60	5	0.08	4.96
	80	5	0.12	8.39
	100	5	0.09	7.94
	20	8	0.02	0.03
	40	8	0.04	0.07
COV	60	8	0.05	0.30
	80	8	0.28	0.82
	100	8	0.63	1.49

$j \in S$. For a given vector $\boldsymbol{y}$, an optimal vector $\boldsymbol{x}$ is given by

$$x_{ij} = \begin{cases} 1 & \text{for some } \ j \in N \ \text{ such that } \ g_{ij} = \max\limits_{h \in S} g_{ih} \\ 0 & \text{otherwise.} \end{cases} \tag{A.2}$$

The weighted coverage (COV) problem can be formulated as the following MIP model.

$$\begin{aligned} \text{maximize} \quad & \sum_{i \in M} w_i y_i \\ \text{subject to} \quad & \sum_{j \in N} a_{ij} x_j \geq y_i \quad i \in M, \\ & \sum_{j \in N} x_j \leq k, \\ & x_j \in \{0,1\}, \qquad j \in N, \\ & y_i \in \{0,1\}, \qquad i \in M, \end{aligned} \tag{A.3}$$

where $x_j = 1$ means that a sensor $j \in N$ is selected to cover items, i.e., the set of sensors $S \subseteq N$ corresponds to a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ satisfying $x_j = 1$ for $j \in S$. For a given vecor $\boldsymbol{x}$, an optimal vector $\boldsymbol{y}$ is given by

$$y_i = \begin{cases} 1 & \sum\limits_{j \in N} a_{ij} x_j \geq 1 \\ 0 & \text{otherwise.} \end{cases} \tag{A.4}$$

We have solved all LOC and COV instances of the above MIP models by a MIP solver (CPLEX 12.8). Table A.1 shows the average computation time (in seconds) and the average gap between the optimal values of MIP and its LP relaxation $(z_{\text{LP}} - z_{\text{IP}})/z_{\text{LP}} \times 100$ (%) for each class of instances. z_{IP} and z_{LP} represent the optimal values solved by an IP problem and an LP relaxation problem, respectively. According to the gap in Table A.1, we can see that the MIP solver attains a good upper bound by only solving an LP relaxation of the simple MIP formulations, while the constraint generation (CG) algorithm solves many reduced BIP problems to attain the comparable upper bounds. Therefore, the simple MIP formulation is very efficient for the LOC and COV problems. However, our algorithms can solve wider class of problems, not only problems which can be formulated as simple MIP formulations.

A.2 Settings of Computational Experiments

We set a few parameters to generate benchmark instances differently from those of Sakaue and Ishihata [70] due to the difference between the cardinality constraint and the knapsack constraint (see Section 2.4).

We set a parameter p for weighted coverage (COV) instances so that a sensor $j \in N$ randomly covers an item $i \in M$ with probability $p = 0.15$, while Sakaue and Ishihata [70] originally set the parameter $p = 0.3$. If $k \times p \geq 1$ holds, then it is possible to cover all items by k sensors with high probability, and the optimal value takes trivial $\sum_{j \in N} w_j$ for the case. We note that COV instances become rather easy for the case because the greedy algorithm often attains the optimal solutions. To make a variety of hardness for COV instances, we set the parameter $p = 0.15$ for satisfying $k \times p < 1$ for COV instances with $k = 5$ and $k \times p > 1$ for those with $k = 8$.

We set a parameter p for bipartite influence (INF) instances so that an edge $(i, j) \in A$ is randomly generated with probability $p = 0.1$, while Sakaue and Ishihata [70] originally set the parameter $p = 0.3$. Based on our preliminary computational experiments, INF instances become rather hard for the case with $p = 0.3$ under the cardinality constraint so that none of the algorithms attain an optimal solution within the time limit for those $n > 50$. We note that the submodular function maximization problem with the cardinality constraint tends to be harder than that with the knapsack constraint. We accordingly set the parameter $p = 0.1$ for INF instances. (more details in [77])

We would like to discuss situations where to solve a reduced BIP problem (2.26) at a node of a search tree is difficult within a reasonable computation time. We considered at least two ways to avoid the situation. The first way is to terminate solving the reduced BIP problem at a node and search its child node within a fixed computation time. If we do not obtain any solution at the node, we do not generate any constraint at the node and search its child node. The second way is to solve a linear programming problem of a reduced BIP problem (2.26) if a LP

solver can not solve the reduced BIP problem. A linear programming problem of the reduced BIP problem is defined as follows.

$$\begin{aligned}
\text{maximize} \quad & z \\
\text{subject to} \quad & z \leq f(S) + \sum_{i \in N \setminus S} f(\{i\} \mid S)\, y_i,\ S \in Q^+, \\
& \sum_{i \in N \setminus (S^0 \cup S^1)} y_i \leq k - |S^1|, \\
& 0 \leq y_i \leq 1, \quad i \in N \setminus (S^0 \cup S^1), \\
& y_i = 0, \qquad i \in S^0, \\
& y_i = 1, \qquad i \in S^1.
\end{aligned} \tag{A.5}$$

However, we need to pay attention to the fact that upper bounds obtained by solving a LP problem are often too loose to prune nodes of a search tree. For applying the improved constraint generation algorithm which generates a promising set of constraints at each iteration, we need to obtain an optimal solution of the reduced LP problem (A.5). We let $(y_1^{(t)}, \ldots, y_n^{(t)})$ be an optimal solution of the reduced LP problem at tth iteration. We select k largest elements with respect to the value $0 \leq y_i^{(t)} \leq 1$, and we consider a set of the elements to be a feasible solution $S^{(t)}$. We generate constraints $S' \subseteq S^{(t)} \cup S^{\natural}$, where $|S'| = k$ and a constraint $S^{\natural} \in Q$ attaining the optimal value of the reduced LP problem at kth iteration.